DMF 5 Administrator's Guide for SGI® InfiniteStorage

007–5484–007

New Features in this Guide

This guide includes the following changes:

- Support for SLES 11 SP1 on DMF servers
- New configuration parameter: `TSREPORT_OPTIONS` in the `taskgroup` stanza specifies additional options that `run_daily_tsreport.sh` will add to the end of the `tsreport` command line. (This parameter is optional). See "`taskgroup` Parameters" on page 174.
- Support for *DMF direct archiving*, which lets users manually archive files from a POSIX filesystem that is not DMF-managed (an *unmanaged filesystem*, such as Lustre) directly to secondary storage via DMF. This includes the new `dmarchive`(1) command and associated library subroutines. See:
  - "DMF Direct Archiving: Copying Unmanaged File Data to Secondary Storage" on page 13
  - "DMF Direct Archiving Requirements" on page 19
  - "DMF File Concepts" on page 21
  - "`filesystem` Object" on page 187
  - "`archive` File Requests" on page 414
  - "Filesystem Information Subroutine" on page 409
  - "`SiteArchiveFile()`" on page 441

Record of Revision

Version	Description
001	December 2008 Original publication. Supports DMF 4.0 in SGI® InfiniteStorage Software Platform (ISSP) 1.5.
002	March 2009 Supports DMF 4.1 in ISSP 1.6.
003	June 2009 Supports DMF 4.2 in ISSP 1.7.
004	September 2009 Supports DMF 4.3 in ISSP 1.8.
005	January 2010 Supports DMF 5.0 in ISSP 2.0.
006	June 2010 Supports DMF 5.1 in ISSP 2.1.
007	September 2010 Supports DMF 5.2 in ISSP 2.2.

Contents

Figures

Tables

Examples

Procedures

About This Guide

This publication documents administration of the Data Migration Facility (DMF) environment.

Related Publications

For information about this release, see the SGI® InfiniteStorage Software Platform (ISSP) release notes (`README.txt`) and the DMF release notes (`README_DMF.txt`).

The *DMF 5 Filesystem Audit Guide for SGI InfiniteStorage* describes how to solve problems with DMF should you encounter them.

Also see the following DMF man pages:

`dmf.conf`(5)
`dmarchive`(1)
`dmarenadump`(8)
`dmatsnf`(8)
`dmattr`(1)
`dmaudit`(8)
`dmatvfy`(8)
`dmcheck`(8)
`dmcollect`(8)
`dmconfig`(8)
`dmdadm`(8)
`dmdate`(8)
`dmdbcheck`(8)
`dmdbrecover`(8)
`dmdidle`(8)
`dmdskvfy`(8)
`dmdstat`(8)
`dmdstop`(8)
`dmdu`(1)
`dmdump`(8)
`dmdumpj`(8)
`dmfill`(8)
`dmfind`(1)
`dmflicense`(8)

`dmfsfree`(8)
`dmfsmon`(8)
`dmfdaemon`(8)
`dmget`(1)
`dmhdelete`(8)
`dmlockmgr`(8)
`dmmaint`(8)
`dmmigrate`(8)
`dmmove`(8)
`dmnode_admin`(8)
`dmov_keyfile`(8)
`dmov_loadtapes`(8)
`dmov_makecarts`(8)
`dmscanfs`(8)
`dmselect`(8)
`dmsnap`(8)
`dmtag`(1)
`dmusage`(8)
`dmversion`(1)
`dmvoladm`(8)
`dmxfsrestore`(8)
`dmxfsprune`(8)
`sgi_dmdu`(1)
`sgi_dmfind`(1)
`sgi_dmls`(1)
`vi`(1)
`xfsdump`(8) (Linux®)
`xfsrestore`(8) (Linux)

For information about running DMF in a high-availability (HA) Linux cluster, see the SGI HA documentation.

Also see:

- *CXFS 6 Administration Guide for SGI InfiniteStorage*
- *CXFS 6 Client-Only Guide for SGI InfiniteStorage*
- *TMF 5 Administrator's Guide for SGI InfiniteStorage*
- *OpenVault Operator's and Administrator's Guide*
- *SGI Foundation Software 2.2 Start Here*

- *XVM Volume Manager Administrator's Guide*

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`
  - 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 release notes and manuals are provided in the `noarch/sgi-isspdocs` RPM and will be installed on the system 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.
`manpage` (*x*)	Man page section identifiers appear in parentheses after man page names.
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 any 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.

- Send mail to the following address:

  SGI
  Technical Publications
  46600 Landing Parkway
  Fremont, CA 94538

SGI values your comments and will respond to them promptly.

Introduction

This chapter provides an overview of the SGI® InfiniteStorage Data Migration Facility (DMF). It discusses the following:

What Is DMF?

This section discusses the following:

Transparent Tiered-storage Management

DMF is an automated tiered-storage management system for SGI environments. As a filesystem migrator, DMF manages the capacity of online disk resources by transparently moving file data from disk to offline media. This lets you cost-effectively maintain a seemingly infinite amount of data without sacrificing accessibility for users.

DMF automatically detects a drop below the filesystem free-space threshold and then selects files for migration based on site-defined criteria, such as time of last access. DMF then migrates the file data from high-performance but more expensive online disk to levels of decreased-performance but less-expensive offline secondary storage.

Although DMF moves the file data, it leaves inodes and directories intact within the native filesystem. When a user accesses a file's data with normal operating system commands, DMF automatically recalls the file data from offline media. Because the inodes and directories are not migrated, users never need to know where the file data actually resides; migrated files remain cataloged in their original directories and are accessed as if they were still online. In fact, when using POSIX-compliant commands for filesystem inquiry, a user cannot determine whether a file is online or offline; determining the data's actual residence requires special commands or command options. The only difference users might notice is a delay in access time.

Therefore, DMF allows you to oversubscribe your online disk in a manner that is transparent to users.

Migration Targets

The secondary storage used by DMF is usually tape, although it can be any bulk-storage device accessible locally through NFS or FTP.

DMF can migrate data to the following:

- Disk
- Tape
- Another server (via NFS or FTP)
- Disk cache on slower disk and then to tape, providing multiple levels of migration using *n-tier capability*

You should migrate at least two copies of a file to prevent file data loss in the event that a migrated copy is lost. Figure 1-1 shows the concept of migrating a file from high-performance disk to multiple copies on secondary storage, and then recalling

one of the copies. The top diagram uses two tape libraries as secondary storage, the bottom diagram uses multiple tiers of secondary storage (first to lower-performance but less-expensive disk, then to inexpensive tape), plus a second copy that goes directly to tape. The file will be recalled from disk cache as long as it resides there because it is faster than recalling from tape. The file will be recalled from the second tape copy only if necessary.

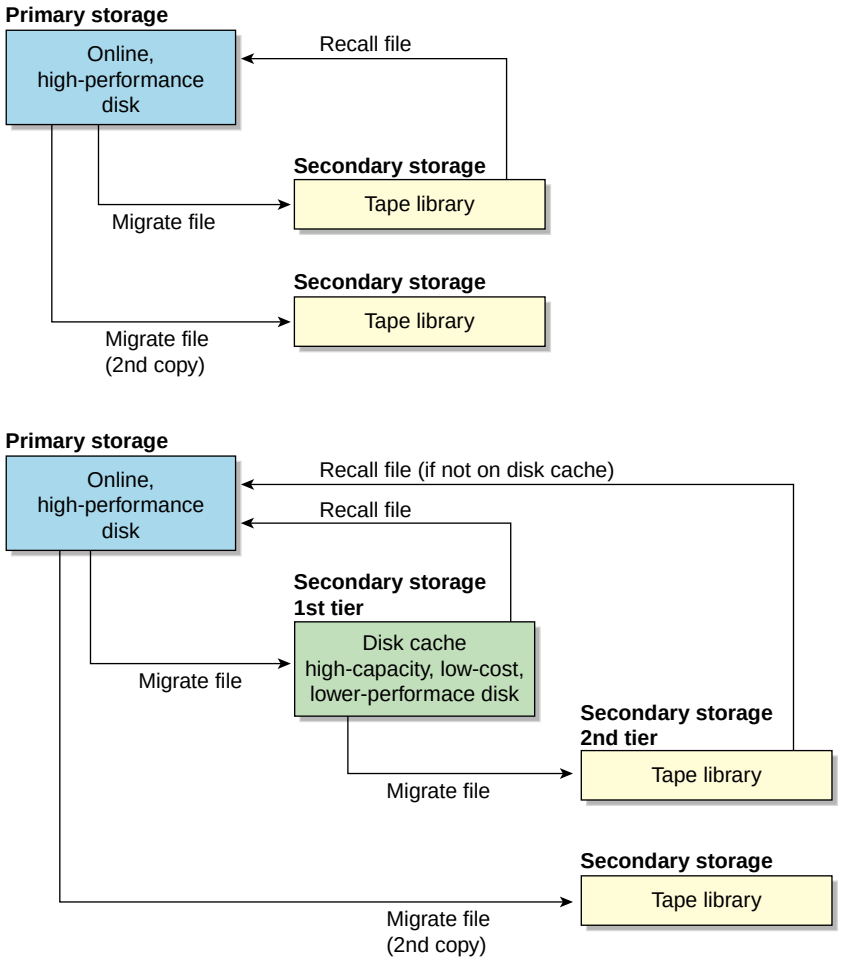

Figure 1-1 Migrating and Recalling Files

Data Flow

Figure 1-2 provides a conceptual overview of the data flow between applications and storage media.

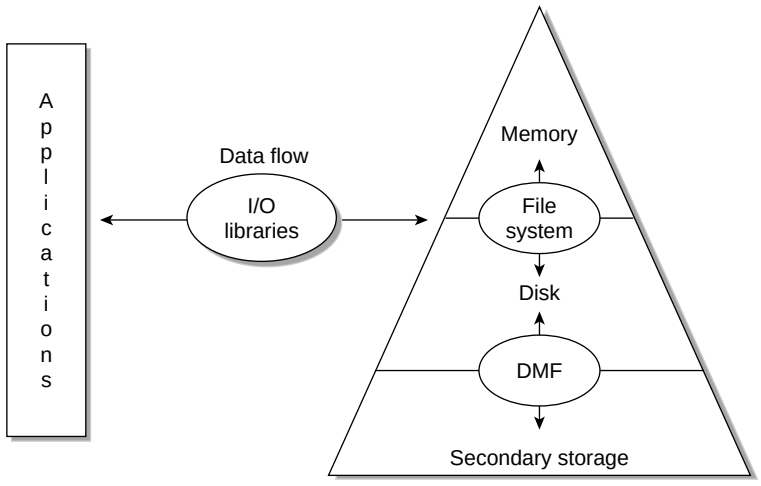

Figure 1-2 Application Data Flow

Support for Storage Management Applications

DMF supports a range of storage management applications. In some environments, DMF is used strictly to manage highly stressed online disk resources. In other environments, it is also used as an organizational tool for safely managing large volumes of offline data. In all environments, DMF scales to the storage application and to the characteristics of the available storage devices.

DMF interoperates with the following:

- Standard data export services such as Network File System (NFS) and File Transfer Protocol (FTP)
- XFS filesystems
- CXFS (clustered XFS®) filesystems

- Microsoft® Server Message Block (SMB), which is also known as the Common Internet File System (CIFS), as used by Samba when fileserving to Windows® systems

By combining these services with DMF, you can configure an SGI system as a high-performance fileserver.

Scalability and Safety

DMF transports large volumes of data on behalf of many users and has evolved to satisfy customer requirements for scalability and the safety of data. Because system interrupts and occasional storage device failures cannot be avoided, it is essential that the integrity of data be verifiable. Therefore, DMF also provides tools necessary to validate your storage environment. See "DMF Tools" on page 36.

Tape Mounting Services

When you purchase DMF, you also receive the following tape mounting services:

- OpenVault storage library management facility
- Tape Management Facility (TMF)

Parallel Data Mover Option

The individual processes that migrate and recall data to tape are known as *data mover processes*. Nodes that run data mover processes are *data movers*; this may include the DMF server node if it is configured to use *integrated data mover functionality* and, if you have purchased the *Parallel Data Mover Option*, the *parallel data mover nodes*. The DMF server and the parallel data mover nodes can each run multiple data mover processes.

As shown in Figure 1-3, the *basic DMF* product (that is, without the optional Parallel Data Mover Option) runs data mover processes on the DMF server. This allows the DMF control system to reside on a single server and minimizes the cost of a DMF implementation. Additional nodes can be installed with DMF client software (see "DMF Clients" on page 10).

Note: All nodes connect to a network. This is not shown in the following figures.

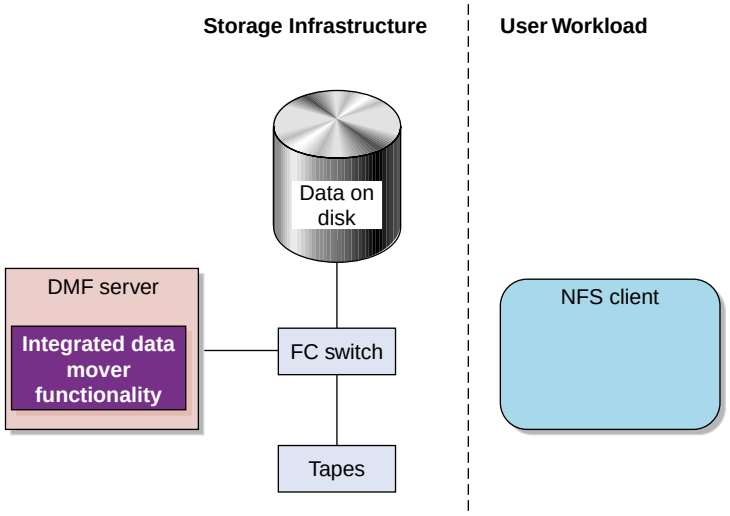

Figure 1-3 Basic DMF in an NFS Environment

Figure 1-4 shows DMF in a CXFS™ clustered filesystem environment.

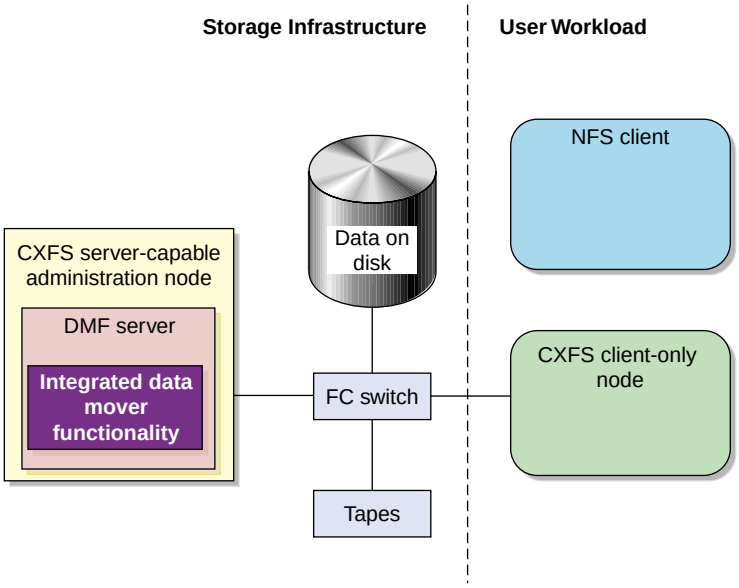

Figure 1-4 Basic DMF in a CXFS Environment

For users with higher throughput requirements, the Parallel Data Mover Option allows additional data movers to operate in parallel with the integrated data mover functionality on the DMF server, increasing data throughput and enhancing resiliency.

The parallel data mover node's dedicated function is to move data from the filesystem to offline tape storage or from offline tape storage back into the filesystem. Offloading the majority of tape I/O from the integrated data mover functionality on the DMF server improves I/O throughput performance.

Because multiple parallel data mover nodes can be used to move data, DMF can scale its I/O throughput capabilities. When one parallel data mover node hits its peak throughput capabilities, you can add another parallel data mover node to the configuration, as often as needed to improve I/O performance. Each parallel data mover node can improve overall DMF performance by up to its maximum performance. For example, if you have parallel data mover nodes that each provide up to a 2–GB/s increase, then having a configuration with three of these parallel data mover nodes would provide a net increase of up to 6 GB/s. Additional tape drives

and filesystem bandwidth may be required to realize the benefit from additional parallel data mover nodes.

Basic DMF can run in an environment with or without CXFS. If DMF is managing a CXFS filesystem, DMF will ensure that the filesystem's CXFS metadata server is the DMF server and will use metadata server relocation if necessary to achieve that configuration (see "Configure DMF Appropriately with CXFS™" on page 69). With the Parallel Data Mover Option, DMF must always run in a CXFS environment. The parallel data mover nodes are SGI ia64 and SGI x86-64 machines that are installed with the DMF Parallel Data Mover software package, which includes the required underlying CXFS software.

Note: From the CXFS cluster point of view, the DMF parallel data mover node is a CXFS client-only node. Therefore, each parallel data mover node counts towards the total CXFS cluster node count, which is 64 nodes maximum. If you have a cluster with 2 CXFS server-capable administration nodes and 2 CXFS client-only nodes installed as parallel data mover nodes, then you could have a total maximum number of 60 other CXFS client-only nodes doing normal client-only work (2+2+60=64). The parallel data mover nodes must be dedicated to DMF data mover activities; they cannot perform any other functions that would be normal for CXFS client-only nodes.

The parallel data mover node has specific hardware requirements and must have separate HBAs for disk and tape I/O. For details, see the DMF release notes. (To access the release notes, see "Obtaining Publications" at the front of this guide.)

If you choose the DMF Parallel Data Mover Option, you must use OpenVault for those drive groups that contain tape drives on parallel data mover nodes.

Figure 1-5 shows the concept of DMF using parallel data mover nodes in a CXFS cluster with only one server-capable administration node.

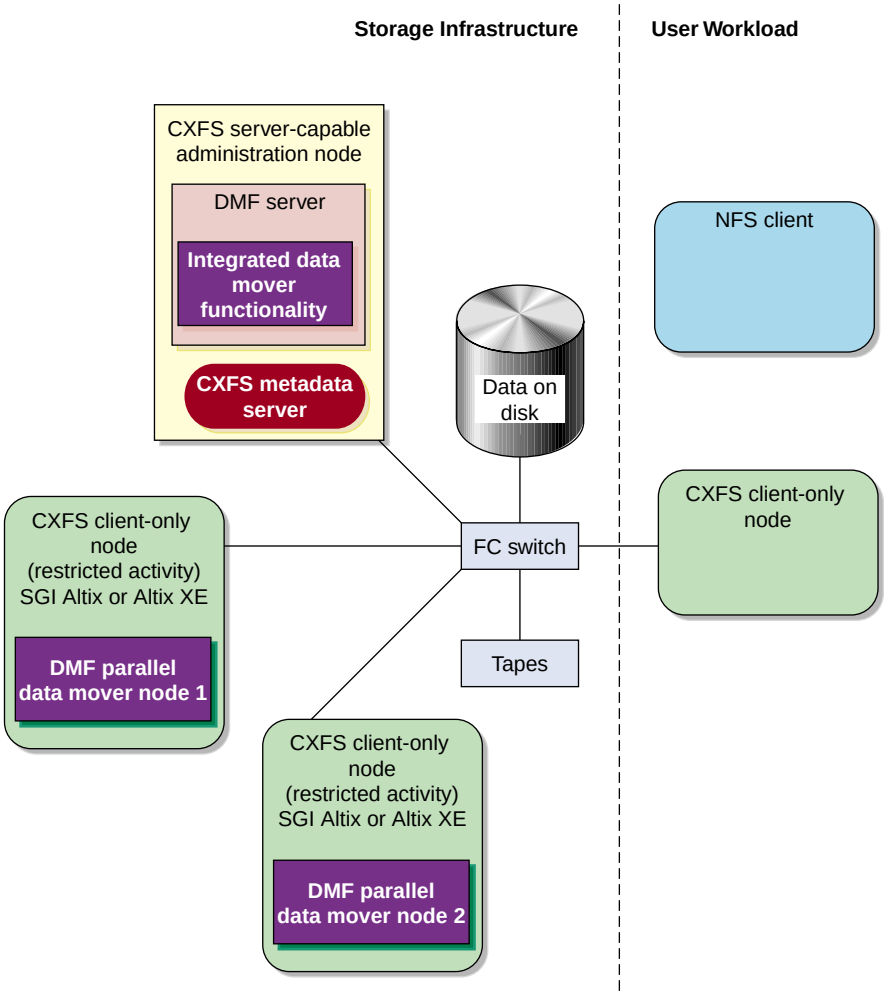

Figure 1-5 DMF with the Parallel Data Mover Option in a CXFS Environment

In a configuration with the Parallel Data Mover Option, the DMF server still provides the following services (just as with basic DMF):

- DMF administration (see "DMF Administration" on page 33)
- Backups
- Snapshots
- All I/O for data transfer to and from disks that is associated with FTP, disk, or disk cache manager (DCM) media-specific processes (MSPs) (see "How DMF Works" on page 20)
- By default, a portion of I/O for data transer to and from tape (using its integrated data mover functionality)

For more information, see Chapter 8, "Parallel Data Mover Option Configuration" on page 271.

For more information about CXFS, see *CXFS 6 Administration Guide for SGI InfiniteStorage* and *CXFS 6 Client-Only Guide for SGI InfiniteStorage*.

DMF Clients

DMF supports clients running one of the following operating systems (for the specific versions supported, see the DMF release notes):

- SGI IRIX®
- Apple® Mac OS X®
- Red Hat® Enterprise Linux® (RHEL)
- SUSE® Linux® Enterprise Server (SLES)
- Sun™ Solaris™

A subset of DMF commands are available on DMF clients. See "User Commands" on page 37.

Note: If you have DMF client platforms in your configuration, the DMF server must be configured to have the `xinetd`(8) daemon running. The `xinetd` daemon is enabled by default.

High Availability

You can run DMF in a high-availability (HA) cluster using the Linux-HA Heartbeat product. The Heartbeat product provides the infrastructure to fail over *HA resources* that survive a single point of failure. An *HA resource* is a service, associated with an IP address, that is managed by Heartbeat. A *resource group* is a set of resources that must be managed and failed over as a set.

Caution: If you run DMF in an HA cluster, there are some configuration requirements and administrative procedures (such as stopping DMF) that differ from the information in this guide. For more information, see the SGI documentation.

Managing DMF

DMF provides a set of tools to help you configure, monitor, and manage the DMF system. See "DMF Tools" on page 36.

DMF Manager is a web-based tool you can use to configure DMF, deal with day-to-day DMF operational issues, and focus on work flow. DMF Manager is useful for all DMF customers from enterprise to HPC and is available via the Firefox® and Internet Explorer® web browsers.

At a glance, you can see if DMF is operating properly. An icon in the upper-right corner indicates if DMF is up (green) or down (upside down and red). If DMF requires attention, DMF Manager makes actions available to identify and resolve problems. The tool volunteers information and provides context-sensitive online help. DMF Manager also displays performance statistics, allowing you to monitor DMF activity, filesystems, and hardware.

Figure 1-6 is an example of the **Overview** panel. It shows that DMF is up (green icon), that it has some warnings that may require action (yellow icon), and that the `/dmi_fs2` filesystem is related to the `volume1` and `volume2` tape volume groups (VGs).

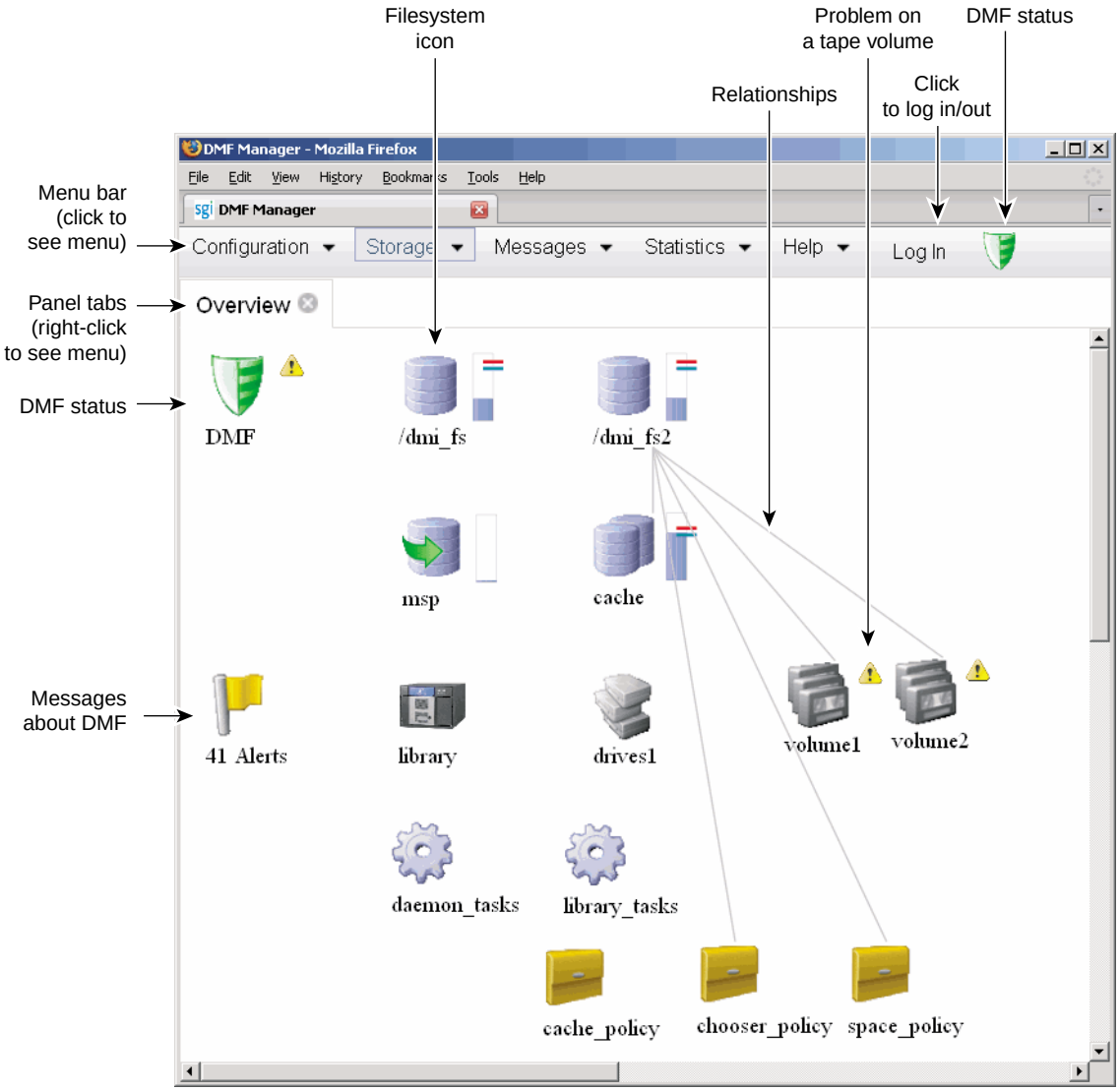

Figure 1-6 DMF Manager

For details, see Chapter 6, "Using DMF Manager" on page 101.

DMF Client Commands Web Service

DMF provides access to a subset of the DMF client functions via the DMF client Simple Object Access Protocol (SOAP) web service. For more information, see Chapter 15, "DMF Client SOAP Service" on page 365.

DMF Direct Archiving: Copying Unmanaged File Data to Secondary Storage

If your primary workspace is a POSIX filesystem that is not DMF-managed (an *unmanaged filesystem*, such as Lustre™ filesystems), *DMF direct archiving* lets you manually copy files directly to secondary storage via DMF by using the `dmarchive`(1) command. DMF copies the file data to secondary storage while placing the metadata in a visible DMF-managed filesystem, as shown in Figure 1-7.

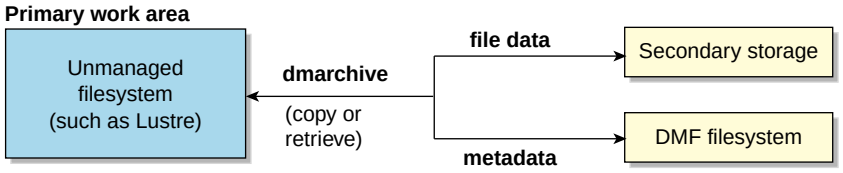

Figure 1-7 Archiving Files from an Unmanaged Filesystem to Secondary Storage

This is a more-efficient process than first manually copying the file to a DMF-managed filesystem and then migrating it. Without the `dmarchive` command, you would have to first copy the file to a DMF-managed filesystem (such as `/dmf`) and then migrate the files. For example:

```
% cp -a /lustrefs/work /dmf
% dmput /dmf/work/*
```

However, using `dmarchive`, you can achieve the same results with a single command:

```
% dmarchive -a /lustrefs/work /dmf
```

The file data will be copied directly to DMF secondary storage and only the file metadata will be copied to the DMF-managed filesystem (`/dmf`). On retrieval, the data is copied directly from DMF secondary storage to the DMF-unmanaged filesystem. The `dmarchive` method is therefore more efficient because it requires less time, bandwidth, and filesystem capacity.

Figure 1-8 shows that the Lustre server is serving the `/lustrefs/work` filesystem, which is mounted on both the DMF server and the DMF client, allowing you to run the `dmarchive` command. The DMF server is managing the `/dmf` filesystem, which is NFS-mounted at `/mnt/dmfusr1` on the DMF client.

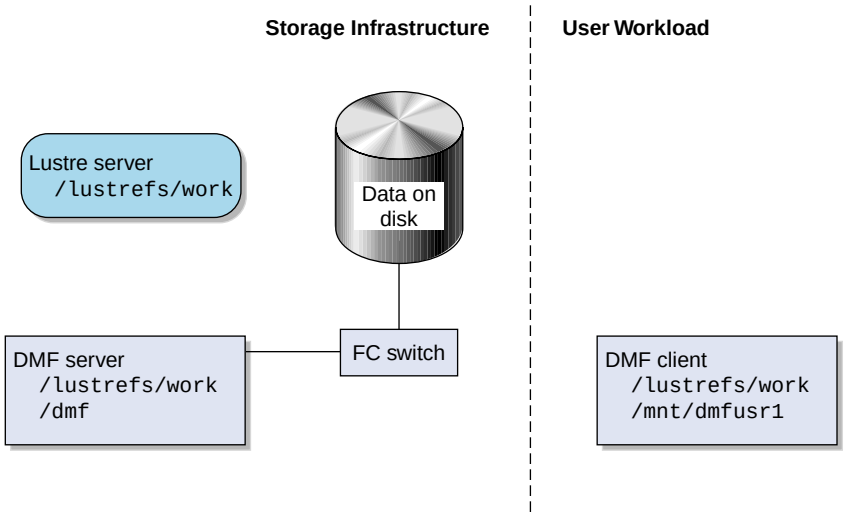

Figure 1-8 DMF Direct Archiving

DMF Requirements

Note: See the ISSP release notes and the DMF release notes for the supported kernels, update levels, service pack levels, software versions, libraries, and tape devices.

This section discusses the following:

Server Node Requirements

A DMF server node requires the following:

- SGI ia64 or SGI x86-64 hardware

  Note: In a DMF configuration with CXFS or HA, the DMF server must run on either **all SGI ia64 systems** or **all SGI x86-64 systems**.

- SUSE Linux Enterprise Server Service Pack 1 (SLES 11 SP1) and SGI Foundation Software as documented in the ISSP release notes
- DMF server software and associated products distributed with the ISSP release

Parallel Data Mover Node Requirements

DMF parallel data mover nodes require the following:

- SGI ia64 or SGI x86_64 hardware
- Same operating system and SGI Foundation Software versions as the DMF server and CXFS server
- DMF parallel data mover node software and associated products distributed with the ISSP release (which includes the required underlying CXFS client-only software)

If you use the Parallel Data Mover Option, you must use OpenVault for those drive groups that contain tape drives on parallel data mover nodes. See "Parallel Data Mover Option" on page 5.

Device Requirements

DMF supports the following devices:

- Fibre Channel tapes and tape libraries.
- SCSI low-voltage differential (LVD) tapes and tape libraries. If you have a high-voltage differential (HVD) tape or tape library that you want to use for DMF, you must contact SGI Professional Services for assistance in obtaining the appropriate HVD-LVD converter.

Note: The LVD requirement is only for tapes and tape libraries. It does not apply to HVD disk.

- Media transports and robotic automounters. Generally, DMF can be used with any transport and automounter that is supported by either OpenVault or TMF.

Tape Mounting Service Requirements

OpenVault requires `ksh`, not `pdksh`.

TMF has no DMF-specific requirements.

License Requirements

DMF is a licensed product. See Chapter 2, "DMF Licensing" on page 47.

DMAPI Requirement

For filesystems to be managed by DMF, they must be mounted with the DMAPI interface enabled. See "Filesystem Mount Options" on page 85.

SAN Switch Zoning or Separate SAN Fabric Requirement

Tape drives must be visible only from the active DMF server, the passive DMF server (if applicable), and the parallel data mover nodes. The tape drives must not be visible to any other nodes. You must use either independent switches (in a separate SAN fabric) or independent switch zones for CXFS/XVM volume paths and DMF tape drive paths.

Warning: If the tape drives are visible to any other nodes, such as CXFS client-only nodes (other than those that are dedicated to being parallel data mover nodes), data can become corrupted or overwritten.

DMF requires independent paths to tape drives so that they are not fenced by CXFS. The ports for the tape drive paths on the switch must be masked from fencing in a CXFS configuration.

XVM must not failover CXFS filesystem I/O to the paths visible through the tape HBA ports when Fibre Channel port fencing occurs.

DMF Manager Requirements

DMF Manager has the following requirements:

- The DMF Manager software is installed on the DMF server node.
- One of the following web browsers:
  - Firefox 2.*n* through Firefox 3.6 *(Firefox is the preferred browser)*
  - Internet Explorer 7.*n* (7.0 or later) and Internet Explorer 8.
- Before saving or applying configuration changes, you must make and mount the DMF administrative filesystems. See "Configure DMF Administrative Filesystems and Directories Appropriately" on page 64.

DMF Client SOAP Requirements

To use the DMF Client SOAP capability, the software must be installed on the DMF server node.

DMF Direct Archiving Requirements

DMF direct archiving has the following requirements:

- The unmanaged filesystem must be visible and mounted in the same location on the DMF server and any parallel DMF mover nodes. (The DMF server need not be the server of the unmanaged filesystem; for example, the DMF server need not be the Lustre server.)
- The unmanaged filesystem must be visible to DMF clients from which you want to run the `dmarchive`(1) command, but may have the filesystem mounted on a different mount point.
- The unmanaged filesystem must be mounted on the DMF server and any parallel DMF mover nodes so that the `root` user is able to access the filesystem with `root` privileges (that is, with `root` *squashing* disabled).
- The unmanaged filesystem must be fast enough to permit efficient streaming to/from secondary storage. If this is not the case, the tape speed could be so slow as to render DMF useless; in that situation, copying the file to a DMF-managed filesystem via `cp`(1) and migrating the file may be a better option.

If a filesystem does not meet these requirements, it should not be added to `dmf.conf` as an unmanaged filesystem.

How DMF Works

This section discusses the following:

DMF File Concepts

DMF regards files as being one of the following:

- *Regular files* are user files residing only on online disk.
- *Migrating files* are files whose offline copies are in progress.
- *Migrated files* are files that have one or more complete offline copies and no pending or incomplete offline copies. Migrated files one of the following:
  - *Dual-state files* are files where the data resides both on online disk and on secondary storage
  - *Offline files* are files where the data is no longer on online disk
  - *Unmigrating files* are previously offline files in the process of being recalled to online disk
  - *Partial-state files* are files with some combination of dual-state, offline, and/or unmigrating regions

DMF does not migrate pipes, directories, or UNIX® or Linux special files.

Like a regular file, a migrated file has an inode. An offline file or a partial-state file requires the intervention of the DMF daemon to access its offline data; a dual-state file is accessed directly from the online disk copy.

The operating system informs the DMF daemon when a migrated file is modified. If anything is written to a migrated file, the offline copy is no longer valid, and the file becomes a regular file until it is migrated again.

If you are using DMF direct archiving to copy files from a filesystem that is not DMF-managed, *archiving files* are files where the original resides on an unmanaged filesystem (one not managed by DMF, such as Lustre) and whose offline copies are in progress. When the process completes, the files are offline files.

Offline Media

Offline media is the destination of all migrated data and is managed by daemon-like DMF components called the *library server* (LS) and the *media-specific process (MSP)*:

- *LS* (`dmatls`) transfers to and from magnetic tape in a tape library (also known as a *robotic library* or *silo*).

- *FTP MSP* (`dmftpmsp`) uses the file transfer protocol to transfer to and from disks of another system on the network.
- *Disk MSP* (`dmdskmsp`) uses a filesystem mounted on the DMF server itself. This can be a local filesystem or a remote filesystem mounted through NFS or similar filesharing protocol.
- *Disk cache manager (DCM) MSP* is the disk MSP configured for *n*–tier capability by using a dedicated filesystem as a cache. DMF can manage the disk MSP's storage filesystem and further migrate it to tape, thereby using a slower and less-expensive dedicated filesystem as a cache to improve the performance when recalling files. The filesystem used by the DCM must be a local XFS or CXFS filesystem.

A site can use any combination of LS, disk MSP, FTP MSP, or DCM MSP; they are not mutually exclusive.

Figure 1-9 summarizes these concepts.

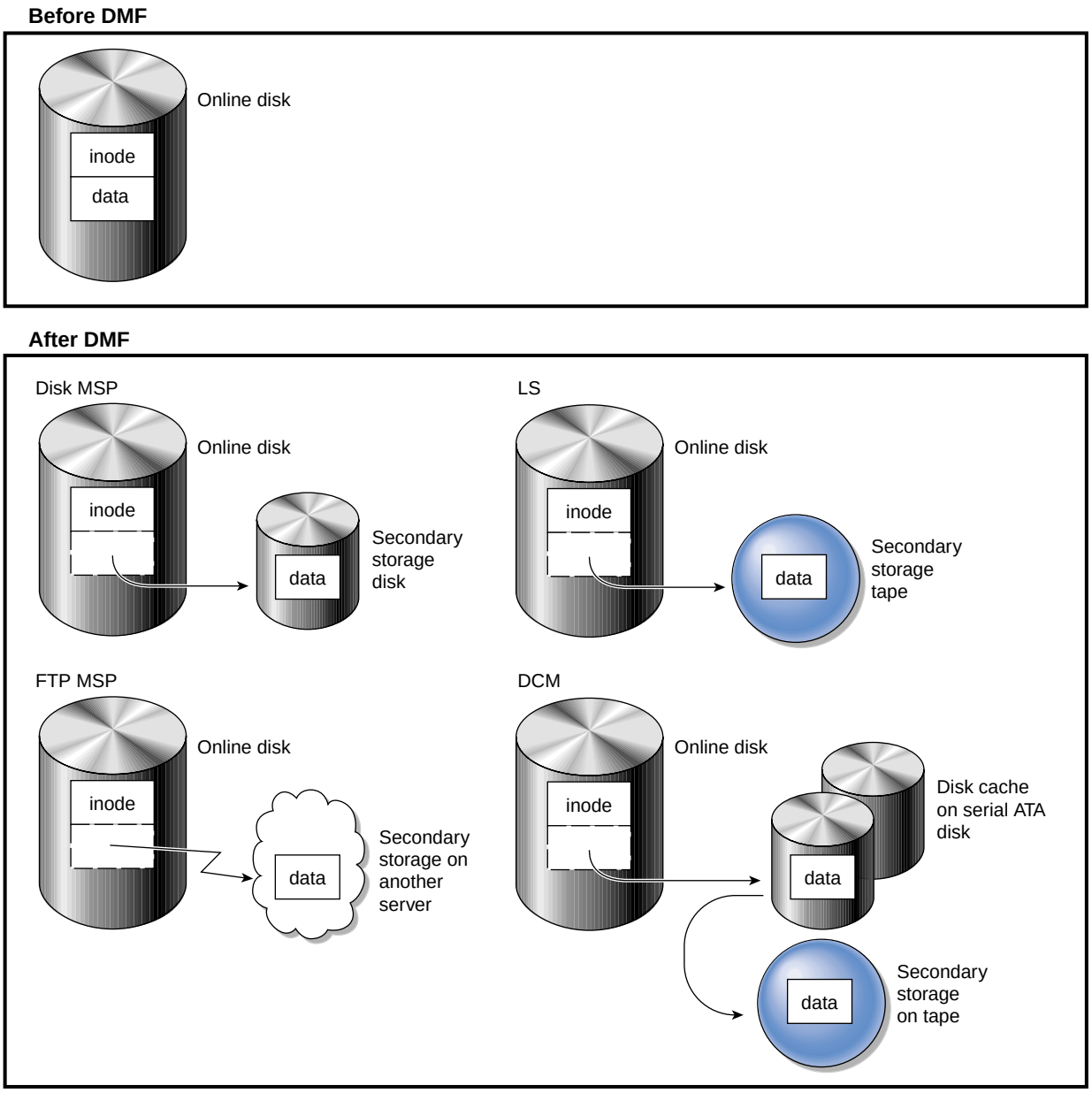

Figure 1-9 DMF Mechanisms

DMF Databases

The DMF daemon keeps track of migrated files in its *daemon database*. The key to each file is its *bit-file identifier (BFID)*. For each migrated file, the daemon assigns a BFID that is stored in the file's inode. There is a daemon database record for each copy of a migrated file.

The daemon database also contains information such as the following:

- The MSP or volume group (VG) name
- The MSP or VG key for each copy of a migrated file

When you use an MSP, the daemon database contains all of the information required to track a migrated file.

If you use an LS, there is also the *LS database*, which contains two tables of records:

- *Catalog (CAT) records* track the location of migrated data on tape volumes. There is one CAT record for each migrated copy of a file. If a migrated copy is divided between two physical media, there will be a CAT record for each portion or *chunk*. See "CAT Records" on page 306.
- *Volume (VOL) records* contain information about the tape volumes. There is one VOL record for each volume. See "VOL Records" on page 306.

Detailed information about the DMF databases and their associated utilities is provided in "CAT Records" on page 306 and "VOL Records" on page 306.

Note: The databases consist of multiple files. However, these are not text files and cannot be updated by standard utility programs. See "Database Backups" on page 361.

For information about the OpenVault database, see *OpenVault Operator's and Administrator's Guide*.

Migrating a File

As a DMF administrator, you determine how disk space capacity is handled by doing the following:

- Selecting the filesystems that DMF will manage
- Specifying the volume of free space that will be maintained on each filesystem

Space management begins with a list of user files that you rank according to your own criteria. File size and file age are among the most common ranking criteria.

File migration occurs in two stages:

- Stage One: A file is copied (*migrated*) to secondary storage.
- Stage Two: After the copy is secure, the file is eligible to have its data blocks released (this usually occurs only after a minimum space threshold is reached).

A file that has one or more complete offline copies and no pending or incomplete offline copies is called *migrated*. A file that is migrated but whose data blocks have not yet been released is called a *dual-state file*; its data exists both online and offline, simultaneously. After a file's data blocks have been released, the file is called an *offline file*.

You choose both the percentage of filesystem volume to migrate and the volume of free space. You as the administrator can trigger file migration or file owners can issue manual migration requests.

A file is migrated when the automated space management controller `dmfsfree`(8) selects the file or when an owner requests that the file be migrated by using the `dmput`(1) command.

When the daemon receives a request to migrate a file, it does the following:

1. Adjusts the state of the file
2. Ensures that the necessary MSP or VGs are active
3. Sends a request to the MSPs or VGs, who in turn copy data to the offline storage media

When the MSPs or VGs have completed the offline copies, the daemon marks the file as migrated in its database and changes the file to dual-state. If the user specifies the `dmput -r` option, or if `dmfsfree` requests that the file's space be released, the daemon releases the data blocks and changes the user file state to offline.

For more information, see the `dmput`(1) man page.

Recalling a Migrated File

When a migrated file must be recalled, a request is made to the DMF daemon. The daemon selects an MSP or VG from its internal list and sends that MSP/VG a request

to recall a copy of the file. If more than one MSP or VG has a copy, the first one in the list is used. (The list is created from the configuration file.)

After a user has modified or removed a migrated file, its BFID is *soft-deleted*, meaning that it is logically deleted from the daemon database. This is accomplished by setting the delete date field in the database to the current date and time for each entry referring to the modified or removed file.

A file is *hard-deleted* when its BFID is completely removed from the DMF databases. You can configure DMF to automatically perform hard-deletes. This is done using the `run_hard_delete.sh` task, which uses the `dmhdelete`(8) utility.

The soft-delete state allows for the possibility that the filesystem might be restored after the user has removed a file. When a filesystem is reloaded from a dump image, it is restored to a state at an earlier point in time. A file that had been migrated and then removed might become migrated again due to the restore operation. This can create serious problems if the database entries for the file have been hard-deleted. In this case, the user would receive an error when trying to open the file because the file cannot be retrieved.

Do not hard-delete a database entry until after you are sure that the corresponding files will never be restored. Hard-delete requests are sent to the relevant MSPs and VGs so that copies of the file can be removed from media. For a VG, this involves volume merging.

File Regions and Partial-State Files

DMF-managed files can have different residency states (online or offline) for different regions of a file. A *region* is a contiguous range of bytes that have the same residency state. This means that a file can have one region that is online for immediate access and another region that is offline and must be recalled to online media in order to be accessed.

DMF allows for multiple distinct file regions. A file that has more than one region is called a *partial-state* file. A file that is in a *static state* (that is, not currently being migrated or unmigrated) can have multiple online and offline regions. You can use the `MAX_MANAGED_REGIONS` parameter to configure the maximum number of file regions that DMF will allow on a file. You can set this parameter on a per-filesystem basis.

Note: You should use `MAX_MANAGED_REGIONS` cautiously. If set capriciously, filesystem scan times can increase greatly. For details about using `MAX_MANAGED_REGIONS`, see "`filesystem` Object" on page 187.

Partial-state files provide the following capabilities:

- *Accelerated access to first byte*, which allows you to access the beginning of an offline file before the entire file has been recalled.
- *Partial-state file online retention*, which allows you to keep a specific region of a file online while freeing the rest of it (for example, if you wanted to keep just the beginning of a file online). See "`ranges` clause" on page 206.
- *Partial-state file recall*, which allows you to recall a specific region of a file without recalling the entire file. For more information, see the `dmput`(1) and `dmget`(1) man pages.

To turn off the partial-state file feature, set the `PARTIAL_STATE_FILES` daemon configuration parameter to `off`.

For additional details, see Appendix F, "Considerations for Partial-State Files" on page 471.

Ensuring Data Integrity

DMF provides capabilities to ensure the integrity of offline data. For example, you can have multiple MSPs or VGs with each managing its own pool of media volumes. Therefore, you can configure DMF to copy filesystem data to multiple offline locations.

DMF stores data that originates in a CXFS or XFS filesystem. Each object stored corresponds to a file in the native filesystem. When a user deletes a file, the inode for that file is removed from the filesystem. Deleting a file that has been migrated begins the process of invalidating the offline image of that file. In the LS, this eventually creates a gap in the migration medium. To ensure effective use of media, the LS provides a mechanism for reclaiming space lost to invalid data. This process is called *volume merging*.

Much of the work done by DMF involves transaction processing that is recorded in databases. The DMF databases provide for full transaction journaling and employ two-phase commit technology. The combination of these two features ensures that DMF applies only whole transactions to its databases. Additionally, in the event of an

unscheduled system interrupt, it is always possible to replay the database journals in order to restore consistency between the DMF databases and the filesystem. DMF utilities also allow you to verify the general integrity of the DMF databases themselves.

See "DMF Administration" on page 33 for more information.

DMF Architecture

DMF consists of the DMF daemon and one or more MSPs or LSs. The DMF daemon accepts requests to migrate filesystem data from the DMF administrator or from users, and communicates with the operating system kernel to maintain a file's migration state in that file's inode.

The DMF daemon is responsible for dispensing a unique bit-file identifier (BFID) for each file that is migrated. The daemon also determines the destination of migration data and forms requests to the appropriate MSP/LS to make offline copies.

The MSP/LS accepts requests from the DMF daemon. For outbound data, the LS accrues requests until the volume of data justifies a volume mount. Requests for data retrieval are satisfied as they arrive. When multiple retrieval requests involve the same volume, all file data is retrieved in a single pass across the volume.

DMF uses the DMAPI kernel interface defined by the Data Management Interface Group (DMIG). DMAPI is also supported by X/Open, where it is known as the *XDSM standard*.

Figure 1-10 illustrates the basic DMF architecture. Figure 1-11 shows the architecture of the LS.

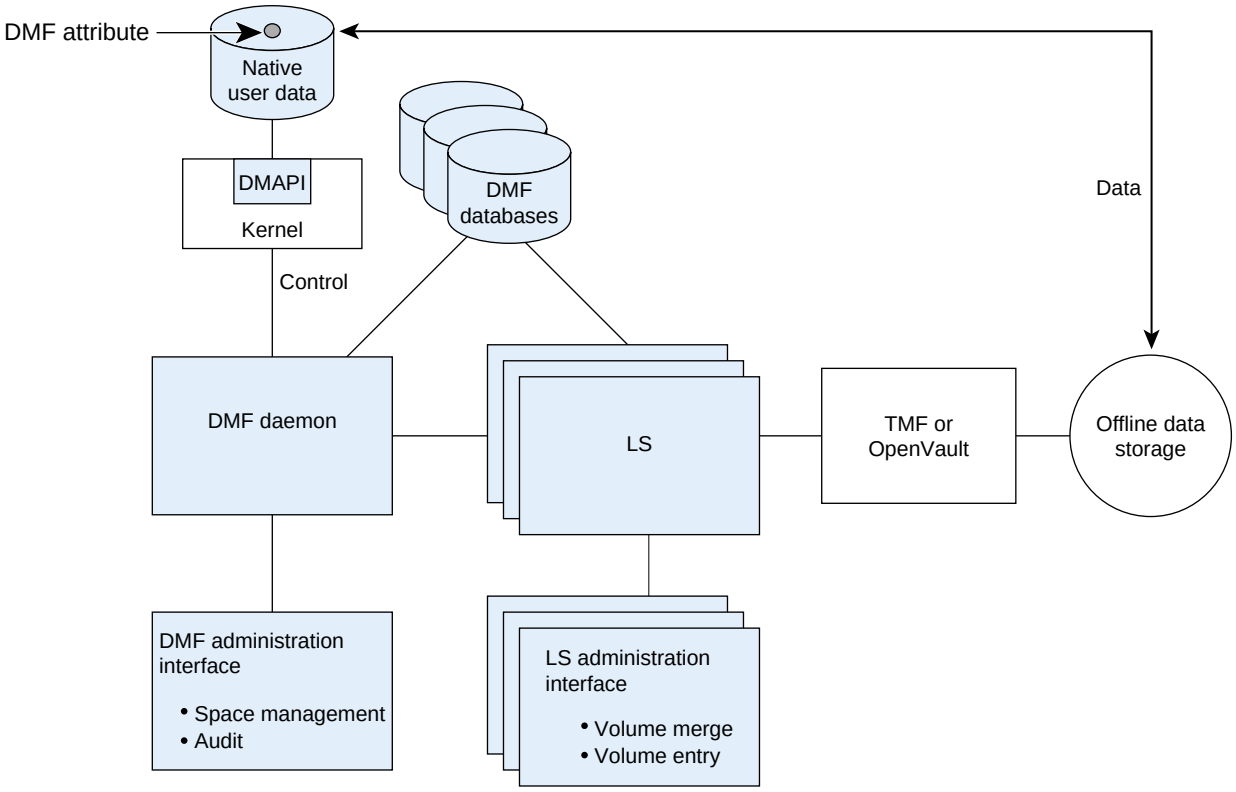

Figure 1-10 Basic DMF Architecture

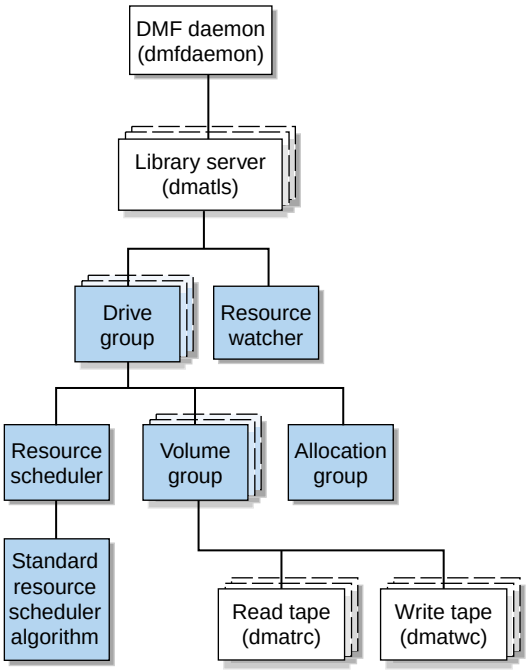

Figure 1-11 Library Server Architecture

There is one LS process (`dmatls`) per tape library, which maintains a database that all of its components share. The entities in the shaded boxes in Figure 1-11 are internal components of the `dmatls` process. Their functions are as follows:

Drive group

The drive group is responsible for the management of a group of interchangeable tape drives located in the tape library. These drives can be used by multiple VGs (see *volume group* below) and by non-DMF processes, such as backups and interactive users. However, in the latter cases, the drive group has no management involvement; the mounting service (TMF or OpenVault) is responsible for ensuring that these possibly competing uses of the tape drives do not interfere with each other.

	The main tasks of the drive group are to: • Monitor tape I/O for errors • Attempt to classify the errors as volume, drive, or mounting service problems • Take preventive action
Volume group	The VG holds at most one copy of user files on a pool of tape volumes, of which it has exclusive use. It can use only the tape drives managed by a single drive group.
Allocation group	The allocation group is really a special type of VG, used to hold a communal pool of empty tapes. These tapes can be transferred to a VG as they are needed, and can be returned when empty again. Use of an allocation group is optional.
Resource scheduler	In a busy environment, it is common for the number of drives requested by VGs to exceed the number available. The purpose of the resource scheduler is to decide which VGs should have first access to drives as they become available, and which should wait, and to advise the drive group of the result. The DMF administrator can configure the resource scheduler to meet site requirements.
Standard resource scheduler algorithm	These routines are an internal component of the `dmatls` process. Standard algorithms are provided with DMF.
Resource watcher	The resource watcher monitors the activity of the other components and frequently updates files that contain data of use to the administrator. These are usually HTML files viewable by a web browser, but can also be text files designed for use by `awk` or `perl` scripts.

The `dmatrc` and `dmatwc` processes are called the *read children* and *write children*. They are created by VGs to perform the actual reading and writing of tapes. Unlike most of the other DMF processes that run indefinitely, these processes are created as needed, and are terminated when their specific work has been completed.

Media transports and robotic automounters are also key components of all DMF installations. Generally, DMF can be used with any transport and automounter that is supported by either OpenVault or TMF. Additionally, DMF supports *absolute block*

positioning, a media transport capability that allows rapid positioning to an absolute block address on the tape volume. When this capability is provided by the transport, positioning speed is often three times faster than that obtained when reading the volume to the specified position.

Capacity and Overhead

DMF has evolved in production-oriented, customer environments. It is designed to make full use of parallel and asynchronous operations, and to consume minimal system overhead while it executes, even in busy environments in which files are constantly moving online or offline. Exceptions to this rule will occasionally occur during infrequent maintenance operations when a full scan of filesystems or databases is performed.

The capacity of DMF is measured in several ways, as follows:

- Total number of files. The daemon database addressing limits the size of the daemon database to approximately 4 billion entries. There is one database entry for each copy of a file that DMF manages. Therefore, if a site makes two copies of each DMF-managed file, DMF can manage approximately 2 billion files.
- Total volume of data. Capacity in data volume is limited only by the physical environment and the density of media.
- Total volume of data moved between online and offline media. The number of tape drives configured for DMF, the number of tape channels, and the number of disk channels all figure highly in the effective bandwidth. In general, DMF provides full-channel performance to both tape and disk.
- Storage capacity. DMF can support any file that can be created on the CXFS or XFS filesystem being managed.

DMF Administration

This section discusses the following aspects of DMF administration:

Initial Planning

DMF manages two primary resources:

- Pools of offline media
- Free space on native filesystems

You can configure DMF to manage those resources in a variety of environments, including the following:

- Support of batch and interactive processing in a general-purpose environment with limited disk space
- Dedicated fileservers
- Lights-out operations

As the DMF administrator, you must evaluate the environment in which DMF will run. You should plan for a certain capacity, both in the number of files and in the volume of data. You should also estimate the rate at which you will be moving data between the DMF store of data and the native filesystem. You should select autoloaders and media transports that are suitable for the data volume and delivery rates you anticipate.

Installation and Configuration

You will install the DMF server software (which includes the software for TMF and OpenVault) from the SGI InfiniteStorage Software Platform media.

To configure DMF, you must define a set of parameters in the DMF configuration file (`/etc/dmf/dmf.conf`). See Chapter 4, "Installing and Configuring the DMF Environment" on page 81.

To make site-specific modifications to DMF, see "Customizing DMF" on page 93.

Recurring Administrative Duties

DMF requires that you perform recurring administrative duties in the following areas:

- "Free Space Management" on page 34
- "File Ranking" on page 34
- "Offline Data Management" on page 34
- "Data Integrity and Reliability" on page 35

Note: You can use tasks that automate these duties. A *task* is a process initiated on a time schedule you determine, similar to a `cron`(1) job. Tasks are defined with configuration file parameters and are described in detail in "`taskgroup` Object" on page 170 and "LS Tasks" on page 240.

Free Space Management

You must decide how much free space to maintain on each managed filesystem. DMF has the ability to monitor filesystem capacity and to initiate file migration and the freeing of space when free space falls below the prescribed thresholds. See Chapter 10, "Automated Space Management" on page 279.

File Ranking

You must decide which files are most important as migration candidates. When DMF migrates and frees files, it selects files based on criteria you chose. The ordered list of files is called the *candidate list*. Whenever DMF responds to a critical space threshold, it builds a new migration candidate list for the filesystem that reached the threshold. See "Generating the Candidate List" on page 280.

Offline Data Management

DMF offers the ability to migrate data to multiple offline locations. Each location is managed by a separate MSP or VG and is usually constrained to a specific type of medium.

Complex strategies are possible when using multiple MSPs, LSs, or VGs. For example, short files can be migrated to a device with rapid mount times, while long files can be routed to a device with extremely high density.

You can describe criteria for MSP or VG selection. When setting up a VG, you assign a pool of tapes for use by that VG. The `dmvoladm`(8) utility provides management of the VG media pools.

You can configure DMF to automatically merge tapes that are becoming *sparse*—that is, full of data that has been deleted by the owner. With this configuration (using the `run_merge_tapes.sh` task), the media pool is merged on a regular basis in order to reclaim unusable space.

Recording media eventually becomes unreliable. Sometimes, media transports become misaligned so that a volume written on one cannot be read from another. The following utilities support management of failing media:

- `dmatsnf`(8) is used to check a DMF volume for flaws
- `dmatread`(8) is used for recovering data

Additionally, the volume merge process built into the LS is capable of effectively recovering data from failed media.

Chapter 13, "Media-Specific Processes and Library Servers" on page 301, provides more information on administration.

Data Integrity and Reliability

To maintain the integrity and reliability of data managed by DMF, you must do the following:

- Run backups. DMF moves only the data associated with files, not the file inodes or directories, so you must still run filesystem backups in order to preserve the metadata associated with migrated files and their directories. You can configure DMF to automatically run backups of your DMF-managed filesystems. See "Back Up Migrated Filesystems and DMF Databases" on page 75.

  The `xfsdump` and `xfsrestore` utilities understand when a file is migrated. The `xfsdump` utility has an option that allows for dumping only files that are not migrated. Files that are dual-state or offline have only their inodes backed up.

  You can establish a policy of migrating 100% of DMF-managed filesystems, thereby leaving only a small volume of data that the dump utility must record.

This practice can greatly increase the availability of the machine on which DMF is running because, generally, dump commands must be executed in a quiet environment.

You can configure the `run_full_dump.sh` and `run_partial_dump.sh` tasks to ensure that all files have been migrated. These tasks can be configured to run when the environment is quiet.

- Configure DMF to automatically run `dmaudit` to examine the consistency and integrity of the databases it uses. DMF databases record all information about stored data. The DMF databases must be synchronized with the filesystems that DMF manages. Much of the work done by DMF ensures that the DMF databases remain aligned with the filesystems.

  You can configure DMF to periodically copy the databases to other devices on the system to protect them from loss (using the `run_copy_databases.sh` task). This task also uses the `dmdbcheck` utility to ensure the integrity of the databases before saving them.

  DMF uses journal files to record database transactions. Journals can be replayed in the event of an unscheduled system interrupt that causes database corruption. You must ensure that journals are retained in a safe place until a full backup of the DMF databases can be performed.

  You can configure the `run_remove_logs.sh` and `run_remove_journals.sh` tasks to automatically remove old logs and journals, which will prevent the DMF `SPOOL_DIR` and `JOURNAL_DIR` directories from overflowing.

- You can configure the `run_hard_delete.sh` task to automatically perform hard-deletes to remove expired daemon database entries and release corresponding MSP or VG space, resulting in logically less active data. See "Recalling a Migrated File" on page 25.

DMF Tools

The DMF administrator has access to a wide variety of commands for controlling DMF. Users require just a few commands that let file owners affect the manual storing and retrieval of their data. This section discusses the following:

Note: The functionality of some of these commands can be affected by site-defined policies; see "Customizing DMF" on page 93.

The FTP MSP uses no special commands, utilities, or databases.

User Commands

End users can run the following commands on DMF clients:

Command	Description
`dmattr`(1)	Displays whether files are migrated or not by returning a specified set of DMF attributes (for use in shell scripts).
`dmarchive`(1)	Directly copies data between DMF secondary storage and a POSIX filesystem that is not managed by DMF, such as Lustre. It is intended to streamline a work flow in which users work in an unmanaged filesystem and later want to archive a copy of their data via DMF.
`dmcopy`(1)	Copies all or part of the data from a migrated file to an online file.
`dmdu`(1)	Displays the number of blocks contained in specified files and directories on a DMF-managed filesystem.
`dmget`(1)	Recalls the specified files.

`dmfind`(1)	Displays whether files are migrated or not by searching through files in a directory hierarchy.
`dmls`(1)	Displays whether files are migrated or not by listing the contents of a directory.
`dmput`(1)	Migrates the specified files.
`dmtag`(1)	Allows a site-assigned 32-bit integer to be associated with a specific file (which can be tested in the `when` clause of particular configuration parameters and in site-defined policies).
`dmversion`(1)	Displays the version number of the currently installed DMF

The DMF `libdmfusr.so` user library lets you write your own site-defined DMF user commands that use the same application program interface (API) as the above DMF user commands. See Appendix B, "DMF User Library `libdmfusr.so`" on page 385.

Also see Chapter 15, "DMF Client SOAP Service" on page 365.

Licensing Commands

The following commands help you to manage DMF licenses:

Command	Description
`dmusage`(8)	Displays information about the capacity allowed by the DMF licenses and the amount of data that DMF is currently managing against those licenses.
`dmflicense`(8)	Prints DMF license information.

DMF Manager

DMF Manager is an intuitive web-based tool you can use to configure DMF, deal with day-to-day DMF operational issues, and focus on work flow, introduced in "Managing DMF" on page 11. To access DMF Manager, point your browser to the following secure address:

`https://`*YOUR_DMF_SERVER*`:1179`

For details, see Chapter 6, "Using DMF Manager" on page 101.

Configuration Commands

The DMF configuration file (`/etc/dmf/dmf.conf`) contains *configuration objects* and associated *configuration parameters* that control the way DMF operates. By changing the values associated with these objects and parameters, you can modify the behavior of DMF.

To configure DMF, you can use DMF manager or the `dmmaint`(8) command.

For information about configuration, see:

- "Overview of the Installation and Configuration Steps" on page 81
- Chapter 5, "Using `dmmaint` to Install Licenses and Configure DMF" on page 97
- Chapter 6, "Using DMF Manager" on page 101
- Chapter 7, "DMF Configuration File" on page 149
- Chapter 8, "Parallel Data Mover Option Configuration" on page 271

The following man pages are also related to the configuration file:

Man page	Description
`dmf.conf`(5)	Describes the DMF configuration objects and parameters in detail.
`dmconfig`(8)	Prints DMF configuration parameters to standard output.

DMF Daemon and Related Commands

The DMF daemon, `dmfdaemon` (8), communicates with the kernel through a device driver and receives backup and recall requests from users through a socket. The daemon activates the appropriate MSPs and LSs for file migration and recall, maintaining communication with them through unnamed pipes. It also changes the state of inodes as they pass through each phase of the migration and recall process. In addition, the daemon maintains a database containing entries for every migrated file on the system. Updates to database entries are logged in a journal file for recovery. See Chapter 11, "The DMF Daemon" on page 285, for a detailed description of the DMF daemon.

Caution: If used improperly, commands that make changes to the daemon database can cause data to be lost.

The following administrator commands are related to `dmfdaemon` and the daemon database:

Command	Description
`dmaudit`(8)	Reports discrepancies between filesystems and the daemon database. This command is executed automatically if you configure the `run_audit.sh` task.
`dmcheck`(8)	Checks the DMF installation and configuration and reports any problems.
`dmdadm`(8)	Performs daemon database administrative functions, such as viewing individual database records.
`dmdbcheck`(8)	Checks the consistency of a database by validating the location and key values associated with each record and key in the data and key files (also an LS command). If you configure the `run_copy_database.sh` task, this command is executed automatically as part of the task. The consistency check is completed before the DMF databases are saved.
	Note: See "Run Certain Commands Only on a Copy of the DMF Databases" on page 75.
`dmdbrecover`(8)	Applies journal records to a restored backup copy of the daemon database or LS database in order to create an up-to-date sane database.
`dmdidle`(8)	Causes files not yet copied to tape to be flushed to tape, even if this means forcing only a small amount of data to a volume.
`dmdstat`(8)	Indicates to the caller the current status of `dmfdaemon`.

`dmdstop`(8)	Causes `dmfdaemon` to shut down.
`dmfdaemon`(8)	Starts the DMF daemon. The preferred method is to use the `service dmf start` command. [1]
`dmhdelete`(8)	Deletes expired daemon database entries and releases corresponding MSP or VG space, resulting in logically less active data. This command is executed automatically if you configure the `run_hard_delete.sh` task.
`dmmigrate`(8)	Migrates regular files that match specified criteria in the specified filesystems, leaving them as dual-state. This utility is often used to migrate files before running backups of a filesystem, hence minimizing the size of the dump image. It may also be used in a DCM environment to force cache files to be copied to tape if necessary.
`dmsnap`(8)	Copies the daemon database and the LS database to a specified location. If you configure the `run_copy_database.sh` task, this command is executed automatically as part of the task.
`dmversion`(1)	Reports the version of DMF that is currently executing.

1 For instructions about starting and stopping DMF and the mounting service in an HA environment, see the SGI HA documentation.

Space Management Commands

The following commands are associated with automated space management, which allows DMF to maintain a specified level of free space on a filesystem through automatic file migration:

Command	Description
`dmfsfree`(8)	Attempts to bring the free space and migrated space of a filesystem into compliance with configured values.
`dmfsmon`(8)	Monitors the free space levels in filesystems configured with automated space management enabled (`auto`) and lets you maintain a specified level of free space.
`dmscanfs`(8)	Scans DMF filesystems or DCM caches and prints status information to `stdout`.

See Chapter 10, "Automated Space Management" on page 279, for details.

LS Commands

The following commands manage the CAT and VOL records for the LS:

Command	Description
dmcatadm(8)	Provides maintenance and recovery services for the CAT records in the LS database.
dmvoladm(8)	Provides maintenance and recovery services for the VOL records in the LS database, including the selection of volumes for merge operations.

Most data transfers to and from tape media are performed by components internal to the LS. However, the following commands can read LS volumes directly:

Command	Description
dmatread(8)	Copies data directly from LS volumes to disk.
dmatsnf(8)	Audits and verifies the format of LS volumes.

The following commands check for inconsistencies in the LS database:

Command	Description
dmatvfy(8)	Verifies the contents of the LS database against the daemon database. This command is executed automatically if you configure the run_audit.sh task.
dmdbcheck(8)	Checks the consistency of a database by validating the location and key values associated with each record and key in the data and key files.
dmdskvfy(8)	Verifies disk MSP file copies against the daemon database.

Disk MSP Command

The following command supports the disk MSP:

Command	Description
`dmdskvfy`(8)	Verifies disk MSP file copies against the daemon database.

Disk Cache Manager (DCM) Commands

The following commands support the DCM:

Command	Description
`dmdskfree`(8)	Manages file space within the disk cache and as needed migrates files to tape or removes them from the disk cache.
`dmdskvfy`(8)	Verifies disk MSP file copies against the daemon database.

Other Commands

The following commands are also available:

Command	Description
`dmclripc`(8)	Frees system interprocess communication (IPC) resources and token files used by `dmlockmgr` and its clients when abnormal termination prevents orderly exit processing.
`dmcollect`(8)	Collects relevant details for problem analysis when DMF is not functioning properly. You should run this command before submitting a bug report to DMF support, should this ever be necessary.
`dmdate`(8)	Performs calculations on dates for administrative support scripts.
`dmdump`(8)	Creates a text copy of an inactive database file or a text copy of an inactive complete daemon database.

Note: See "Run Certain Commands Only on a Copy of the DMF Databases" on page 75.

`dmdumpj`(8)	Creates a text copy of DMF journal transactions.
`dmfill`(8)	Recalls migrated files to fill a percentage of a filesystem. This command is mainly used in conjunction with dump and restore commands to return a corrupted filesystem to a previously known valid state.
`dmlockmgr`(8)	Invokes the database lock manager. The lock manager is an independent process that communicates with all applications that use the DMF databases, mediates record lock requests, and facilitates the automatic transaction recovery mechanism.
`dmmove`(8)	Moves copies of a migrated file's data to the specified MSPs/VGs.
`dmmvtree`(8)	Moves files from one DMF-managed filesystem to another without requiring file data to be recalled.
`dmov_keyfile`(8)	Creates the file of DMF OpenVault keys, ensuring that the contents of the file are semantically correct and have the correct file permissions. This command removes any DMF keys in the file for the OpenVault server system and adds new keys at the front of the file.
`dmov_loadtapes`(8)	Scans a tape library for volumes not imported into the OpenVault database and allows the user to select a portion of them to be used by a VG. The selected tapes are imported into the OpenVault database, assigned to the DMF application, and added to the LS's database. This command can perform the equivalent actions for the filesystem backup scripts; just use the name of the associated task group instead of the name of a VG.
`dmov_makecarts`(8)	Makes the tapes in one or more LS databases accessible through OpenVault by importing into the OpenVault database any tapes unknown to it and by registering all volumes to the DMF application not yet so assigned. This command can perform the equivalent actions for

	the filesystem backup scripts; just use the name of the associated task group instead of the name of a VG.
`dmselect`(8)	Selects migrated files based on given criteria. The output of this command can be used as input to `dmmove`(8).
`dmsort`(8)	Sorts files of blocked records.
`dmstat`(8)	Displays a variety of status information about DMF, including details about the requests currently being processed by the daemon, statistics about requests that have been processed since the daemon last started, and details of current tape drive usage by VGs.
`dmxfsrestore`(8)	Calls the `xfsrestore`(8) command to restore files dumped to tape volumes that were produced by DMF administrative maintenance scripts.
`tsreport`(8)	Displays information about tape drive errors, alerts, and usage when the `ts` tape driver is used. The `tsreport` command is included in the `apd` RPM.

DMF Licensing

This chapter discusses the following:

DMF License Types

DMF uses software licensing based on License Keys (LK), an SGI Linux product. A production DMF environment requires the following licenses to be installed on the DMF server node: [1]

- DMF *server capability license.*
- One or more cumulative DMF *capacity licenses* (*base* and optional *incremental*), available in different amounts, as shown in Table 2-1.

[1] To support training and functional demonstrations, DMF will run on a server with no license at all up to a maximum stored capacity of 1 TB without TMF or OpenVault.

Table 2-1 Capacity Amounts

Base Capacity	Incremental Capacity
10TB	10TB+
100TB	100TB+
1PB	1PB+
10PB	

At least one base capacity license is required. If multiple base capacity licenses are installed, they are additive.

In order to install an incremental capacity license, the total capacity amount already installed (base plus incremental) must equal or exceed the amount of the new incremental amount. For example, to install a new 100TB+ incremental license, the environment must already be licensed for a total of 100 TB, which could be accomplished by several licensing methods, including any of the following:

– One 100TB base license

– One 10TB base license plus nine 10TB+ incremental licenses

– Two 10TB base licenses plus eight 10TB+ incremental licenses

Note: Some combinations are more cost-effective than others.

- One or more DMF *parallel data mover node licenses*, when using the Parallel Data Mover Option

In high-availability (HA) environments, the passive DMF server will require a DMF HA capability license. In addition, corresponding licenses will be issued for the DMF passive server for each DMF capacity license and each parallel data mover node license that has been purchased for the DMF server.

Figure 2-1 shows the DMF licenses required for a DMF HA environment using two parallel data mover nodes (each of which requires a Parallel Data Mover Option capability license to be installed on the DMF server). The amount of data that DMF is managing requires two capacity licenses. Because this is an HA configuration, the passive DMF server must have the same set of licenses that are installed on the active DMF server.

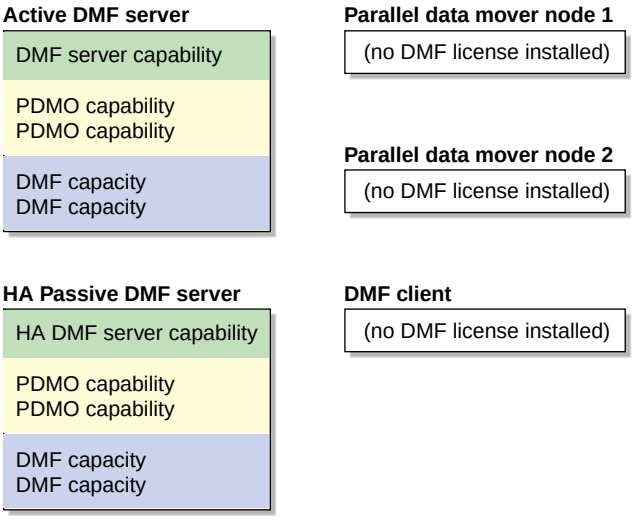

Figure 2-1 DMF Licenses

Determining DMF Capacity

The `dmusage`(8) command displays information about the capacity allowed by the current DMF licenses that are installed on your DMF server and the amount of data that DMF is currently managing, which is defined as the total number of bytes that DMF is managing on all library server (LS) tapes and in all disk cache manager (DCM) or disk media-specific process (MSP) stores.

For example:

```
# dmusage -v
  Store Type        Name                  Bytes
  ----------        ----                  -----
  Disk MSP          dsk1                539934720
  Library Server    ls                  852123456
  Disk MSP          dsk2                539934720
                                         --------
       Total bytes managed             1931992896
      DMF license capacity    21100000000000000 (21100TB)
```

Note: In the DCM and disk MSP calculation, if the `STORE_DIRECTORY` configuration parameter defined for that MSP does not define the root directory of a filesystem, or if other subdirectories of that filesystem are used by other users or processes to store data, the amount of data that DMF is managing that is currently being charged to that MSP may exceed the actual amount of data being managed by that MSP.

The DMF daemon compares the amount of data that DMF is currently managing against the licensed capacity and takes action if the following thresholds are exceeded:

- At 95%, the daemon will send a warning alert once per day.
- At 100%, the daemon will send a critical alert once per day. DMF will continue to function and will recall any data that has already been migrated, but further migrations will not be allowed. The daemon will check once every 2 minutes to see if the usage once again becomes legal (below capacity). This can be achieved by either of the following:
  - The deletion of managed data.
  - The addition of one or more capacity licenses in order to increase the cumulative capacity total to the new desired limit.

    Note: In order to install an incremental capacity license, the total capacity amount already installed (base plus incremental) must equal or exceed the amount of the new incremental amount. See "DMF License Types" on page 47.

  The daemon will issue another alert when the usage once again becomes legal (below capacity).

For details about the right set of licenses for your site, contact SGI Support.

Parallel Data Mover Option and Licensing

Each *active parallel data mover node* requires a corresponding license on the DMF server. DMF will allow as many DMF parallel data mover nodes to become active at one time as there are DMF parallel data mover licenses in the DMF server's license file. (However, a parallel data mover license is not required for the DMF server's integrated data mover functionality.) No license is installed on the parallel data mover node itself.

Mounting Services and Licensing

Use of the TMF or OpenVault mounting service requires DMF licenses.

Gathering the Host Information

When you order DMF, you will receive an entitlement ID. You must submit the system host ID, host name, and entitlement ID when requesting your permanent DMF license keys.

To obtain the host information for a server, execute the following command:

```
/usr/sbin/lk_hostid
```

For example, the following shows that the serial number is `000423d5fd92` and the license ID is `23d5fd92`:

```
# /usr/sbin/lk_hostid
000423d5fd92 23d5fd92 socket=1 core=2 processor=2
#-------------------------------------------------------------------------
--
#The above is the default selected by lk_hostid. See below for additional
#hostid pairs.
#-------------------------------------------------------------------------
--
#Interface  SN                   LI         Driver ( Comment )
#-------------------------------------------------------------------------
--
 eth0       000423d5fd92         23d5fd92   e1000
 eth1       000423d5fd93         23d5fd93   e1000
```

Obtaining the License Keys

To obtain your DMF license keys, see information provided in your customer letter and the following web page:

http://www.sgi.com/support/licensing

Installing the License Keys

To install the license keys, copy them into the `/etc/lk/keys.dat` file.

Verifying the License Keys

You can verify your licenses with the following commands:

- "`dmflicense`" on page 53
- "`lk_verify`" on page 53

dmflicense

You can use the dmflicense(8) to verify the license keys. To see more output, use the -v option. For example:

```
# dmflicense -v
        File /etc/lk/keys.dat, line 6 is a valid DMF_SERVER license
        File /etc/lk/keys.dat, line 24 is a valid DMF_PDMO license
        File /etc/lk/keys.dat, line 29 is a valid DMF_PDMO license
        File /etc/lk/keys.dat, line 12 is a valid DMF_CAPACITY TB=100 license
        File /etc/lk/keys.dat, line 18 is a valid DMF_CAPACITY TB=100+ license
        Valid DMF license found.
        DMF capacity is 200TB.
```

lk_verify

You can use the lk_verify(1) command with the -A option to verify LK licenses. To see more output, use the -v option. For example:

```
# lk_verify -A -vvv
lk_check        All     All : total found=5

  1 /etc/lk/keys.dat:005        product=DMF_SERVER, version=5.000, count=0, begDate=1256145804, \
        expDate=1263967199, licenseID=201e8636, key=OOphH5GgIu25rNQN1GtQAS8OcA4uQ1kB, \
        info='DMF 5.X Server', vendor='Silicon Graphics International', \
        ref_id='207552'
               Verdict:         SUCCESS. Nodelock. Uncounted.
                                Available since 12 days on 21-Oct-2009 12:23:24.
                                No End Date.

               Attribute 1 of 3 : info=DMF 5.X Server
               Attribute 2 of 3 : vendor=Silicon Graphics International
               Attribute 3 of 3 : ref_id=207552


  2 /etc/lk/keys.dat:011        product=DMF_PDMO, version=5.000, count=0, begDate=1256145990, \
        expDate=1263967199, licenseID=201e8636,  key=7hspbTt4yFQ8EWZhzvQiNX8HzbTCw5Yp, \
        info='DMF 5.X PDMO 1 NODE',attr='NODE 1', \
        vendor='Silicon Graphics International',ref_id='207554'
               Verdict:         SUCCESS. Nodelock. Uncounted.
                                Available since 12 days on 21-Oct-2009 12:26:30.
```

```
                              No End Date.

               Attribute 1 of 4 : info=DMF 5.X PDMO 1 NODE
               Attribute 2 of 4 : attr=NODE 1
               Attribute 3 of 4 : vendor=Silicon Graphics International
               Attribute 4 of 4 : ref_id=207554

3 /etc/lk/keys.dat:017        product=DMF_PDMO, version=5.000, count=0, begDate=1256145990, \
       expDate=1263967199, licenseID=201e8636, key=bG962h6V2yKZ9Wii3FD5FvyOLC9EL+1F, \
       info='DMF 5.X PDMO 1 NODE',attr='NODE 1', \
       vendor='Silicon Graphics International',ref_id='207555'
               Verdict:         SUCCESS. Nodelock. Uncounted.
                                Available since 12 days on 21-Oct-2009 12:26:30.
                                No End Date.

               Attribute 1 of 4 : info=DMF 5.X PDMO 1 NODE
               Attribute 2 of 4 : attr=NODE 1
               Attribute 3 of 4 : vendor=Silicon Graphics International
               Attribute 4 of 4 : ref_id=207555


4 /etc/lk/keys.dat:023        product=DMF_CAPACITY, version=5.000, count=0, begDate=1256146084, \
       expDate=1263967199, licenseID=201e8636, key=QDDj528gHnysskk8jTKgIOSj78j01seU, \
       info='DMF 5.X 10tb base',attr='TB=10', \
       vendor='Silicon Graphics International',ref_id='207559'
               Verdict:         SUCCESS. Nodelock. Uncounted.
                                Available since 12 days on 21-Oct-2009 12:28:04.
                                No End Date.

               Attribute 1 of 4 : info=DMF 5.X 10tb base
               Attribute 1 of 4 : info=DMF 5.X 10tb base
               Attribute 2 of 4 : attr=TB=10
               Attribute 3 of 4 : vendor=Silicon Graphics International
               Attribute 4 of 4 : ref_id=207559


5 /etc/lk/keys.dat:029        product=DMF_CAPACITY, version=5.000, count=0, begDate=1256146158, \
       expDate=1263967199, licenseID=201e8636, key=5Muz5pXr3xo7bEG/9OS8pD6AgIRBrHhB, \
       info='DMF 5.X 10tb incrmt',attr='TB=10+', \
       vendor='Silicon Graphics International',ref_id='207560'
               Verdict:         SUCCESS. Nodelock. Uncounted.
```

```
                              Available since 12 days on 21-Oct-2009 12:29:18.
                              No End Date.

                  Attribute 1 of 4 : info=DMF 5.X 10tb incrmt
                  Attribute 2 of 4 : attr=TB=10+
                  Attribute 3 of 4 : vendor=Silicon Graphics International
                  Attribute 4 of 4 : ref_id=207560

lk_check          All     All : total matched=5
```

For More Information About Licensing

To request software keys or information about software licensing, see the following web page:

http://www.sgi.com/support/licensing

If you do not have access to the web, please contact your local Customer Support Center.

DMF Best Practices

This chapter discusses the following:

Installation, Upgrade, and Downgrade Best Practices

This section discusses the following:

Use the Correct Mix of Software Releases

In a production system, all nodes that are functioning as DMF servers or DMF data movers (DMF server nodes and any DMF parallel data mover nodes) should run the same set of releases as supported by and provided in a given ISSP release; the same level of OS, SGI Foundation Software, DMF, and (in a system with parallel data mover nodes) CXFS.

To support upgrading without having to take down the whole environment, nodes can temporarily run different releases during the upgrade process, as provided by the CXFS rolling upgrade procedure.

Caution: You must upgrade all CXFS server-capable administration nodes before upgrading any CXFS client-only nodes (server-capable administration nodes must run the same or later release as client-only nodes.) Operating a cluster with clients running a mixture of older and newer CXFS versions may result in a performance loss. Relocation to a server-capable administration node that is running an older CXFS version is not supported.

Although CXFS client-only nodes and DMF parallel data mover nodes that are not upgraded might continue to function without problems, new functionality may not be enabled until all nodes are upgraded; SGI does not provide support for any problems encountered on the nodes that are not upgraded.

For details, see the section about CXFS release versions and rolling upgrades in the *CXFS 6 Administration Guide for SGI InfiniteStorage*.

Do Not Use YaST to Configure Network Services

If you try to configure network services using YaST and you are using DHCP, YaST will modify the `/etc/hosts` file to include the following entry, where `hostname` is the name of your machine:

`127.0.0.2` *hostname hostname*

The above line will prevent `ov_admin` from working because there cannot be multiple IP addresses defined for the DMF server hostname. You will see an error such as the following:

```
The OpenVault server name "hostname" matches this host's hostname,
but network packets for this hosts's IP address:
        127.0.0.2
are not being accepted by any installed ethernet card, so there appears
to be a problem with the configuration of /etc/hosts.  Please correct
this problem before continuing.
```

If you are using OpenVault, you should do one of the following:

- Remove the `127.0.0.2` line from the `/etc/hosts` file prior to configuring OpenVault
- Do not use YaST to configure network services

Take Appropriate Steps when Upgrading DMF

Note: If you are upgrading from DMF 3.9 or earlier, see the information about upgrade caveats in the *SGI InfiniteStorage Software Platform* (ISSP) release note for more information.

To perform an upgrade, do the following:

1. Read the ISSP release note, DMF release note, and any late-breaking caveats on Supportfolio. Pay particular attention to any installation and upgrade caveats.
2. Stop all applications that are writing data to the DMF-managed filesystems.
3. Save the established DMF and mounting service configurations to an external storage medium.
4. Stop DMF (non-HA environment):

Caution: For instructions about starting and stopping DMF and the mounting service in an HA environment, see the SGI HA documentation.

```
# service dmf stop
```

5. Stop the applicable mounting service:
   - TMF:

     ```
     # service tmf stop
     ```

   - OpenVault:

     ```
     # service openvault stop
     ```

6. If the DMF administration filesystems are XFS filesystems, make a copy of the `fstab`(5) file. For example:

   ```
   # cp /etc/fstab /myupgrade/fstab
   ```

7. Make a copy of the following:
   a. The DMF configuration file `dmf.conf`. For example:

      ```
      # cp /etc/dmf/dmf.conf /myupgrade/dmf.conf
      ```

b. The mounting service configuration information:

- TMF: copy the `tmf.config` file to a safe location. For example:

```
# cp /etc/tmf/tmf.config /myupgrade/tmf.config
```

- OpenVault (if the OpenVault configuration is set up on the boot partition and not under a DMF administration filesystem): create a compressed file of the OpenVault configuration directory `/var/opt/openvault`. For example:

```
# cd /var/opt
# /bin/tar cf /myupgrade/somefile.tar openvault/*
# /usr/bin/compress /myupgrade/somefile.tar
```

c. Networking files for `exports`(5), `auto.master`(5), and `resolve.conf`(5). For example:

```
# cp /etc/exports /myupgrade/exports
# cp /etc/auto.master /myupgrade/auto.master
# cp /etc/resolv.conf /myupgrade/resolv.conf
```

8. Upgrade the operating system software and SGI Foundation Software to the level supported by the version of DMF that you are upgrading to, paying particular attention to any installation and upgrade caveats in the release notes and any late-breaking caveats on Supportfolio.

9. If your DMF administration filesystems are of XFS type, do the following:

Note: To avoid copying the `fstab` information from a previous partition, **do not copy** the saved `/myupgrade/fstab` file to the new `/etc` directory in the upgraded system.

---

a. Use the `cat` command to view the previous `fstab` file:

```
# cat /myupgrade/fstab
```

The following is an example of how DMF administration filesystems could be set up within a `/etc/fstab` file:

```
/dev/lxvm/home          /dmf/home       xfs     defaults                0 0
/dev/lxvm/journals      /dmf/journals   xfs     defaults                0 0
/dev/lxvm/move          /dmf/move       xfs     dmi,mtpt=/dmf/move      0 0
/dev/lxvm/spool         /dmf/spool      xfs     defaults                0 0
```

```
/dev/lxvm/cache         /dmf/store      xfs     dmi,mtpt=/dmf/store     0 0
/dev/lxvm/tmp           /dmf/tmp        xfs     defaults                0 0
/dev/lxvm/dmfusr1       /dmfusr1        xfs     dmi,mtpt=/dmfusr1       0 0
/dev/lxvm/dmfusr3       /dmfusr3        xfs     dmi,mtpt=/dmfusr3       0 0
```

b. Verify the existence of the matching XFS devices on the upgraded system by using the `ls` command:

```
# ls -al /dev/lxvm*
```

c. Copy and paste the DMF administration filesystem entry lines (those that contain `/dmf/`*filesystemname*) from the copy of the `fstab` (`/myupgrade/fstab`) into the new `/etc/fstab` file for the upgraded system.

10. Reestablish the files and directories copied in step 7 above to their normal locations on the upgrade system. For example:

```
# cp /myupgrade/dmf.conf /etc/dmf/dmf.conf
# cp /myupgrade/exports /etc/exports
# cp /myupgrade/auto.master /etc/auto.master
# cp /myupgrade/resolv.conf /etc/resolv.conf
```

If TMF, also:

```
# cp /myupgrade/tmf.config /etc/tmf/tmf.config
```

If OpenVault (and if the OpenVault configuration is set up on the boot partition and not under a DMF administration filesystem), also do the following, for example:

```
# cd /var/opt
# /bin/tar xf /myupgrade/somefile.tar.Z
```

11. Follow upgrade instructions in the ISSP release note to update the DMF and mounting service software.

Contact SGI Support to Downgrade After Using OpenVault™ 4.0 or Later

If you are running OpenVault and want to downgrade after using OpenVault 4.0, you must contact SGI support for assistance.

Configuration Best Practices

This section discusses the following:

Use Sufficiently Fast Filesystems

A filesystem on which DMF operates must fast-enough to permit efficient streaming to/from all secondary storage media. This is particularly important for tape drives, because slow I/O can lead to increased wear on the drive and cartridges (due to excessive stopping and starting of the drive heads).

Make Changes Safely to the DMF Configuration

It is safest to make changes to the DMF configuration while DMF is stopped.[1] Make a backup copy of the DMF configuration file before making changes.

You should make and mount the DMF administrative filesystems before using DMF Manager or `dmmaint`. If you try to apply configuration changes without these filesystems in place, you will get errors. See "Configure DMF Administrative Filesystems and Directories Appropriately" on page 64.

1 For instructions about starting and stopping DMF and the mounting service in an HA environment, see the SGI HA documentation.

SGI recommends that you use one of the following tools and verify any configuration changes you make:

- When using DMF Manager, select:

  Overview
  > Configuration ...
  > Validate Current Configuration

- When using the `dmmaint` tool, use the `INSPECT` button before committing the changes. See Chapter 5, "Using `dmmaint` to Install Licenses and Configure DMF" on page 97.

If using a file editing tool such as `vi` to directly edit the DMF configuration file, you should run `dmcheck` after making changes.

If you make changes while DMF is running, be cautious. Never change the following parameters while DMF is running:

```
DRIVE_GROUPS
EXPORT_METRICS
LS_NAMES
MSP_NAMES
SERVICES_PORT
VOLUME_GROUPS
```

For more information, see Chapter 7, "DMF Configuration File" on page 149.

Configure DMF Administrative Filesystems and Directories Appropriately

The DMF server uses a set of filesystems and directories in which it stores databases, log files, journal files, and temporary file directories. These are known as the *DMF administrative filesystems and directories*. You specify the location of these filesystems and directories by using the following parameters in the DMF configuration file:

- `HOME_DIR` specifies the base pathname (such as `/dmf/home`) for directories in which the DMF daemon database, LS database, and related files reside.
- `JOURNAL_DIR` specifies the base pathname (such as `/dmf/journals`) for directories in which the journal files for the daemon database and LS database will be written.

- SPOOL_DIR specifies the base pathname (such as /dmf/spool) for directories in which DMF log files are kept.
- TMP_DIR specifies the base pathname (such as /dmf/tmp) for directories in which DMF puts temporary files for its own internal use.
- MOVE_FS specifies the scratch filesystem (such as /move_fs) that is used by dmmove(8) to move files between media-specific processes (MSPs) or volume groups (VGs).
- CACHE_DIR specifies the directory (such as /dmf/cache) in which the VG stores chunks while merging them from sparse tapes.
- STORE_DIRECTORY specifies the directory (such as /remote/dir) that is used to hold files for a DCM or disk MSP (there is one STORE_DIRECTORY parameter for each DCM or disk MSP). You must mount *STORE_DIRECTORY* with the dirsync option in order to ensure the integrity and consistency of *STORE_DIRECTORY* with the DMF daemon database in the event of a system crash.

Note: To minimize the possibility that the filesystem will become full and cause DMF to abort, *HOME_DIR*, *JOURNAL_DIR*, and *SPOOL_DIR* should each be the mount point of a filesystem that is used only by DMF. To provide the best chance for database recovery, *HOME_DIR* must be on a different physical device from *JOURNAL_DIR*. When using the Parallel Data Mover Option, *HOME_DIR*, *SPOOL_DIR*, *TMP_DIR*, *MOVE_FS*, *CACHE_DIR* (if used), and *STORE_DIRECTORY* for a DCM must be CXFS filesystems or be in CXFS filesystems.

In general, these filesystems should be sized in terms of gigabytes. Table 3-1 shows the minimum recommended sizes.

Table 3-1 Minimum Sizes for DMF Administrative Filesystems / Directories

Filesystem / Directory	Minimum Recommended Size (GB)
`HOME_DIR`	300
`JOURNAL_DIR`	300
`SPOOL_DIR`	1000
`TMP_DIR`	300
`MOVE_FS`	The capacity of one new tape cartridge

For individual guidelines and requirements for each filesystem or directory, see the specific parameter descriptions in Chapter 7, "DMF Configuration File" on page 149 and the following sections:

- "*HOME_DIR* Size" on page 66
- "*MOVE_FS* Performance and Size" on page 67
- "`mkfs` and `mount` Parameters" on page 68

See also "Make Changes Safely to the DMF Configuration" on page 63.

HOME_DIR Size

You can better estimate the size required for the *HOME_DIR* filesystem that holds the daemon database and the LS database by using the following guidelines:

- The daemon data records file (`dbrec.dat`) holds 27 records per 4096–byte block, and there is one record for each copy of a migrated file.
- An additional file that indexes the daemon records (`dbrec.keys`) requires 50% as much space the daemon data records file.
- If you have an LS:
  - The CAT data records file (`tpcrdm.dat`) holds 35 records per 4096–byte block, and there is one CAT record for each migrated copy of a file. (If a migrated copy of a file is divided between two physical media, there will be a CAT record for each chunk.)

– Two additional files that index the CAT records (`tpcrdm.key1.keys` and `tpcrdm.key2.keys`) require 75% and 50% as much space, respectively, as the CAT data records file.

For example, suppose you have 10 million DMF-managed files, each with 2 migrated copies controlled by an LS. This situation would require approximately the following amounts of space:

- Daemon database:

  `dbrec.dat`: 10m * 2 / 27 * 4096 = 3.0 GB
  `dbrec.keys`: 3.0 GB * 0.50 = 1.5 GB
  Total: 3.0 + 1.5 = 4.5 GB

- CAT records in the LS database:

  `tpcrdm.dat`: 10m * 2 / 35 * 4096 = 2.3 GB
  `tpcrdm.key1.keys`: 2.3 GB * 0.75 = 1.7 GB
  `tpcrdm.key2.keys`: 2.3 GB * 0.50 = 1.2 GB
  Total : 2.3 + 1.7+ 1.2 = 5.2 GB

Therefore, the *HOME_DIR* directory in this case must have at least 9.7 GB (4.5 GB + 5.2 GB) available for the databases.

Note: Other database information, such as for the VOL records in the LS database, requires an insignificant amount of space in comparison to the daemon database and CAT records in the LS database.

MOVE_FS Performance and Size

The *MOVE_FS* filesystem should have performance characteristics similar to the primary DMF-managed filesystems because DMF will follow the same rules for drive utilization as defined in the drive groups and volume groups (`DRIVE_MAXIMUM` and `MAX_PUT_CHILDREN`) when moving large numbers of files. A *MOVE_FS* filesystem with slower bandwidth than what `DRIVE_MAXIMUM` and `MAX_PUT_CHILDREN` are tuned for may become overloaded with DMF requests. In extreme cases, DMF can become backlogged on the *MOVE_FS* filesystem and delay the processing of user requests.

The size of the *MOVE_FS* filesystem should be approximately the capacity of a data cartridge, including compression, times the `MAX_PUT_CHILDREN` value.

For example:

*500 GB native capacity * 1.6 compression * 3 drives = 2.4 TB*

`mkfs` and `mount` Parameters

Tuning the XFS log is important for performance, especially on the *MOVE_FS* filesystem, which will have heavy metadata activity from the quantity of small files that pass through it.

SGI recommends the following options to tune the XFS log:

- `mkfs.xfs` options:

  ```
  -i attr=2 -l version=2,sunit=512,size=128m
  ```

- `mount` options:

  ```
  logbufs=8,logbsize=256k
  ```

By default, the XFS create inodes such that inode numbers will not occupy more than 32 bits of significance. You should not use the `inode64` option for the *MOVE_FS* filesystem because DMF places all files being moved in a single directory (such as*MOVE_FS*`/.dmfprivate/unmigdir`), and `inode64` will try to create all files in the same allocation group, which may not be ideal. The default (equivalent to `inode32`) will place each file in the next allocation group and spread the work around the filesystem.

The defaults for the allocation group size and number are usually sufficient.

Also see the following:

- "Filesystem Mount Options" on page 85
- "`DMAPI_PROBE` Must Be Enabled for SLES 10 or SLES 11 Nodes When Using CXFS" on page 90
- The `mount`(8) and `mkfs.xfs`(8) man pages

Use Inode-Resident Extended Attributes and 256–byte Inodes

SGI recommends that you configure your filesystems so that the extended attribute used by DMF is always inode-resident and that you use 256–byte inodes and the

default `attr2` (`-i attr=2` option to `mkfs.xfs`) when possible. See "Inode Size Configuration" on page 86.

Limit Path Segment Extension Records

You should configure your database record length so that the number of records that require a path segment extension record is kept to as small a number as possible. See "Daemon Database Record Length" on page 87.

Do Not Change Script Names

Do not change the pathnames or script names of the DMF administrative tasks. For more information, see "Automated Maintenance Tasks" on page 89.

Configure DMF Appropriately with CXFS™

DMF must make all of its DMAPI interface calls through the CXFS active metadata server. The CXFS client nodes do not provide a DMAPI interface to CXFS mounted filesystems. A CXFS client routes all of its communication to DMF through the metadata server. This generally requires that DMF run on the CXFS metadata server. If DMF is managing a CXFS filesystem, DMF will ensure that the filesystem's CXFS metadata server is the DMF server and will use metadata server relocation if necessary to achieve that configuration.

Note: DMF data mover processes must run only on the DMF server node and any parallel data mover nodes. Do not run data mover processes on CXFS standby metadata server nodes.

To use DMF with CXFS, do the following:

- For server-capable administration nodes, install the `sgi-dmapi` and `sgi-xfsprogs` packages from the SGI InfiniteStorage Software Platform (ISSP) release. These are part of the **DMF Server** and **DMF Parallel Data Mover** YaST patterns. The DMF software will automatically enable DMAPI, which is required to use the `dmi` mount option.

  For CXFS client-only nodes, no additional software is required.

- When using the Parallel Data Mover Option, install the **DMF Parallel Data Mover** software package, which includes the required underlying CXFS client-only software. (From the CXFS cluster point of view, the DMF parallel data mover node is a CXFS client-only node but one that is dedicated to DMF data mover activities.) For more information, see:
  - "Parallel Data Mover Option" on page 5
  - "Parallel Data Mover Option Configuration Procedure" on page 271
- Use the `dmi` option when mounting a filesystem to be managed.
- Start DMF on the CXFS active metadata server for each filesystem to be managed.

See also "SAN Switch Zoning or Separate SAN Fabric Requirement" on page 17.

For more information about CXFS, see:

- *CXFS 6 Administration Guide for SGI InfiniteStorage*
- *CXFS 6 Client-Only Guide for SGI InfiniteStorage*

Improve Tape Drive Performance with an Appropriate Zone Size

When using a library server (LS), it is critical that the zone size you specify for the VG (the `ZONE_SIZE` parameter) is appropriate for the tape speed and average data compression rate at your site. A value that is too small can cause poor write performance because a tape mark is written at the end of each zone; a value that is too large can reduce parallelism when migrating files.

The optimal zone size depends upon several site-specific factors. Answering the following questions will help you determine the correct zone size for your site:

- How long does it take the tape drive to flush data to media?

  Note: Different drive types have different bandwidths, and the same drive type can have different bandwidths with different cartridge types.

- How fast can the tape drive write data?
- What is the average data compression rate? If your data compresses well, the zone size should be larger; if the data does not compress well, the zone size should be smaller.

A good zone size is one where the time spent flushing data to media is not a significant amount of the total I/O time. For increased write performance, choose a zone size such that the average time to write a tape mark for the drive type is a small percentage (for example, 5%) of the time to write a zone at the drive's native rate.

For example, suppose the following:

- The tape drive requires 2 seconds to flush the data to tape
- The tape drive writes data at 120 MB/s
- The average compression rate is 2 to 1

In order to waste no more than 5% of the full bandwidth of the drive flushing data to media, the ZONE_SIZE value in this case must be large enough to hold 40 seconds (2 seconds / 0.05) worth of data in each zone. Because the tape drive writes at about 120 MB/s, then 40*120=4800 MB of data that can be written in 40 seconds. Not considering compression, a good preliminary ZONE_SIZE value is therefore 5g (5 GB).

Because the example site has a compression rate of 2 to 1, the preliminary ZONE_SIZE value should be multiplied by 2; the resulting ZONE_SIZE value should be 10g (10 GB), which is how much data will get written in 40 seconds while still keeping the flush waste within 5% of the total bandwidth.

Note: The zone size influences the required cache space. The value for the CACHE_SPACE parameter should be at least twice the value used for ZONE_SIZE. Increasing the ZONE_SIZE value without also increasing CACHE_SPACE could cause tape merging to become inefficient. Tape merges could have problems if the ZONE_SIZE value is larger than the CACHE_SPACE value. For more information about CACHE_SPACE, see "libraryserver Object" on page 215.

For more information about zone size, see the following:

- ZONE_SIZE parameter in "volumegroup Object" on page 224
- "Media Concepts" on page 303
- Appendix G, "Case Study: Impact of Zone Size on Tape Performance" on page 473

Back Up the DMF Configuration

After you have initially successfully configured DMF, make a backup copy of the DMF configuration file (`/etc/dmf/dmf.conf`) so that you can return to it in case of failure.

If you are using DMF Manager, it will automatically make a time-stamped backup for you.

Add HBA Drivers to the `initrd` Image

The `ts` tape driver reads HBA information from `sysfs` just after being loaded in order to discover controller information. To ensure that this information is available when `ts` loads, SGI recommends that you add the HBA drivers to the `initrd` image so that they load early in the boot process. Do the following:

1. Add the HBA driver to the `INITRD_MODULES` line in the `/etc/sysconfig/kernel` file. For example, to add the driver QLogic QLA2200, you would include `qla2xxx` in the `INITRD_MODULES` line.
2. Create the initial RAM disk image so that it contains your modification:

   ```
   # mkinitrd
   ```

3. Reboot the DMF server.

Use Default Setting for `RECALL_NOTIFICATION_RATE`

You should use the default setting for `RECALL_NOTIFICATION_RATE` unless you are aware of a value that is more appropriate for your site. Setting `RECALL_NOTIFICATION_RATE` to `0` is no longer a standard best practice.

Set the `xinetd tcpmux instances` Parameter Appropriately

You must use a sufficient setting for the `tcpmux instances` parameter in either the `/etc/xinetd.conf` file or the `/etc/xinetd.d/tcpmux` file.

Each remote DMF client command will consume one instance of a `tcpmux` service while it is active. For that reason, SGI recommends that you add the `instances` parameter to `/etc/xinetd.d/tcpmux` rather than increasing the `instances` parameter in `/etc/xinetd.conf`.

Determining the correct setting of this parameter depends on what the maximum number of simultaneous remote DMF user commands might be combined with any other `xinetd` `tcpmux` services that will be used. See the `xinetd`(8) man page for more information on setting the parameter.

Additionally, it is important that the `tcpmux` service is not disabled. If the following configuration line exists in `/etc/xinetd.d/tcpmux`, remove it:

```
disable = yes
```

Avoid Unintentional File Recall by Filesystem Browsers

Graphical user interface (GUI) filesystem browsers (such as Windows Explorer, GNOME™ Nautilus / File Manager) can unintentionally cause files to be recalled because they read the first few blocks of the file in order to show the correct icon in the view screen:

- Windows Explorer: if you follow the directions in "Modify Settings If Providing File Access via Samba" on page 79, you can avoid this problem for Windows Explorer.
- Nautilus and other filesystem browsers: these filesystem browsers may have settings to prevent them from reading the file for thumbnail icons, but testing is still required because the browser may still read the file for other reasons. Also, file browser behavior may change in future releases, so you must retest after upgrading. You should do one of the following for these filesystem browsers:
  - Do not use GUI filesystem browsers on a DMF-managed filesystem.
  - Set the DMF policy to keep the number of kilobytes permanently on disk required by your filesystem browser, to allow the reading activity to happen without recalling files. Do the following:
    1. Determine how many kilobytes are read by your filesystem browser.
    2. Verify that the partial-files feature is enabled (see `PARTIAL_STATE_FILES` in "`dmdaemon` Object" on page 160).
    3. Use the `ranges` clause to keep the required number of bytes of each file online. See:
       - "`ranges` clause" on page 206

- "Automated Space-Management Procedure" on page 209, particularly Procedure 7-1 step 4d and Example 7-15, page 212

4. Repeat the above steps as needed after upgrading the filesystem browser.

See also:

- "File Regions and Partial-State Files" on page 26
- Appendix F, "Considerations for Partial-State Files" on page 471

Administrative Best Practices

This section discusses the following:

Monitor DMF Daily

You should monitor DMF on a daily basis to ensure that it is operating properly and that you find any problems in time to retrieve data.

DMF provides a number of automated tasks that you can configure to generate reports about errors, activity, and status. Additionally, some serious error conditions generate email messages. Examining this information on a timely basis is important to ensure that DMF is operating properly and to diagnose potential problems.

Migrate Multiple Copies of a File

When you migrate a file, make at least two copies of it on separate media to prevent file data loss in the event that a migrated copy is lost.

Back Up Migrated Filesystems and DMF Databases

DMF moves only the data associated with files, not the file inodes or directories, so you must still run filesystem backups in order to preserve the metadata associated with migrated files and their directories. You can configure DMF to automatically run backups of your DMF-managed filesystems.

You must also back up the daemon database and the LS database regularly using the `run_copy_databases.sh` task.

See:

- "DMF Administration" on page 33
- "`taskgroup` Object" on page 170
- "Backups and DMF" on page 349

Run Certain Commands Only on a Copy of the DMF Databases

You should run the following commands **only on a copy** of the DMF databases:

- `dmdbcheck`(8)
- `dmdump`(8)

If you run these commands on an active database, (that is, on a database located in the *HOME_DIR* directory while DMF is running), the results of the commands will be unreliable because DMF may be actively changing the data while the command is running.

Be Aware of Differences in an HA Environment

If you run DMF in a high-availability (HA) cluster, there are a some configuration requirements and administrative procedures that differ from the information in this guide. For example, in an HA environment you must first remove Heartbeat control of the resource group before stopping DMF.

Avoid Bottlenecks when Tape Drives and Host Port Speeds Do Not Match

Note: This section does not apply to STK drives. For those drives, the only control is the size of the tape drive I/O request, which DMF determines. STK 4–Gbit adapters perform at approximately 200 MB/s.

If you have one 4-Gbit host port and are writing data to multiple 2-Gbit tape drives, the aggregate desired bandwidth on the host port is greater than the data rate of the Fibre Channel adapters on the tape drives. This can cause the switch's frame buffers to fill up, causing the switch to stop accepting data from the 4-Gbit HBA, dropping the effective data rate close to that of a 2-Gbit HBA.

You can correct this situation by changing the maximum burst size (`burst_size`) for the tape drive. The maximum burst size specifies the maximum amount of data that the port can transfer during a single operation. It should be double the switch port buffering (after unit conversions, because maximum burst size is in units of 512 bytes). For example, a Brocade 4100 switch has at least 32 KB of buffering per port, so you would start with a value of `128`.

Note: Determining the optimum value for `burst_size` depends upon many site-specific factors, including HBA speed, switch speed, tape speed, and number of tapes per port; it may take some trial-and-error to set optimally. SGI suggests beginning by using a value of `64` or `128`, which have been shown to improve results without negative impact.

Before changing the maximum burst size, ensure that you have stopped DMF, APD, and the TMF or OpenVault mounting service.[2]

If you have installed the optional `sdparm` RPM from SLES, you can use the `sdparm` command to set the burst size:

```
# sdparm -t fcp --set MBS=burstsize /dev/sgNN
```

You can test the effects of changing the burst size by doing the following:

1. Stop DMF, APD, and the TMF or OpenVault mounting service.[3]
2. Ensure you have two 2–Gbit tape drives on 4–Gbit FC switch with one 4–Gbit host connection.
3. Set the maximum burst size to `0` (no limit) on both drives. For example:

   ```
   # sdparm -t fcp --set MBS=0 /dev/sg0
   ```

4. Load scratch tapes on the drives.
5. Enter the following for each drive separately and then both drives in parallel and monitor performance with PCP or an FC switch tool:

   ```
   # dd  if=/dev/zero of=/dev/ts/... bs=256k
   ```

6. Change maximum burst size. For example, to set it to `128`:

   ```
   # sdparm -t fcp --set MBS=128 /dev/sg0
   ```

7. Enter the following for each drive separately and then both drives in parallel and monitor performance with PCP or an FC switch tool:

   ```
   # dd  if=/dev/zero of=/dev/ts/... bs=256k
   ```

2 For instructions about starting and stopping DMF and the mounting service in an HA environment, see the SGI HA documentation.

3 For instructions about starting and stopping DMF and the mounting service in an HA environment, see the SGI HA documentation.

To determine the current maximum burst size, use the `sginfo -D` command. For example:

```
# sginfo -D /dev/sg0
Disconnect-Reconnect mode page (0x2)
------------------------------------
Buffer full ratio                   0
Buffer empty ratio                  0
Bus Inactivity Limit (SAS: 100us)  0
Disconnect Time Limit               0
Connect Time Limit (SAS: 100us)    0
Maximum Burst Size                  128
EMDP                                0
Fair Arbitration (fcp:faa,fab,fac) 0
DIMM                                0
DTDC                                0
First Burst Size                    0
```

You can also use the `sdparm --get` command if you have installed the optional `sdparm` RPM from SLES. For example:

```
# sdparm -t fcp --get MBS /dev/sg0
```

For more information about `sdparm`, see:

http://freshmeat.net/projects/sdparm/

http://dag.wieers.com/rpm/packages/sdparm/

Use `N-port` Topology for All LSI Fibre Channel Ports Used with Tape Drives

During error recovery, a bus reset will cause the LSI Fibre Channel port to renegotiate its connection with the Fibre Channel switch. This renegotiation can result in the LSI host port acquiring a different port ID. Should this happen, reservation conflicts or errors that result in the tape driving transitioning to `swdn` can occur. To avoid this problem, use `lsiutil` to set the link topology to `N-port` for all LSI Fibre Channel ports used with tape drives, which eliminates the possibility that the host adapter port could acquire a different port ID.

Start Site-Specific Configuration Parameters and Stanzas with "LOCAL_"

If you choose to add site-specific parameters or object stanzas to the DMF configuration file, you should begin the parameter name or stanza name with "LOCAL_" (such as LOCAL_MYPARAM) so that the names will not cause conflict with future SGI DMF parameters and stanzas.

Use TMF Tracing

Each TMF process writes debugging information to its own trace file, located in the directory specified by the `trace_directory` parameter in the TMF configuration file `/etc/tmf/tmf.config`. If you use TMF, you should leave TMF tracing on so that this debugging information is available if problems occur.

The trace files are circular, meaning they only contain the most recent activity from a TMF process. To change the amount of history available in a trace file, modify the `trace_file_size` configuration parameter.

When TMF is restarted, any trace files from the previous instance of TMF are moved to the directory specified in `trace_save_directory`.

For more information, see *TMF 5 Administrator's Guide for SGI InfiniteStorage.*

Run `dmcollect` If You Suspect a Problem

As soon as you suspect a problem with DMF, run the `dmcollect` (8)command to gather the relevant information about your DMF environment that will help you and SGI analyze the problem.

Note: Take care to enter the correct number of previous days from which to gather information, so that logs containing the first signs of trouble are included in the collection.

Also see Chapter 16, "Troubleshooting" on page 369.

Modify Settings If Providing File Access via Samba

You can avoid an unnecessary Windows SMB request timeout by setting the `SessTimeout` parameter to a value appropriate for a DMF environment, such as 300

seconds. This is especially important for slower mounting/positioning libraries and tape drives. For details, see the following website:

http://technet.microsoft.com/en-au/library/cc938292.aspx

The Windows Explorer desktop can show which files in an SMB/CIFS network share are in a fully or partially offline state. If so enabled, Windows Explorer overlays a small black clock on top of a migrated file's normal icon; the black clock symbol indicates that there may be a delay in accessing the contents of the file. (This feature is disabled by default.)

To enable this feature, do the following:

1. Install the `sgi-samba` RPMs from ISSP.

   Note: This feature is not available in community Samba.

2. Add the following line to the Samba configuration file `/etc/samba/smb.conf` on the DMF server:

   ```
   dmapi support = Yes
   ```

3. Restart the `smb` daemon on the DMF server:

   ```
   server# /etc/init.d/smb restart
   ```

For more information, see the `smb.conf`(5) man page.

Disable Journaling When Loading an Empty Database

If you are loading an empty database, you should disable journaling in order to eliminate unnecessary overhead. To do this, use the `-j` option to the `dmdadm`(8) and `dmcatadm`(8) commands. For example:

```
# dmdadm -j -u -c "load /dmf/scratch/daemon.txt"
# dmcatadm -j -m ls -u -c "load /dmf/scratch/ls_cat_txt" > /dmf/tmp/load.ls.db.out 2>&1
```

Use Sufficient Network Bandwidth for Socket Merges

If you perform a merge using a socket, you must ensure that the network has sufficient bandwidth. For more information, contact SGI technical support.

Installing and Configuring the DMF Environment

This chapter discusses the following:

Overview of the Installation and Configuration Steps

To install and configure the DMF environment, perform the following steps:

Procedure 4-1 Configuring the DMF Environment

1. Read "Installation and Configuration Considerations" on page 83.
2. Install the DMF server software (which includes the software for TMF and OpenVault) according to the instructions in the *SGI InfiniteStorage Software Platform* release note and any late-breaking caveats posted to Supportfolio:

   https://support.sgi.com

   See "ISSP DMF YaST Patterns" on page 84.
3. Determine the DMF drive groups that you want to use.
4. Configure the TMF or OpenVault mounting service (if used) according to the following documentation:
   - *TMF 5 Administrator's Guide for SGI InfiniteStorage*
   - *OpenVault Operator's and Administrator's Guide*
5. Determine how you want to complete periodic maintenance tasks. See "Automated Maintenance Tasks" on page 89.

6. Make and mount the DMF administrative filesystems. See "Configure DMF Administrative Filesystems and Directories Appropriately" on page 64.
7. Invoke `dmmaint`(8) to the install the DMF license (and optional DMF Parallel Data Mover Option license) on the primary DMF server and the passive DMF server (if applicable). See:
   - "Overview of `dmmaint`" on page 97
   - "Installing the DMF License" on page 99

   Note: Nodes running DMF client software do not require a DMF license.
8. Create or modify your configuration file and define objects for the following:
   - Pathname and file size parameters necessary for DMF operation (the `base` object)
   - DMF daemon
   - Daemon maintenance tasks
   - Filesystems
   - Automated space management
   - Media-specific process (MSP) or library server (LS)
   - MSP/LS maintenance tasks

   See one of the following:
   - Chapter 8, "Parallel Data Mover Option Configuration" on page 271
   - "Configuration Objects Overview" on page 149
9. Verify the configuration:
   - In DMF Manager, select the following:

     Overview
     > Configuration ...
     > Validate Current Configuration
   - In `dmmaint`, click the **Inspect** button

If there are errors, fix them and repeat the validation until there are no errors. (To edit the configuration in `dmmaint`, click the **Configure** button.)

10. If you are using the DMF Parallel Data Mover Option, see "Parallel Data Mover Option Configuration Procedure" on page 271 and the *SGI InfiniteStorage Software Platform* release note.
11. Start the DMF environment. See "Starting and Stopping the DMF Environment" on page 91.
12. If you want to install the DMF client packages on other systems, see the *SGI InfiniteStorage Software Platform* release note and the client installation `DMF.Install` instructions. Also see "DMF Client Configurations and `xinetd`" on page 84.

To administer and monitor DMF, see Chapter 6, "Using DMF Manager" on page 101.

Installation and Configuration Considerations

This section discusses the configuration considerations that will affect your system:

ISSP DMF YaST Patterns

The ISSP release includes the following DMF YaST patterns:

- **DMF Server**, which provides:
  - The full set of DMF server functionality, including the DMF daemon, infrastructure, user and administrator commands, and all man pages. This applies to SGI ia64 and SGI x86-64 servers running the operating system and SGI Foundation Software version as specified in the ISSP and DMF release notes. You should install this software only on those machines that can be the DMF server.
  - Client installers, which download the client software onto the server so that you can later transfer the DMF client software to the DMF client nodes. The client packages are installed along with their installation instructions on the server in the following directory:

    `/opt/dmf/client-dist/`*`DMFversion/clientOS&architecture`*

    The client software contains the limited set of user commands, libraries, and man pages. This applies to all supported operating systems. You should install this software on machines from which you want to give users access to DMF user commands, such as `dmput` and `dmget`.
- **DMF Parallel Data Mover**, which provides the infrastructure for parallel data mover nodes to move data offline to tape and retrieve it, plus the required underlying CXFS client-only software.

Only one of these patterns can be installed on a given machine.

DMF Client Configurations and `xinetd`

If your configuration includes DMF client platforms, you must ensure that the DMF server is running the `xinetd`(8) daemon. The `xinetd` daemon is enabled by default. If it has been disabled, you must reenable it at boot time via the following command:

```
dmfserver# chkconfig xinetd on
```

If `xinetd` is not running, you can start it immediately via the following command:

```
dmfserver# /usr/sbin/xinetd
```

See also "Set the `xinetd` `tcpmux` `instances` Parameter Appropriately" on page 72.

Filesystem Mount Options

DMAPI is the mechanism between the kernel and the XFS or CXFS filesystem for passing file management requests between the kernel and DMF. Ensure that you have installed DMAPI and the appropriate patches.

For filesystems to be managed by DMF, they must be mounted with the DMAPI interface enabled. Failure to enable DMAPI for DMF-managed user filesystems will result in a configuration error.

Do the following:

- Use the following command:

  # **`mount -o dmi -o mtpt =`** ***`mountpoint`***

- Add `dmi, mtpt =` *mountpoint* to the fourth field in the `fstab` entry

For more information, see:

- "`mkfs` and `mount` Parameters" on page 68
- The `mount`(8) and `fstab`(5) man pages

Tape Mounting Service Considerations

Tape mounting services are available through OpenVault or the Tape Management Facility (TMF). The LS checks the availability of the mounting service when it is started and after each occurrence in which an LS data mover process (either a *write child* that migrates data to tape or a *read child* that recalls data from tape) was unable to reserve its drive. If the mounting service is found to be unavailable, the LS does not start any new child processes until the mounting service is once again available.

If the unavailable mounting service is OpenVault, the LS sends an e-mail message to the administrator, asking that OpenVault be started. It then periodically polls OpenVault until it becomes available, at which time child processes are again allowed to run. For the LS, this is the default procedure. You can use `MAX_MS_RESTARTS` to configure the number of automatic restarts.

If the unavailable mounting service is TMF, the LS not only attempts to initiate `tmdaemon` if it is not up (based on the exit status of `tmstat`), but it waits until a

TMF device in the `configuration pending` state is configured up before it resumes processing. If TMF cannot be started or if no devices are configured up, the LS sends e-mail to the administrator and polls TMF until a drive becomes available. For the LS, this is the default procedure. You can use `MAX_MS_RESTARTS` to configure the number of automatic restarts.

Inode Size Configuration

In DMF user filesystems and DCM filesystems, DMF state information is kept within a filesystem structure called an *extended attribute.*

Extended attributes can be either inside the inode or in attribute blocks associated with the inode. DMF runs much faster when the extended attribute is inside the inode, because this minimizes the number of disk references that are required to determine DMF information. In certain circumstances, there can be a large performance difference between an inode-resident extended attribute and a non-resident extended attribute.

The size of inodes within a filesystem impacts how much room is available inside the inode for storing extended attributes. Smaller inode sizes have much less room available for attributes. Likewise, the legacy inode attribute format (`-i attr=1` option to `mkfs.xfs`) results in less available extended attribute space than does the current default format (`-i attr=2` option).

SGI recommends that you configure your filesystems so that the extended attribute is always inode-resident. Whenever both 256-byte and 512-byte inode sizes will work, you should use the 256-byte inode size (`-i size=256` option to `mkfs.xfs`) because the inode scans will be up to twice as fast. SGI also highly recommends that you use `attr2` (`-i attr=2` option) when possible if it allows 256-byte inode sizes to be used.

For optimal performance, you should create any DCM filesystems with 256-byte inode sizes and `attr2` attribute format. (For other DMF administrative filesystems, the inode size does not matter.) For best performance, you should use 512-byte inode sizes for DMF user filesystems only under the following circumstances:

- If users require other XFS attributes such as ACLs or other user-specified attributes
- If the user filesystem will have large numbers of partial-state files with more partial-state regions than will fit in a 256-byte inode

If you must have a 512–byte inode size for a DMF user filesystem, you can do so by using the Linux `mkfs.xfs` command with the `-i size=512` option. (Filesystems that already exist must be dumped, recreated, and restored.)

Table 4-1 summarizes the relationship among the inode size, `attr` type, and file regions.

Table 4-1 Default Maximum File Regions for XFS and CXFS Filesystems

Size of inode	`attr` Type	Default Maximum Number of File Regions
256	1	(Not recommended)
256	2	2
512 or greater	1	8
512 or greater	2	11

For more information about setting the inode size and the `attr` type, see the `mkfs.xfs`(8) or `mount`(8) man pages.

Daemon Database Record Length

A daemon database entry is composed of one or more fixed-length records:

- The base record (`dbrec`), which consists of several fields, including the `path` field
- Zero or more path segment extension (`pathseg`) records

If the value that is returned to the daemon by the MSP/LS (such as the pathname resulting from the `NAME_FORMAT` value template in an FTP or disk `msp` object) can fit into the `path` field of the daemon's `dbrec` record, DMF does not require `pathseg` records. If the MSP/LS supplies a path value that is longer than the `path` field, DMF creates one or more `pathseg` records to accommodate the extra space.

The default size of the `path` field of the `dbrec` is 34 characters. This size allows the default paths returned by `dmatls`, `dmdskmsp`, and `dmftpmsp` to fit in the `path` field of `dbrec` as long as the user name portion of the `dmftpmsp` or `dmdskmsp` default path (*username/bit_file_identifier*) is 8 characters or fewer. If you choose to use a value for `NAME_FORMAT` that results in longer pathnames, you may want to resize the `path` field in `dbrec` in order to increase performance.

The default size of the path field in the `pathseg` record is 64. For MSP path values that are just slightly over the size of the `dbrec` path field, this will result in a large amount of wasted space for each record that overflows into the `pathseg` record. The

ideal situation would be to have as few `pathseg` records as possible, because retrieving `pathseg` records slows down the retrieval of daemon database records.

The size of the path field in the daemon `dbrec` record can be configured at any time before or after installation. (The same holds true for any installation that might be using the `dmftpmsp` or `dmdskmsp` with a different path-generating algorithm or any other MSP that supplies a path longer than 34 characters to the daemon.)

Procedure 4-2 Configuring the Daemon Database Record Length

The steps to configure the daemon database entry length are as follows:

1. If `dmfdaemon` is running, use the following command to halt processing in a non-HA environment:

   

   Caution: For instructions about starting and stopping DMF and the mounting service in an HA environment, see the SGI HA documentation.

   ```
   # service dmf stop
   ```

2. If a daemon database already exists, perform the following commands:

   ```
   # cd HOME_DIR/daemon
   # dmdump -c . > textfile
   # cp dbrec* pathseg* dmd_db.dbd backup_dir
   # rm dbrec* pathseg* dmd_db.dbd
   ```

   Where:

   - *HOME_DIR* is the value of `HOME_DIR` returned by the `dmconfig base` command
   - *textfile* is the name of a file that will contain the text representation of the current daemon database
   - *backup_dir* is the name of the directory that will hold the old version of the daemon database

3. Change to the `rdm` directory:

   ```
   # cd /usr/lib/dmf/rdm
   ```

4. Back up the `dmd_db.dbd` and `dmd_db.ddl` files that reside in `/usr/lib/dmf/rdm`. This will aid in disaster recovery should something go wrong.

5. Edit `dmd_db.ddl` to set the new `path` field lengths for the `dbrec` and/or `pathseg` records.
6. Regenerate the new daemon database definition, as follows:

   # **`/usr/lib/dmf/support/dmddlp -drsx dmd_db.ddl`**

7. Back up the new versions of `dmd_db.dbd` and `dmd_db.ddl` for future reference or disaster recovery.
8. If the daemon database was dumped to text in step 2, enter the following commands:

   # **`cd` *`HOME_DIR`*`/daemon`**
   # **`dmdadm -u -c "load` *`textfile`*`"`**

   (*textfile* was created in step 2)

9. If the daemon was running in step 1, restart it by executing the following command:

   # **`service dmf start`**

Interprocess Communication Parameters

Ensure that the following interprocess communication kernel configuration parameters are set equal to or greater than the default before running DMF:

- `MSGMAX`
- `MSGMNI`

For more information, execute `info ipc` and see the `sysctl`(8) and `msgop`(2) man pages.

Automated Maintenance Tasks

DMF lets you configure parameters for completing periodic maintenance tasks such as the following:

- Making backups (full or partial) of user filesystems to tape
- Making backups of DMF databases to disk
- Removing old log files and old journal files

- Monitoring DMF logs for errors
- Monitoring the status of tapes in LSs
- Running hard deletes
- Running `dmaudit`(8)
- Merging tapes that have become sparse (and stopping this process at a specified time)

Each of these tasks can be configured in the DMF configuration file (`/etc/dmf/dmf.conf`) through the use of `TASK_GROUPS` parameters for the DMF daemon and the LS. The tasks are then defined as objects.

For each task you configure, a time expression defines when the task should be done and a script file is executed at that time. The tasks are provided in the `/usr/lib/dmf` directory.

The automated tasks are described in "`taskgroup` Object" on page 170.

`DMAPI_PROBE` Must Be Enabled for SLES 10 or SLES 11 Nodes When Using CXFS

By default, DMAPI is turned off on SLES 10 and SLES 11 systems. To mount CXFS filesystems on a SLES 10 or SLES 11 client-only node with the `dmi` mount option, you must set `DMAPI_PROBE="yes"` in the `/etc/sysconfig/sysctl` file on the node. Changes to the file will processed on the next reboot.

After setting that system configuration file, you can immediately enable DMAPI for the current boot session by executing the following:

```
# sysctl -w fs.xfs.probe_dmapi=1
```

Note: These steps are not required on the DMF server or DMF parallel data mover nodes because these steps are done automatically when installing the `dmf` or `dmf-mover` packages.

Starting and Stopping the DMF Environment

This section discusses the following:

For more information about the mounting services, see:

- *TMF 5 Administrator's Guide for SGI InfiniteStorage*
- *OpenVault Operator's and Administrator's Guide*

Caution: In an HA environment, procedures differ. For example, you must first remove Heartbeat control of the resource group before stopping DMF and the mounting service. See the SGI HA documentation.

Automatic Start After Reboot

To enable automatic startup of the DMF environment, execute the following `chkconfig`(8) commands as `root` on the DMF server in a non-HA environment: [1]

```
dmfserver# chkconfig tmf on (if TMF)
dmfserver# chkconfig openvault on (if OpenVault)
dmfserver# chkconfig dmf on
dmfserver# chkconfig dmfman on
```

Execute the following on the parallel data mover nodes:

```
pdmn# chkconfig dmf_mover on
```

[1] For instructions about starting and stopping DMF and the mounting service in an HA environment, see the SGI HA documentation.

Explicit Start

To start the DMF environment daemons explicitly, execute the following on the DMF server in a non-HA environment: [2]

```
dmfserver# service tmf start (if TMF)
dmfserver# service openvault start (if OpenVault)
dmfserver# service dmf start
dmfserver# service dmfman start
```

Execute the following on the parallel data mover nodes:

```
pdmn# service dmf_mover start
```

Preventing Automatic Start After Reboot

To prevent automatic startup of the DMF environment, execute the following `chkconfig`(8) commands as `root` on the DMF server in a non-HA environment: [3]

```
dmfserver# chkconfig tmf off (if TMF)
dmfserver# chkconfig openvault off (if OpenVault)
dmfserver# chkconfig dmf off
dmfserver# chkconfig dmfman off
```

Execute the following on the parallel data mover nodes:

```
pdmn# chkconfig dmf_mover off
```

2 For instructions about starting and stopping DMF and the mounting service in an HA environment, see the SGI HA documentation.

3 For instructions about starting and stopping DMF and the mounting service in an HA environment, see the SGI HA documentation.

Explicit Stop

To stop the DMF environment daemons explicitly, execute the following on the DMF server in a non-HA environment: [4]

```
dmfserver# service tmf stop  (if TMF)
dmfserver# service openvault stop (if OpenVault)
dmfserver# service dmf stop
dmfserver# service dmfman stop
```

Execute the following on the parallel data mover nodes:

```
pdmn# service dmf_mover stop
```

Note: Executing `service dmf_mover stop` on a mover node will cause existing data mover processes to exit after the library server notices this change, which may take up to 2 minutes. The existing data mover processes may exit in the middle of recalling or migrating a file; this work will be reassigned to other data mover processes.

Customizing DMF

You can modify the default behavior of DMF as follows:

- "File Tagging" on page 93
- "Site-Defined Policies" on page 94

File Tagging

File tagging allows an arbitrary 32-bit integer to be associated with specific files so that they can be subsequently identified and acted upon. The specific values are chosen by the site; they have no meaning to DMF.

Non-`root` users may only set or change a tag value on files that they own, but the `root` user may do this on any files. The files may or may not have been previously migrated.

[4] For instructions about starting and stopping DMF and the mounting service in an HA environment, see the SGI HA documentation.

To set a tag, use the dmtag(1) command or the libdmfusr.so library. For example:

```
% dmtag -t 42 myfile
```

To view the tag set for a given file, use the dmtag or dmattr commands. For example:

```
% dmtag myfile
42  myfile
% dmattr -a sitetag myfile
42
```

A file's tag (if any) can be tested in the when clause of the following configuration parameters by using the keyword sitetag:

```
AGE_WEIGHT
CACHE_AGE_WEIGHT
CACHE_SPACE_WEIGHT
SELECT_LOWER_VG
SELECT_MSP
SELECT_VG
SPACE_WEIGHT
```

For example:

```
SELECT_VG fasttape when sitetag = 42
```

It may also be accessed in site-defined policies, as described below.

For more information, see the dmtag(1) man page.

Site-Defined Policies

Site-defined policies allow you to do site-specific modifications by writing your own library of C++ functions that DMF will consult when making decisions about its operation. For example, you could write a policy that decides at migration time which volume group (VG) or MSP an individual file should be sent to, using selection criteria that are specific to your site.

Note: If you customize DMF, you should inform your users so that they can predict how the user commands will work with your policies in place. You can add error, warning, and informational messages for commands so that the user will understand why the behavior of the command differs from the default.

For information about the aspects of DMF that may be modified, see Appendix C, "Site-Defined Policy Subroutines and the `sitelib.so` Library" on page 433.

Importing Data From Other HSMs

DMF utilities exist to assist with the import of data from filesystems managed by other HSM packages into DMF, provided that the filesystems to be imported are accessible via FTP or as local or NFS-mounted filesystems. These tools are not distributed with the DMF product. They are for use only by qualified SGI personnel who assist sites doing conversions. To obtain assistance in performing a conversion, contact SGI Support.

Using dmmaint to Install Licenses and Configure DMF

On DMF servers, you can use dmmaint to install your DMF licenses and edit the DMF configuration file. The advantage to using dmmaint rather than a text editor such as vi is that you can edit the configuration file and apply your changes atomically. dmmaint also allows you to verify your changes.

This chapter discusses the following:

You can also use DMF Manager to configure DMF. See "Configuring DMF with DMF Manager" on page 114.

Overview of dmmaint

To use the dmmaint graphical user interface (GUI), ensure that your DISPLAY environment variable is defined, and then enter the following command:

```
# /usr/sbin/dmmaint &
```

Note: If DISPLAY is not defined, dmmaint reverts to line mode, which has menu selections that are equivalent to the fields and buttons on the graphic user interface. Line mode is provided for remote log in, and is not recommended for general use.

The GUI displays the installed version of DMF. The **Help** menu provides access to the dmmaint and dmf.conf man pages.

The GUI buttons are as follows:

Button	Description
Configure	Lets you customize the DMF configuration file for the selected version of DMF. If this is the first time you have configured DMF, a window appears telling you that there is no configuration file. You are asked which file you would like to use as a basis for the new configuration. You may choose an existing file or one of several sample files that are preconfigured for different types of media-specific process (MSP) or the library server (LS). If a configuration file exists, a window appears that asks if you would like to modify the existing file or use an alternate file. If you choose an alternate file, you see the same window that you would see if this were a new configuration. After you choose a file to use as a basis, an editing session is started (in a new window) that displays a copy of that configuration file. You can make changes as desired. After exiting from the editor, you are prompted for confirmation before the original configuration file is replaced with the edited copy. For more information on configuration parameters, see Chapter 7, "DMF Configuration File" on page 149, and the `dmf.conf`(5) man page (available from the **Help** button).
Inspect	Runs the `dmcheck`(8) program to report errors. You should run this program after you have created a configuration file. If there are errors, you can click the **Configure** button, make changes, and continue to alternate between **Configure** and **Inspect** until you are satisfied that the configuration is correct.
Release Note	This button displays the DMF release note that is installed in `/usr/share/doc/packages/sgi-issp-`*ISSPversion*`/README_DMF.txt`

License Info	Displays the hostname and unique system identifier (which you need to obtain a DMF server license), the name of the license file, and a short description of the state of any DMF license within the file.
Update License	Lets you make changes to the license file. An editing session is started in a new window displaying a copy of the contents of the license file. You can add or delete licenses as desired. After you exit the editor, positive confirmation is requested before the original license file is replaced by the modified copy. For more information, see Chapter 2, "DMF Licensing" on page 47.

Installing the DMF License

To install the DMF license, do the following:

1. Select **Dependencies** to read about all the hardware and software requirements that must be fulfilled before running DMF.
2. If needed, select the **Update License** button and use the mouse to copy and paste your license into the file. Close the window. Select **License Info** and examine the output to verify that the license is installed correctly.

Using `dmmaint` to Define the Configuration File

To use `dmmaint` to configure DMF, do the following:

1. Select **Configure** to edit the configuration file. For more information about this button, see "Overview of `dmmaint`" on page 97.
2. Click the **Inspect** button, which runs `dmcheck` to report any errors in the configuration. If there are errors, you can click the **Configure** button, make changes, and continue to alternate between **Configure** and **Inspect** until you are satisfied that the configuration is correct.

If you do not want DMF to be automatically started and stopped, see "Starting and Stopping the DMF Environment" on page 91.

Chapter 6

Using DMF Manager

This chapter discusses the following:

Accessing DMF Manager

This section discusses the following:

Standard URL Access for DMF Manager

To access DMF Manager, do the following:

1. Point your browser to the following secure address:

   `https://`*YOUR_DMF_SERVER*`:1179`

2. Accept the security certificate.

   Note: DMF Manager generates its own SSL certificates, rather than having the SSL certificates signed by a commercial certificate authority. Therefore, the certificate warning is safe to ignore.

3. Enter the DMF Manager `dmfman` access password. The default access password is `INSECURE`.

   Note: You should change the access password after installing DMF and only provide it to those persons who you want to access the GUI.

   Also see "Running Observer Mode or Admin Mode" on page 105.

URL Redirection for DMF Manager

You can optionally have DMF Manager automatically redirect the following address to the secure `1179` port address:

`http://`*YOUR_DMF_SERVER*`/dmfman`

This redirection requires that `apache2` is started. To determine if `apache2` is running, enter the following:

```
dmfserver# service apache2 status
```

If apache2 is not started and you want to use URL redirection, do the following:

- To start apache2 for the current session:

  ```
  dmfserver# service apache2 start
  ```

- To start apache2 across reboots:

  ```
  dmfserver# chkconfig apache2 on
  ```

Getting Started with DMF Manager

DMF Manager lets you configure DMF, view the current state of your DMF system, and make operational changes.

When you initially open DMF Manager, you will see the **Overview** panel, which displays a high-level graphical view of the DMF environment and status for each DMF component, as shown in Figure 6-1. You can also configure DMF from this panel.

Each menu bar selection provides access to a DMF Manager panel, described in Table 6-1. To open a panel, click on the panel name in the menu. Right-click on the tab title to see its menu. Each panel has a key for its symbols.

Note: DMF Manager windows do not automatically update; choose the **Refresh** menu item to update an existing view.

Table 6-1 DMF Manager Panel Menus

Menu Bar Item	Panel Name	Description
Configuration	**Overview**	High-level graphical view of the DMF environment, status for each DMF component, and configuration capability
	Parameters	Details about the current parameter settings in the DMF configuration file and status for each DMF component
Storage	**Tapes**	Tape status
	Libraries	Library status
Messages	**Reports**	Daily activity reports
	Alerts	Informational, warning, and error messages
	DMF Manager Tasks	Status of commands issued via DMF Manager that may take time to complete
Statistics	**DMF Resources**	Current and historical reports about the state and the performance of the DMF filesystems and hardware
	DMF Activity	Current and historical reports about the state and the performance of the DMF requests and throughput
Help	**Getting Started**	This section
	Admin Guide	This manual
	About DMF Manager	Version and copyright information about DMF Manager

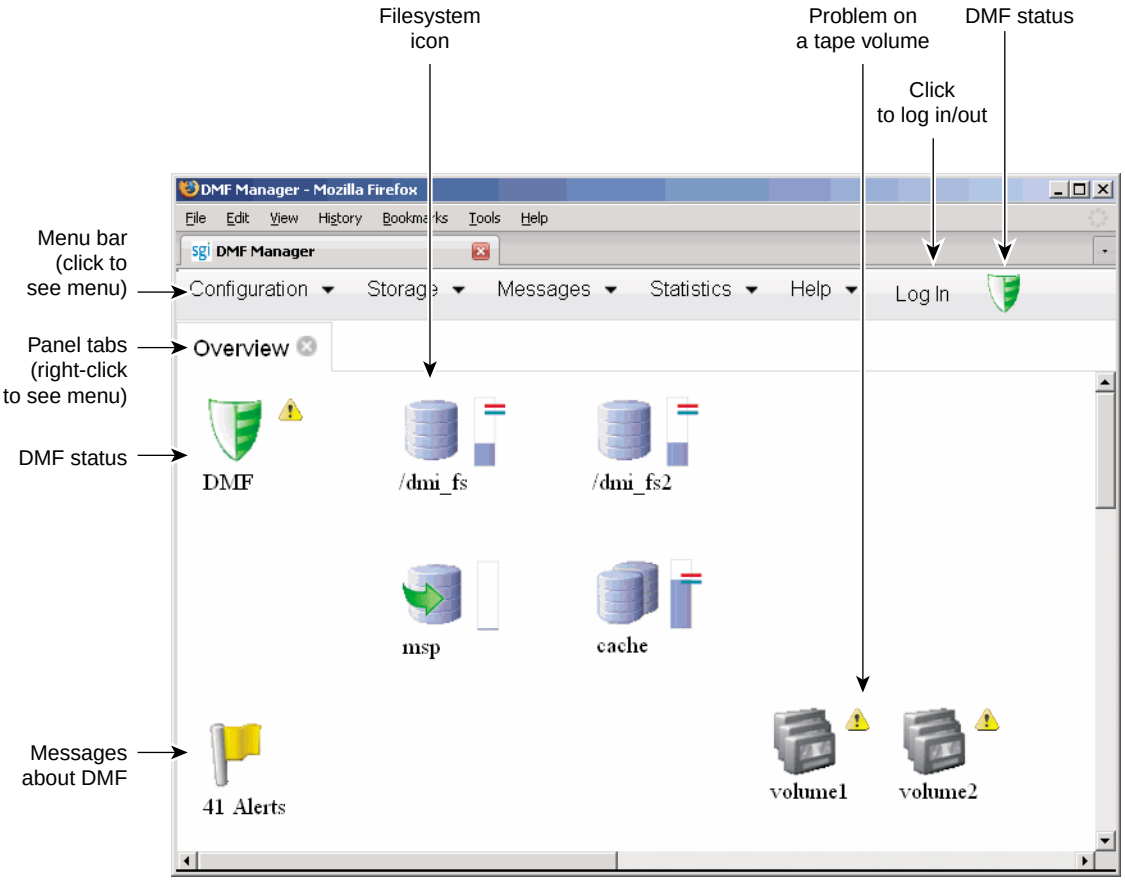

Figure 6-1 DMF Manager Overview Panel

Running Observer Mode or Admin Mode

You can run DMF Manager in observer mode (the default) or you can log in to Admin mode for more functionality, as described in the following sections:

- "Observer Mode Functionality" on page 106
- "Admin Mode Functionality" on page 107
- "Admin Mode Access" on page 108

Observer Mode Functionality

In the default observer mode in DMF Manager, you can do the following:

- View the state of DMF and the mounting service (enabled / disabled). See:
  - "Getting Started with DMF Manager" on page 103
  - "Discovering DMF Problems" on page 127
- View the fullness of each DMF-managed filesystem. See:
  - "Getting Started with DMF Manager" on page 103
  - "Discovering DMF Problems" on page 127
- View the licensed capacity. See "Determining the DMF License Capacity" on page 126.
- View DMF's configuration. See "Displaying DMF Configuration File Parameters" on page 125.
- View relationships among DMF components. See "Seeing Relationships Among DMF Components" on page 130.
- Get context-sensitive help and view the DMF administration guide. See "Getting More Information in DMF Manager" on page 108.
- View information about tapes:
  - View each tape cartridge's status and fullness
  - Sort which tapes to view
  - Display dump tapes
  - View the status of each tape drive
  - Temporarily create a user-defined tape query

Note: Saving the query requires Admin mode. See "Admin Mode Functionality" on page 107.

See:

- "Getting Started with DMF Manager" on page 103
- "Managing Tapes" on page 131

- View the daily reports (with history) and DMF alerts. See "Discovering DMF Problems" on page 127.
- View the long-running DMF Manager tasks (those currently executing and a history of executed tasks). See "Displaying DMF Manager Tasks" on page 134.
- View current and historical reports about DMF activity and resources. See "Monitoring DMF Performance Statistics" on page 135.

Admin Mode Functionality

If you log in to Admin mode, you can perform the following additional tasks:

- Start/stop DMF and the mounting service (in non-HA environments). See "Starting and Stopping DMF and the Mounting Service" on page 127.
- Create or modify the DMF configuration. See "Configuring DMF with DMF Manager" on page 114.
- Manage tapes:
  - Create and save tape cartridge queries
  - Change the hold flags on tape cartridges
  - Manually mark tapes to be sparsed during next DMF sparsing run
  - Empty a damaged tape of all useful data and restore another copy in the volume group (VG)
  - Eject tape cartridges from the library
  - Load tape cartridges into the library and configure them for DMF's use with OpenVault
  - Read data from tape cartridge to verify tape's integrity
  - Enable/disable tape drives

  See "Managing Tapes" on page 131.

- Acknowledge/unacknowledge DMF alerts. See "Discovering DMF Problems" on page 127.
- Control long-running DMF Managed tasks (pause, kill, resume, acknowledge). See "Displaying DMF Manager Tasks" on page 134.

Admin Mode Access

To log in to DMF Manager as the Admin user, click the **Log In** button in the upper-right corner of the window, as shown in Figure 6-1 on page 105.

When logged in, you can also choose to reset the Admin password. The default Admin password is `INSECURE`.

Note: You should change this password after installing DMF and only provide it to those persons who have permission to make DMF administrative changes.

Getting More Information in DMF Manager

Each panel that uses icons has a key for its symbols, available via the **Show Key** menu selection. Figure 6-2 shows the key to icons on the **Overview** panel.

Figure 6-2 DMF Manager Key to Symbols

To display information about an object, you can move the mouse button over the object, as shown for the `/dmi_fs2` filesystem in Figure 6-3.

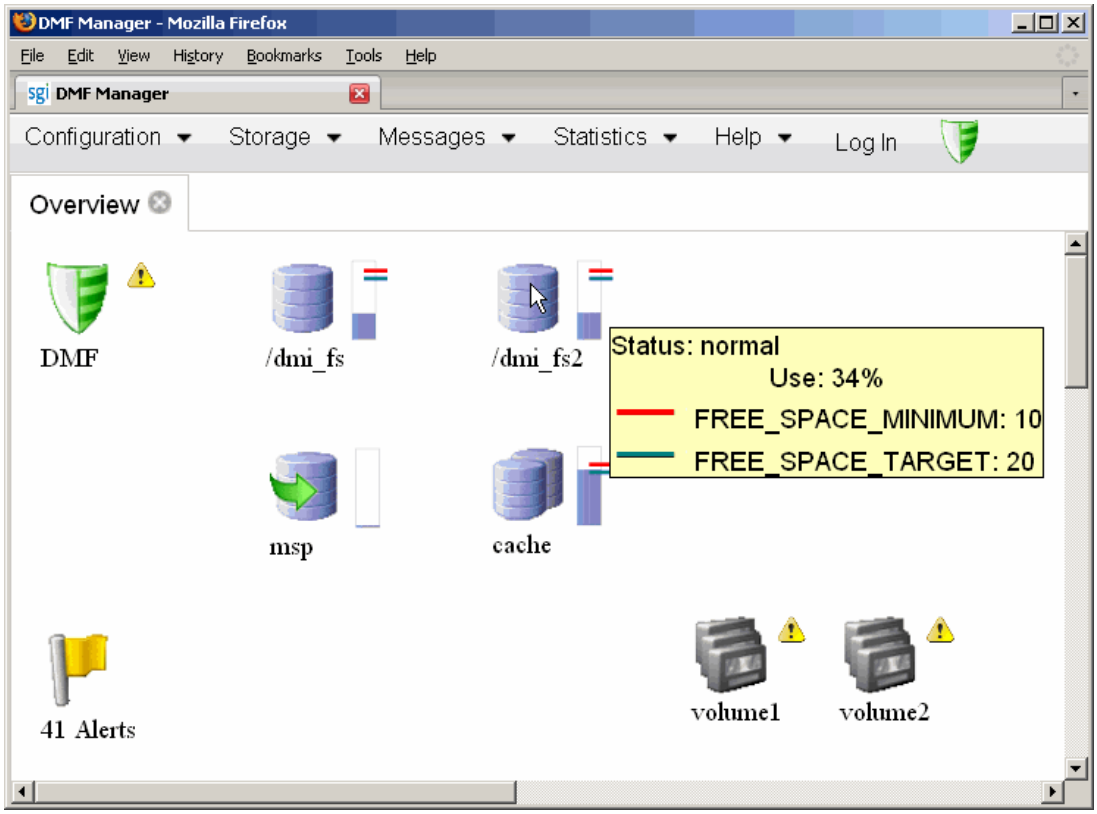

Figure 6-3 Displaying Information About an Icon

To get more information about any item, right-click on it and select the **What is this?** option. For example, Figure 6-4 shows the help text for the **Alerts** icon.

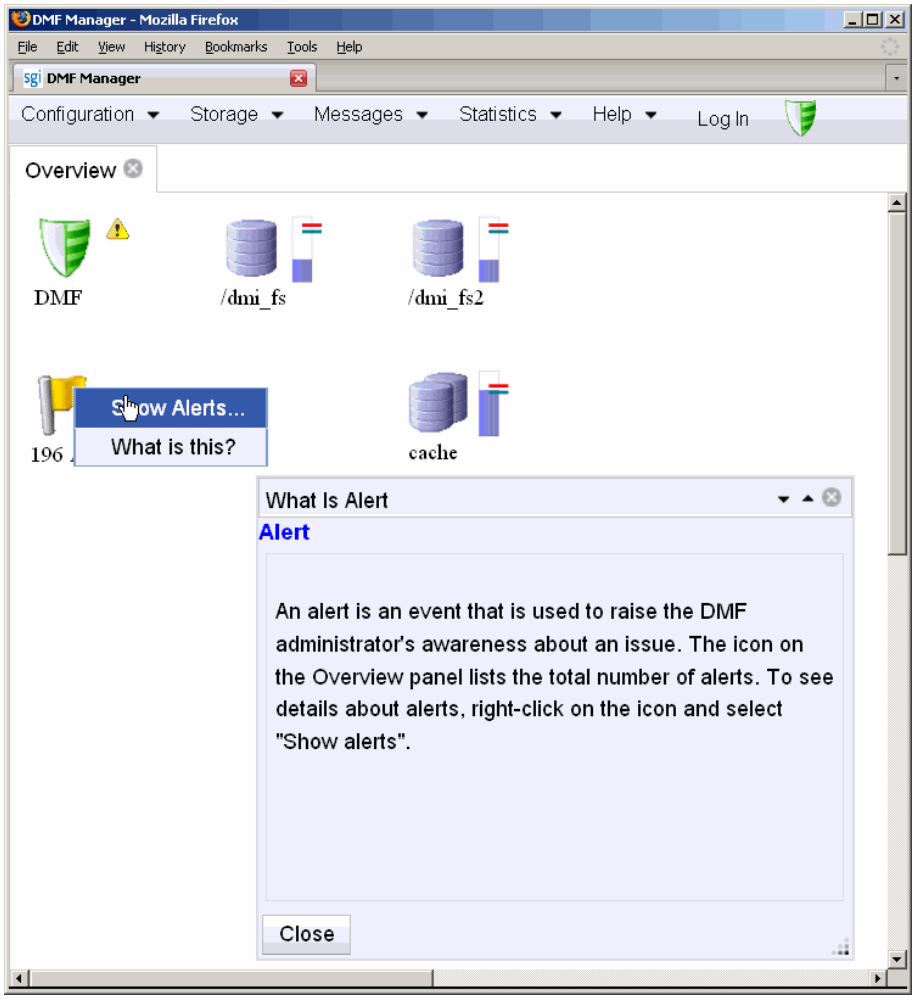

Figure 6-4 "What Is ..." Information

Each panel also has a **What is '*PanelName*'?** menu selection.

For a quick-start to using DMF Manager, select the following from the menu bar:

Help
> Getting Started

To access the DMF administrator's guide (this manual), select the following:

Help
> Admin Guide

Setting Panel Preferences

Each DMF Manager panel (other than the **Help** panels) has a **Set *PanelName* Preferences** menu item that allows you to vary what is shown on the panel, how it behaves, and how often it is refreshed (see "Refreshing the View" on page 114).

For example, Figure 6-5 shows the preferences that you can set for the **Overview** panel.

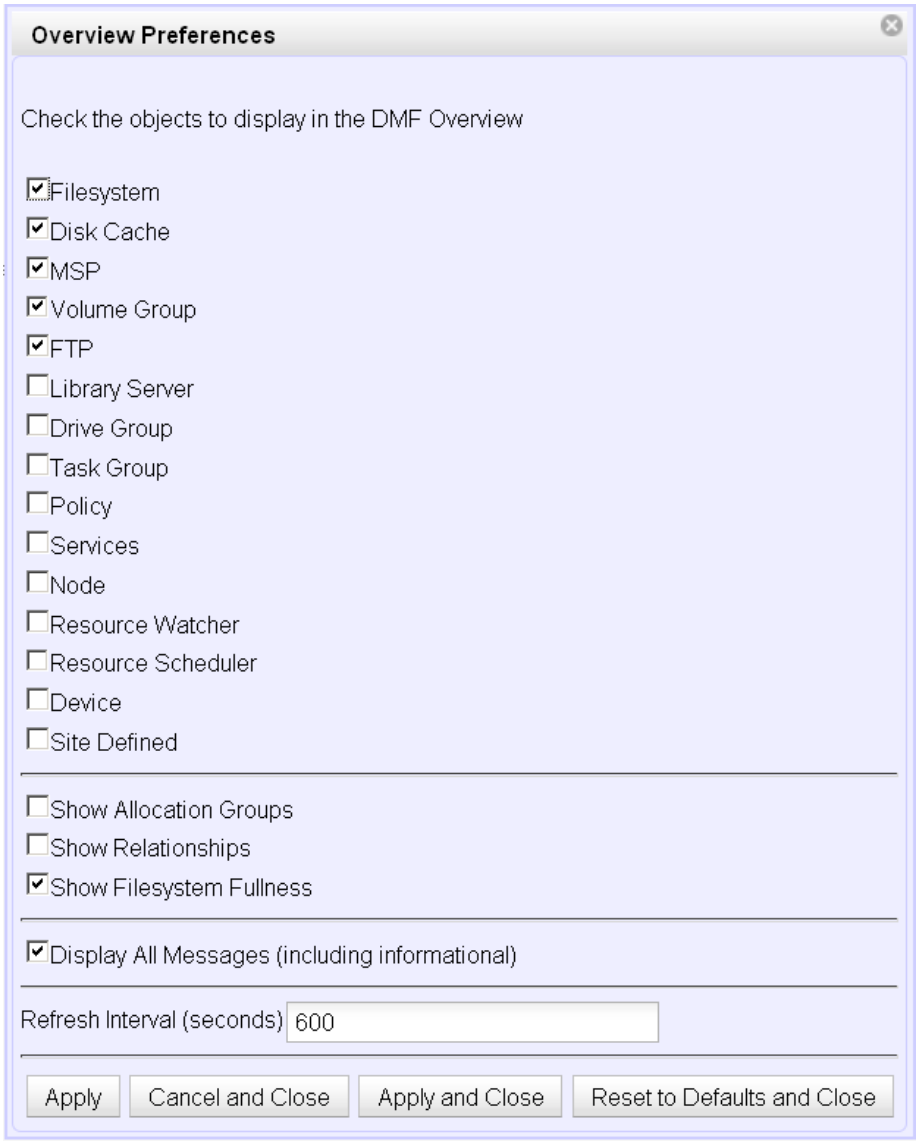

Figure 6-5 DMF Manager Overview Preferences Panel

Refreshing the View

In general, the DMF Manager panels do not automatically update by default. To refresh a panel, choose the **Refresh *PanelName*** menu item. The exception to this is the **Overview** panel, which by default automatically refreshes every 10 minutes (600 seconds) unless you are in configuration mode.

Caution: If you refresh the view while in configuration mode, any changes that have been made but not saved or applied will be lost and you will exit from configuration mode.

You can use **Set *PanelName* Preferences** menu to set an automatic refresh interval for individual panels. See "Setting Panel Preferences" on page 112.

Note: A refresh interval that is too short can cause contention between the DMF server and the browser. On heavily used systems, some displays may not be refreshed at extremely low intervals because the time to gather the information exceeds the refresh time. In such cases, you will only see a refresh as often as one can be completed.

Configuring DMF with DMF Manager

You can establish and edit the DMF configuration by logging in as the Admin user and using the **Overview** panel. If you make a change to the configuration, the background color will change to gray, indicating that you must save or cancel your changes.

This section discusses the following:

Limitations to the DMF Configuration Capability

The configuration capability in DMF Manager has the following limitations:

- Comments are not permitted in the configuration file created or modified by DMF Manager. If you edit an existing configuration file that has comments and save the file, the comments will be deleted from the updated configuration file (`/etc/dmf/dmf.conf`).

  Note: The original DMF configuration file, including the comments, will be preserved in a time-stamped copy (`/etc/dmf/dmf.conf.`*TIMESTAMP*).

- **Adding** site-specific objects or site-specific parameters is not supported (if site-specific items already exist in the DMF configuration file, they are preserved).
- DMF Manager does not have the ability to install or verify DMF licenses (you should use `dmmaint` for this process).
- DMF Manager cannot detect if multiple users have logged in as Admin and are therefore capable of overwriting each other's changes. Only one user should be logged in as Admin and make configuration changes at any given time.

Showing All Configured Objects

To see all currently configured objects, select:

Overview
> Configure...
> Show All Configured Objects

By default, all currently configured objects will also be shown after you make a configuration change and select **Continue**.

After you either save or cancel the configuration changes, the icons that are displayed will return to the preferences you have set. See "Setting Panel Preferences" on page 112.

Setting Up a New DMF Configuration File

To create a new DMF configuration file, right-click in the **Overview** panel and select one of the preconfigured items, such as **DCM MSP Sample**, or choose a specific object, like **Base**. The following figures show examples of the menus.

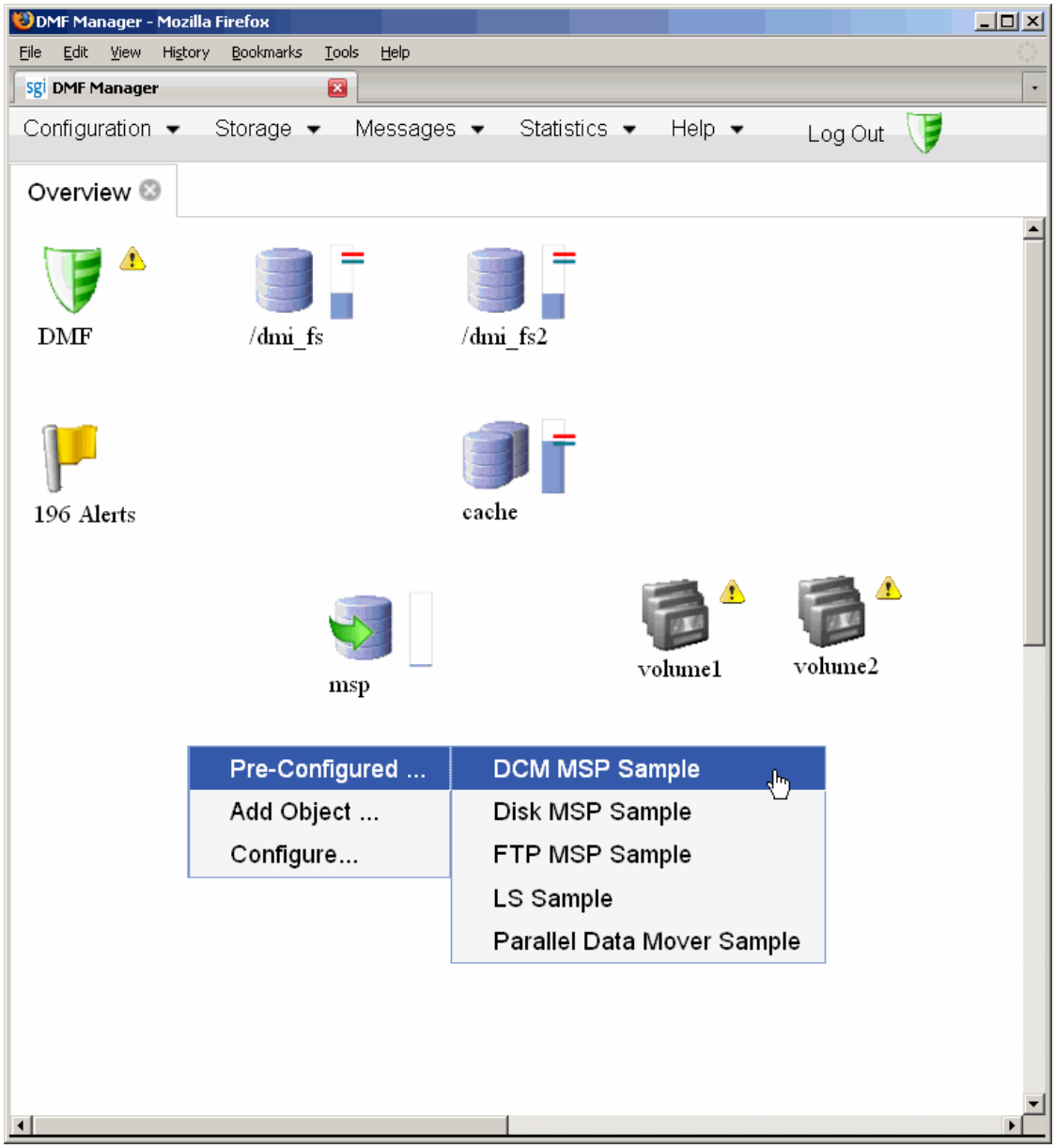

Figure 6-6 Configuration Template Menu

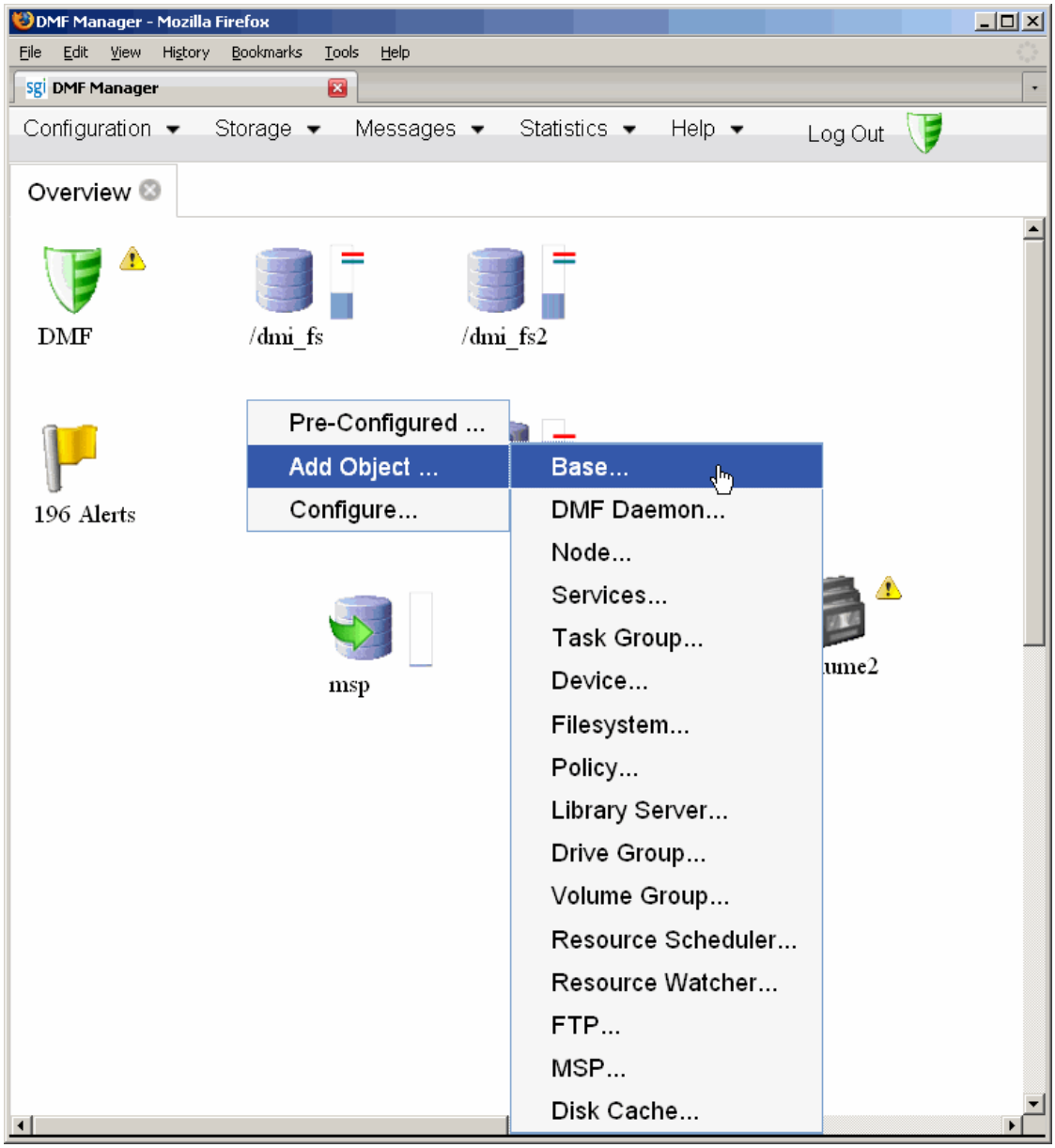

Figure 6-7 Configuration Object Menu

The preconfigured items provide a set of required objects that you can modify with the specific information for your site. For example, the **DCM MSP Sample** provides the following objects:

- `base` object and `dmdaemon` object (represented by the DMF shield icon):
- `filesystem` object named `/dmi_fs`
- `msp` object named `dsk1` configured for a disk MSP
- `policy` objects named `space_policy` and `msp_policy` configured for automated space management and MSP selection
- `taskgroup` object

Figure 6-8 shows the icons that are provided when you select the **DCM MSP Sample**. The gray background indicates that a change has been made that must be applied to the current configuration file, saved to a temporary location, or cancelled.

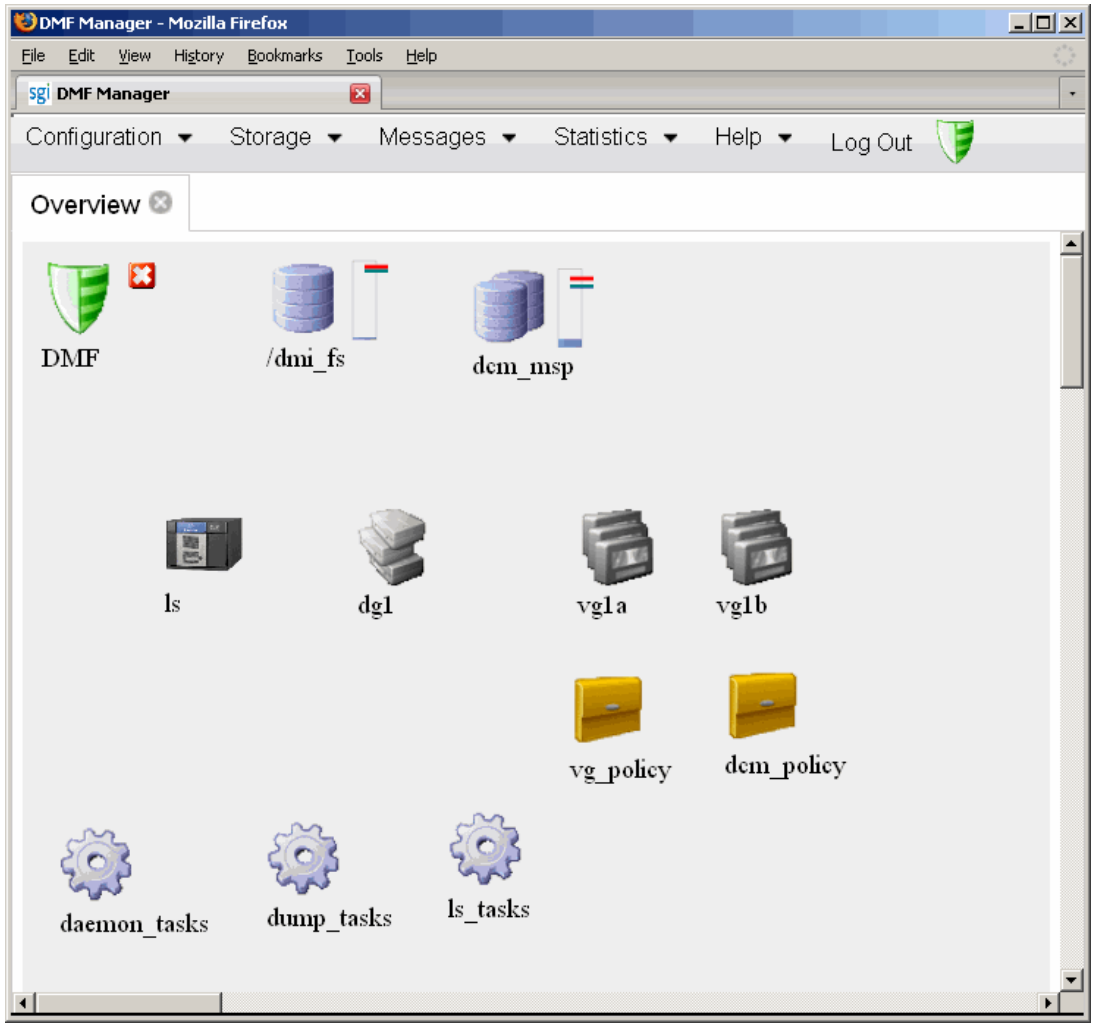

Figure 6-8 Temporary Workspace for a Preconfigured Disk MSP Sample (showing gray background)

To edit each object, select its icon and choose:

Configure ...
> Modify

Type in the values you desire for the parameters shown. To get more information a parameter, right-click on it and select the **What is this?** option. See "Getting More Information in DMF Manager" on page 108.

To make your changes appear in the **Overview** display, select **Continue** on the parameter window. To permanently save your changes, see "Saving Your Configuration Changes" on page 124.

To exit a preconfigured sample without saving any of your changes, select:

Overview
> Configure...
> Cancel Configuration

The **Configure** menu is also available by right-clicking within the **Overview** display.

Note: Many parameters have default values, but these are not shown in the DMF Manager windows. Only those parameters with specified values are added to the configuration file. If a parameter has no value specified, its default value is assumed.

Copying an Object

To copy an object, right-click on it and select:

Configure ...
> Copy

For example, to copy the `/dmi_fs` filesystem configuration object, place the icon over the `/dmi_fs` icon shown in Figure 6-9.

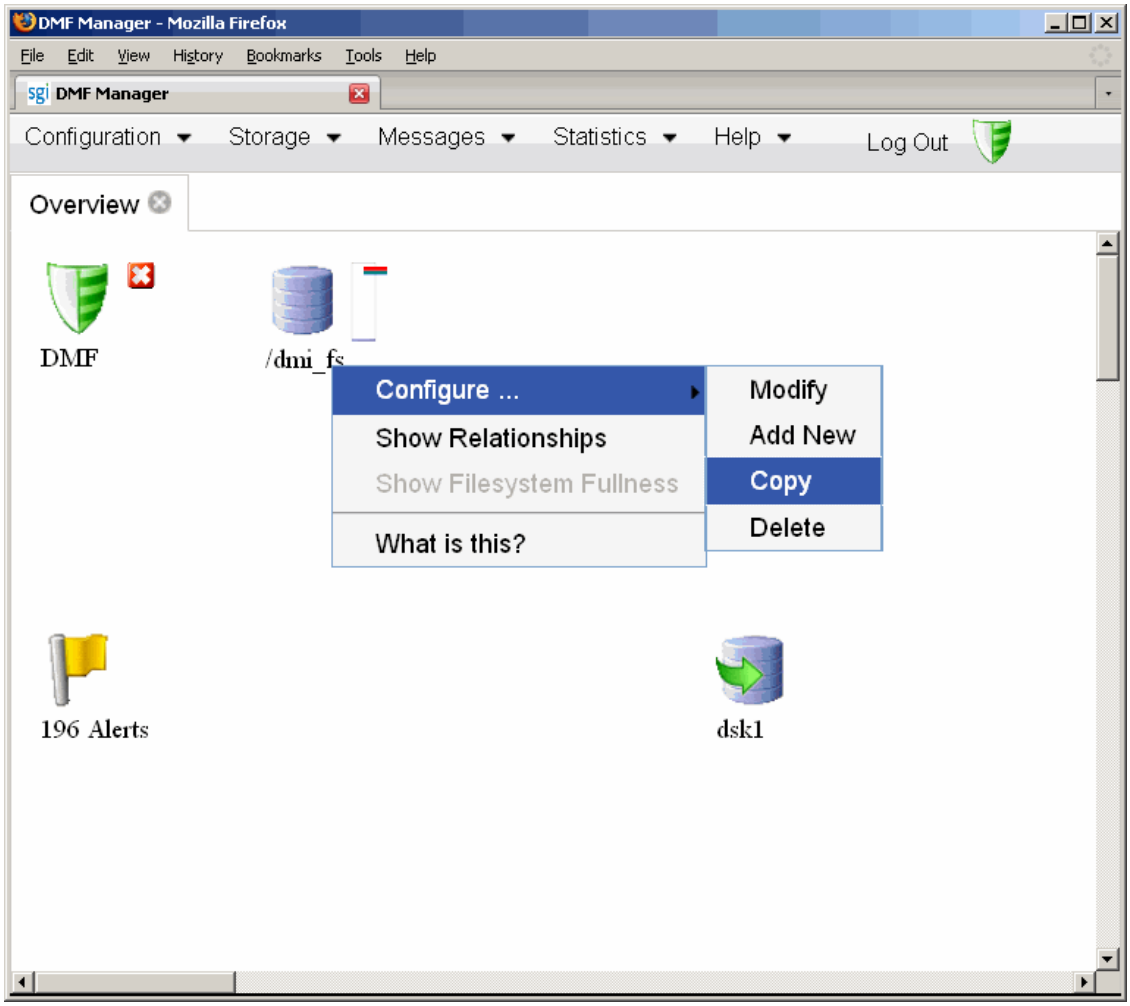

Figure 6-9 Copying an Object

Then name the new object and type in the values you desire for the parameters shown. To get more information a parameter, right-click on it and select the **What is this?** option. See "Getting More Information in DMF Manager" on page 108.

To make your changes appear in the **Overview** display, select **Continue**. To permanently save your changes, see "Saving Your Configuration Changes" on page 124.

Modifying an Object

To edit the parameters for an existing object, right-click on it and select:

Configure ...
> Modify

Then type in the values you desire for the parameters shown. To get more information a parameter, right-click on it and select the **What is this?** option. See "Getting More Information in DMF Manager" on page 108.

To rename an object, delete it and create a new object. See:

- "Deleting an Object" on page 124
- "Creating a New Object" on page 123
- "Copying an Object" on page 121

To make your changes in the temporary configuration view, select **Continue**. To permanently save your changes, see "Saving Your Configuration Changes" on page 124.

Creating a New Object

To create a new object, right-click on blank space anywhere in the **Overview** panel and select the object. Also see "Setting Up a New DMF Configuration File" on page 116.

You can also right-click on an existing object and create another empty object of the same type by selecting:

Configure ...
> Add New

Then name the object and type in the values you desire for the parameters shown. To get more information a parameter, right-click on it and select the **What is this?** option. See "Getting More Information in DMF Manager" on page 108.

To make your changes appear in the **Overview** display, select **Continue**. To permanently save your changes, see "Saving Your Configuration Changes" on page 124. Also see "Exiting the Temporary Configuration without Saving" on page 125.

Deleting an Object

To delete an object, right-click on it and select:

Configure ...
> Delete

Validating Your Changes

To verify that your changes to the temporary configuration are valid, select the following:

Overview
> Configure ...
> Validate Configuration

Saving Your Configuration Changes

Note: Before saving or applying configuration changes, you must make and mount the DMF administrative filesystems. See "Configure DMF Administrative Filesystems and Directories Appropriately" on page 64.

To make your changes appear in the **Overview** display for this DMF Manager session, click **Continue** after creating or modifying an object. (This does not change the DMF configuration file.)

To save the temporary configuration so that you can work on it later, select:

Overview
> Configure ...
> Save Temporary Configuration

To permanently save your changes and apply them to the DMF configuration file `/etc/dmf/dmf.conf`, do the following:

1. Verify that your changes are valid. See "Validating Your Changes" on page 124
2. Select:

   Overview
   > Configure ...
   > Apply Configuration

Exiting the Temporary Configuration without Saving

To exit the temporary configuration entirely without saving any of your changes, select:

Overview
> Configure...
> Cancel Configuration

The **Configure** menu is also available by right-clicking within the **Overview** display. If you refresh the screen, the temporary configuration will also be canceled.

Displaying DMF Configuration File Parameters

The following menu bar selection displays the contents of the DMF configuration file:

Configuration
> Parameters

For example, Figure 6-10 shows the configuration parameters for a drive group. For information about any individual parameter, right-click on it and select the **What is** option.

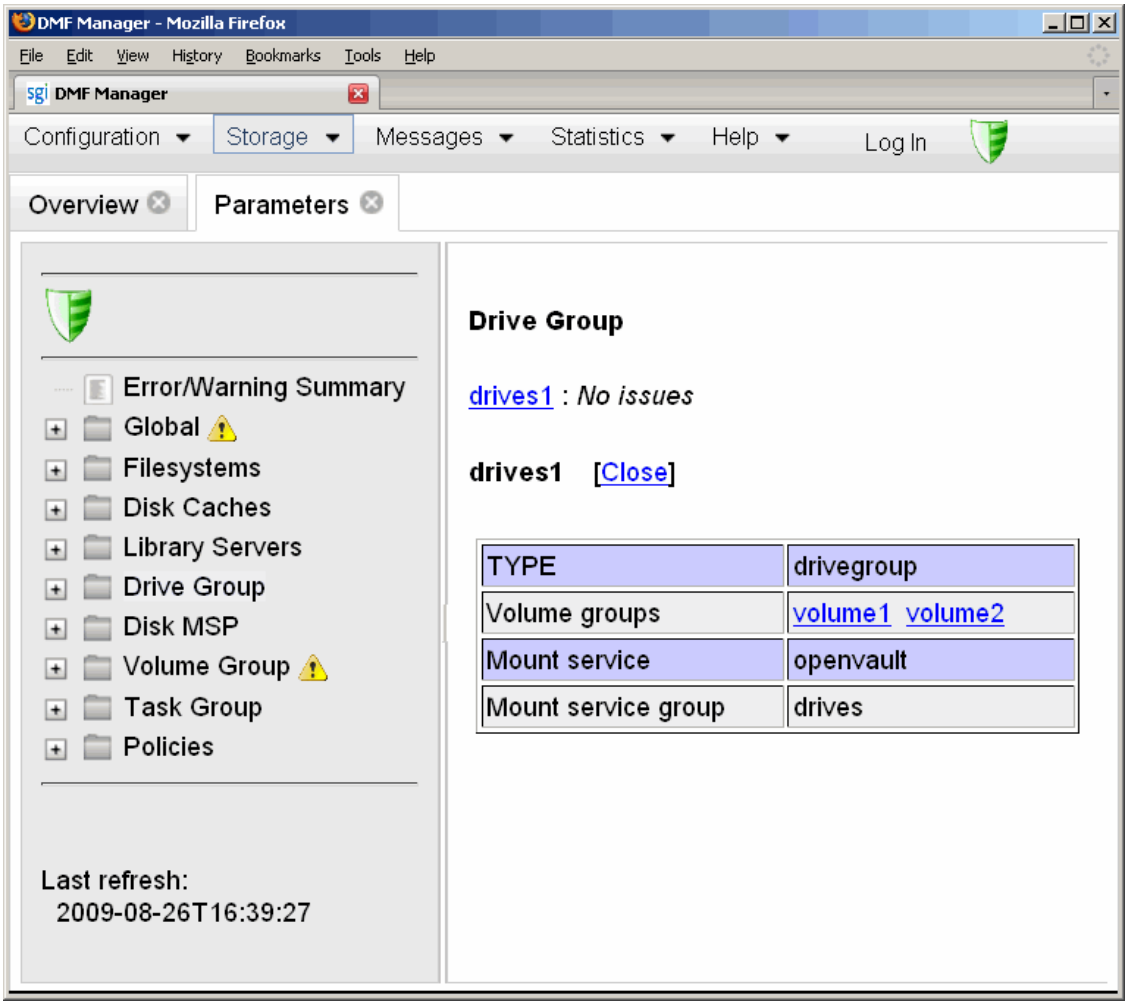

Figure 6-10 DMF Configuration Parameters in DMF Manager

Determining the DMF License Capacity

To determine the current DMF license capacity, right-click on the DMF icon in the **Overview** panel and select **Show Usage**. This will display a pop-up window showing the various media-specific processes (MSPs) and library servers (LSs), the number of bytes managed, and the total DMF license capacity.

Starting and Stopping DMF and the Mounting Service

To start or stop DMF and the mounting service, do the following:

1. Log in as the Admin user.
2. Right-click on the DMF icon in the **Overview** panel.
3. Select the desired action.

Caution: For instructions about starting and stopping DMF and the mounting service in an HA environment, see the SGI HA documentation.

Discovering DMF Problems

DMF Manager denotes areas with problems by adding a red icon next to the component that is experiencing problems. For example, Figure 6-11 shows that although DMF is still running, there is a problem. To investigate, hover the mouse over the shield icon to display pop-up help that details the warning.

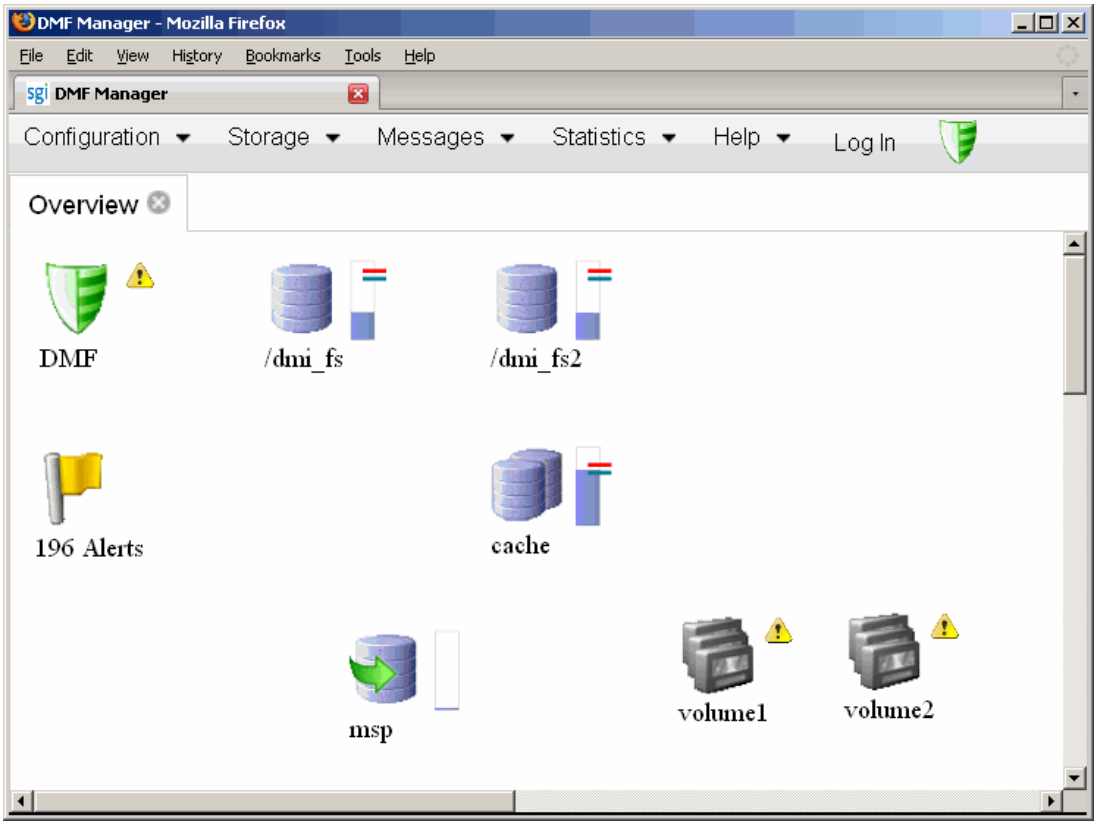

Figure 6-11 DMF Manager Showing Problems in the DMF System

For more information, click the **Alerts** icon flag and select **Show Alerts...** or choose the following from the menu bar:

Messages
> Alerts

Either action will open the **Alerts** panel, which displays the unacknowledged alerts by day or month, as shown in Figure 6-12. For more information about an alert, select it and choose **Help on this alert**.

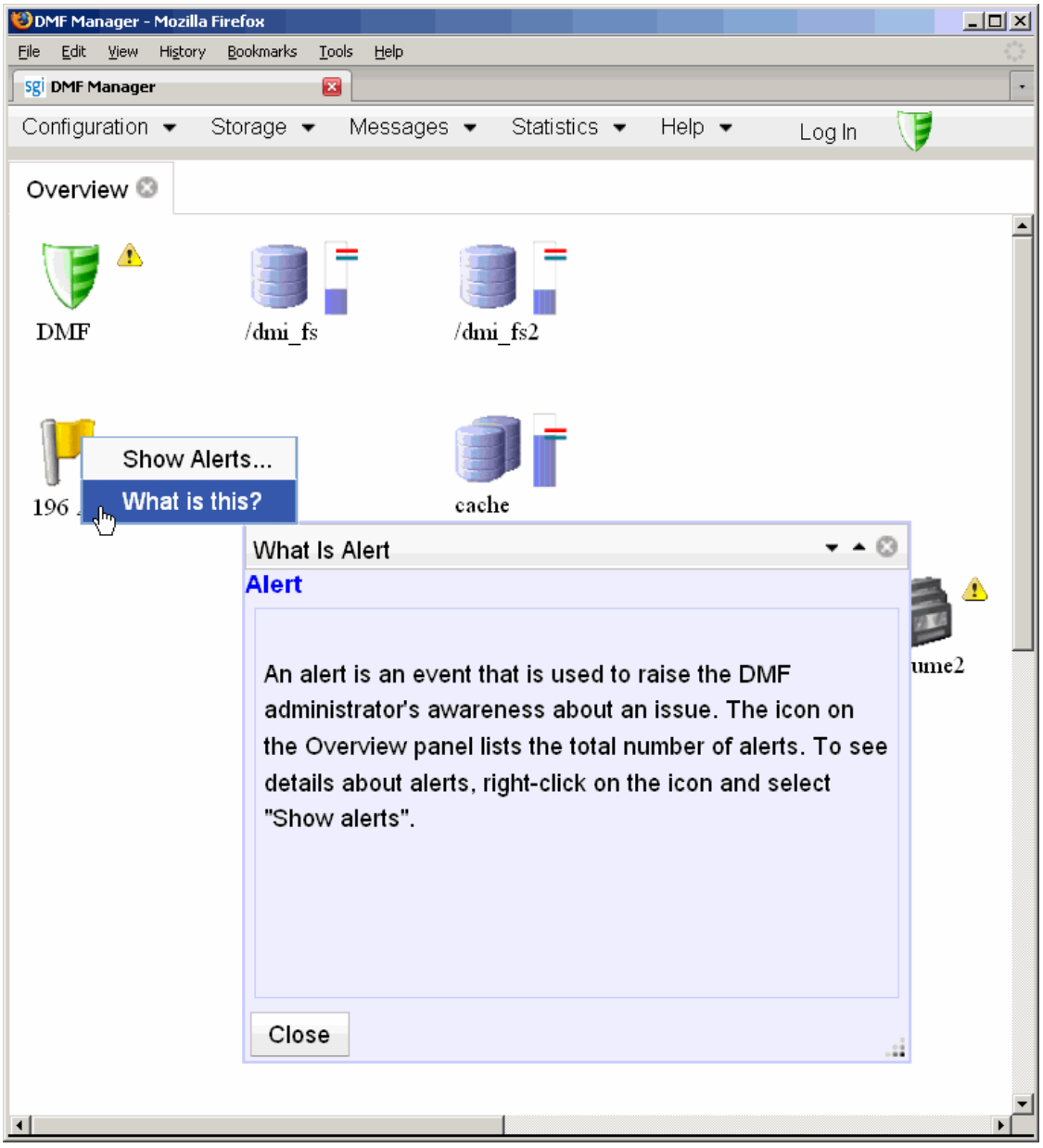

Figure 6-12 DMF Manager Alerts Panel

If you are logged in, you can acknowledge selected alerts or clear all alerts. See "Running Observer Mode or Admin Mode" on page 105.

You can also use the following panel to view daily activity reports (those containing critical log errors show red warning symbols):

Messages
> Reports

Seeing Relationships Among DMF Components

To see the relationships among DMF components, click on a component icon in the **Overview** panel and select its **Show Relationships** menu item. Figure 6-13 shows the relationships for the `/dmi_fs2` filesystem.

To remove the relationship lines, click **Hide Relationships**.

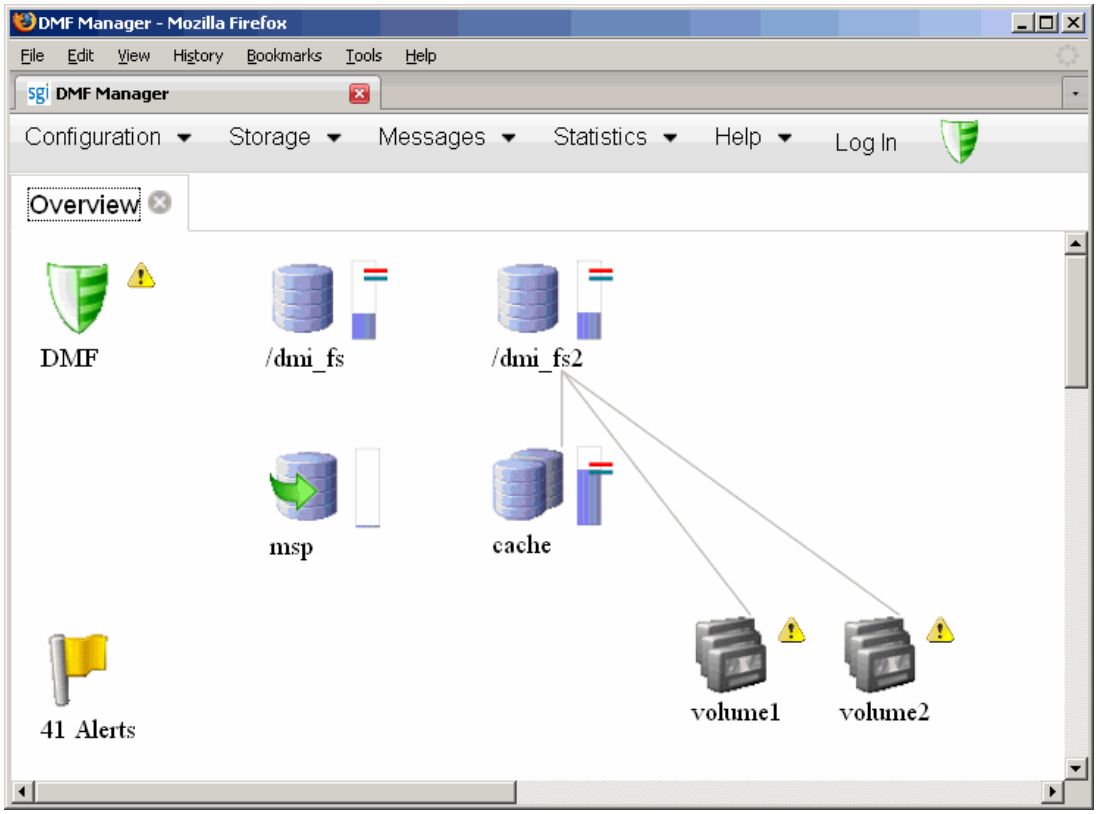

Figure 6-13 Relationships Among DMF Components

Managing Tapes

Figure 6-14 shows an example of the following menu bar selection:

Storage
> Tapes

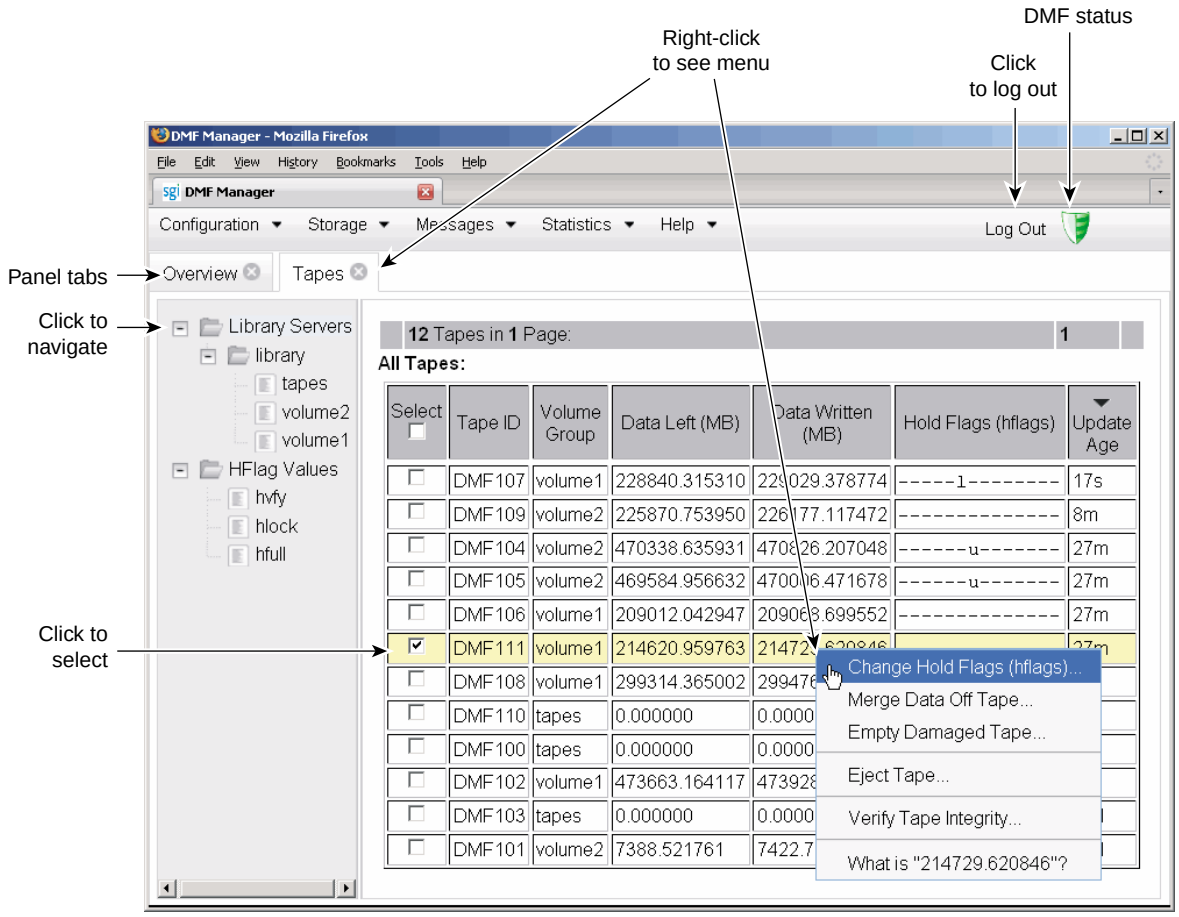

Figure 6-14 DMF Manager Tapes Panel

The following menu selection lets you create a tape group with more-specifically defined characteristics:

Tapes
> Create User-Defined Tape Query ...

If you are logged in as the Admin user, you can save these queries to unique names so you can reuse them later. When logged in, you can also perform the following actions for selected tapes:

- **Change the Hold Flag (hflag)**, shown in Figure 6-15, which sets the hold flag values on individual tapes. In the pop-up menu, click the **On** column to turn a flag on or click the **Off** column to turn a flag off. For more information about the hold flags, click the **Help** button on the pop-up window or select the **What is** menu for the flags displayed in the **Tapes** panel.

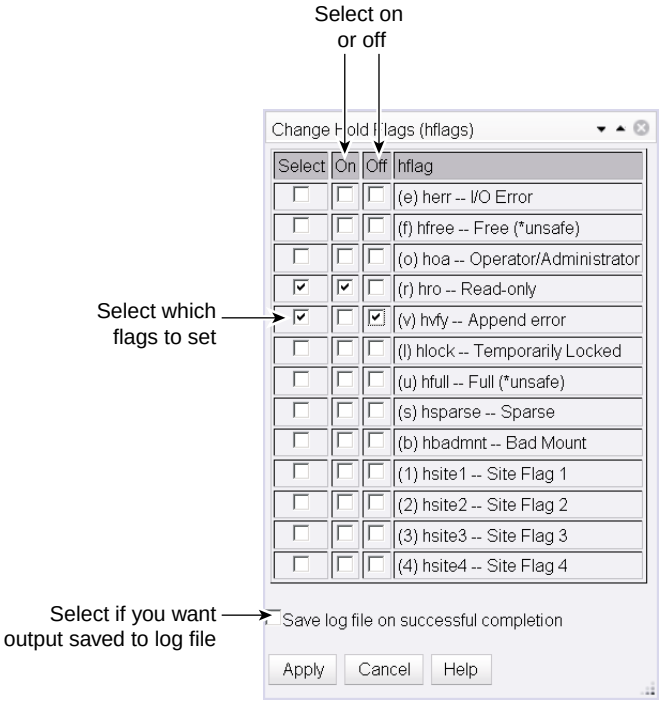

Figure 6-15 Changing Hold Flags in DMF Manager

- **Merge Data Off Tape**, which marks a tape as a candidate to be merged with another tape, thereby recovering space that was lost due to holes in the tape from deleted data (a sparse tape). These operations will be performed when appropriate. This is the preferred way to move data off of tape.
- **Empty Damaged Tape**, which forces data to immediately move to another tape.

Note: Use this as a last resort. You should first try **Merge Data Off Tape**.

- **Eject Tape**, which removes the selected tapes from the library but keeps their tape IDs (volume serial numbers, VSNs) in the VG. (In some cases, this command may cause a door to be unlocked, requiring a human operator to physically extract the cartridge from the library.) This only applies to tapes managed by OpenVault.
- **Verify Tape Integrity**, which runs a verification to make sure that the data on the tape is readable.

See "Running Observer Mode or Admin Mode" on page 105.

You can also use the following menu bar selection to add tapes that are managed by OpenVault:

Tapes
> Add Tapes ...

Managing Libraries

To view the status of libraries, choose the following from the menu bar:

Storage
> Libraries

If you are logged in to DMF Manager, you can enable or disable the selected libraries. See "Running Observer Mode or Admin Mode" on page 105.

Displaying DMF Manager Tasks

A given DMF Manager task may require issuing a set of DMF commands, and these commands may take some time to execute. The following panel displays the long-running DMF Manager tasks that have been issued but not yet acknowledged:

Messages
> DMF Manager Tasks

When logged in, you can choose to acknowledge, checkpoint, kill, or resume each selected DMF command. See "Running Observer Mode or Admin Mode" on page 105.

Monitoring DMF Performance Statistics

The **Statistics** menu provides current and historical views of DMF activity and resources.

This section discusses the following:

Using the Statistics Panels

The **DMF Resources** and **DMF Activity** panels of the **Statistics** menu are divided into the following areas:

- Report tree
- Graphs
- Key

To resize an area, drag the divider lines to the left or right.

Expandable folders in the tree (such as **Requests**) contain reports (such as **Requests Summary**) and subfolders (such as **Filesystem Requests**). Click on the **+** symbol to expand a folder or on the **—** symbol to contract it, or use the **Expand All** and **Collapse All** buttons. Click on a report name to display the associated graphs.

White space within a graph means that nothing happened during that time period, or data was unavailable. This does not indicate an error condition.

DMF Manager distinguishes between *current* and *historic* time:

- Current metrics are either drawn live from the server or are taken from the last few minutes of the metric archives
- Historic metrics are taken exclusively from the metric archives

DMF Manager is able to display this historical information for the following time periods:

- Last hour
- Last day (the previous 24 hours)
- Last month (the previous 30 days)

DMF Activity

Note: To see all of the available statistics via DMF Manager, you must set the `EXPORT_METRICS` configuration parameter to `ON`. This parameter is not configurable while DMF is running; to change the value, DMF must be stopped and restarted. See "`base` Object" on page 152.

The reports in the **DMF Activity** panel show user-generated DMF activity:

- **Requests** reports show the number of requests being worked on
- **Throughput** reports show the rate of data throughput resulting from those requests

---

Note: Values shown are averaged over the previous few minutes, so they are not necessarily integers as would be expected. This process also causes a slight delay in the display, which means that the values of **DMF Activity** reports do not necessarily match the current activity on the system, as seen in the DMF log files.

The following types of requests are reflected in these reports:

- Requests from the user to the DMF daemon. These are presented as an aggregate across the DMF server, and on a per-filesystem basis, using the label of **Filesystems**.
- Requests from the DMF daemon to the subordinate daemons managing the back-end storage, the caches, the volume groups (VGs), and the media-specific processes (MSPs). Technically, caches are a variant of MSP despite their different purpose, hence the description **Non-Cache MSP** in the DMF Manager screens.

Sometimes, there is a 1:1 correspondence between a daemon request and a back-end request by cache, VG, or MSP (such as when a file is being recalled from back-end media back to the primary DMF-managed filesystem), but this is frequently not the

case. For example, migrating a newly created file to back-end media will result in one back-end request per copy, but deleting a migrated file results in a single daemon request but no back-end request at that time. Tape merges may cause a lot of activity within a VG but none at the daemon level.

In the **Summary** reports, the different types of requests are not distinguished from each other. However, if you zoom in (via one of the subfolders, such as **DCM Requests**), the resulting report shows the broad categories as well as by filesystem or by back-end storage group, as appropriate.

Note: Some DMF configuration parameters use multipliers that are powers of 1000, such as KB, MB, and GB. However, the **DMF Activity** and **DMF Resources** panels use multipliers that are powers of 1024, such as kiB, MiB, and GiB. In particular, this means that 1 MiB/s is 2^{20} = 1048576 bytes per second.

Figure 6-16 is an example of a filesystem throughput report. It shows that the primary activity for the `/dmi_fs` filesystem are migrations, with a smaller number of recalls and copies.

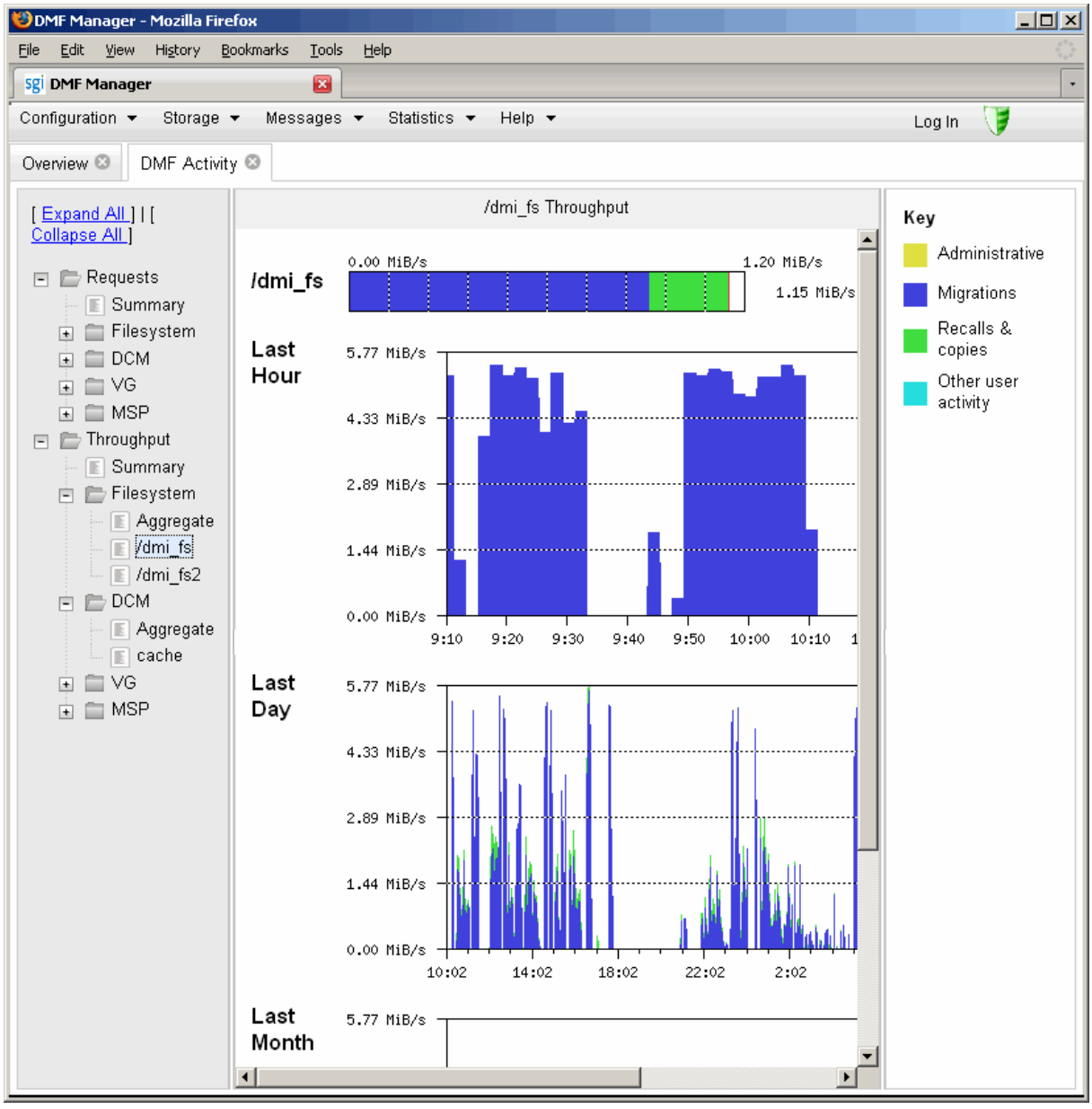

Figure 6-16 DMF Activity

DMF Resources

The **DMF Resources** panel shows how DMF is using its filesystems and hardware, as described in the following sections:

- "Drive Groups Folder" on page 142
- "Volume Groups Folder" on page 143
- "Filesystems Folder" on page 139
- "Disk Caches Folder" on page 145

The reports in this panel are updated at the interval specified in the **DMF Resources Preferences** menu item by those DMF programs that scan the filesystem inodes:

```
dmaudit
dmdadm
dmdskfree
dmfsfree
dmhdelete
dmscanfs
dmselect
```

Note: Some DMF configuration parameters use multipliers that are powers of 1000, such as KB, MB, and GB. However, the **DMF Activity** and **DMF Resources** panels use multipliers that are powers of 1024, such as kiB, MiB, and GiB. In particular, this means that 1 MiB/s is 2^{20} = 1048576 bytes per second.

Filesystems Folder

The reports in the **Filesystems** folder show the proportions of files on DMF-managed filesystems that are migrated and not migrated. The reports also display the amount of offline data related to the filesystems and the over-subscription ratios (which are typically in the range of 10:1 to 1000:1, although they vary considerably from site to site).

Note: Because this is being viewed from the filesystem perspective, the fact that migrated files may have more than one copy on the back-end media is not considered. That is, this is a measure of data that could be on disk but is not at the time, rather than a measure of the amount of back-end media being used.

The data presented in the graph is gathered periodically by DMF. The time at which this information was gathered is displayed at the top of the page. The default configuration is to update this information once daily (at 12:10am).

The **Tape Libraries** report displays the number of slots used by DMF, used by other applications, or empty.

Note: The **Tape Libraries** report is available only if the OpenVault tape subsystem is in use. This report is unavailable if you are using TMF.

Figure 6-17 is an example of a filesystem resource graph. It shows the amount of filesystem space that is free, not migrated, or dual or partial-state for the `/dmi_fs2` filesystem.

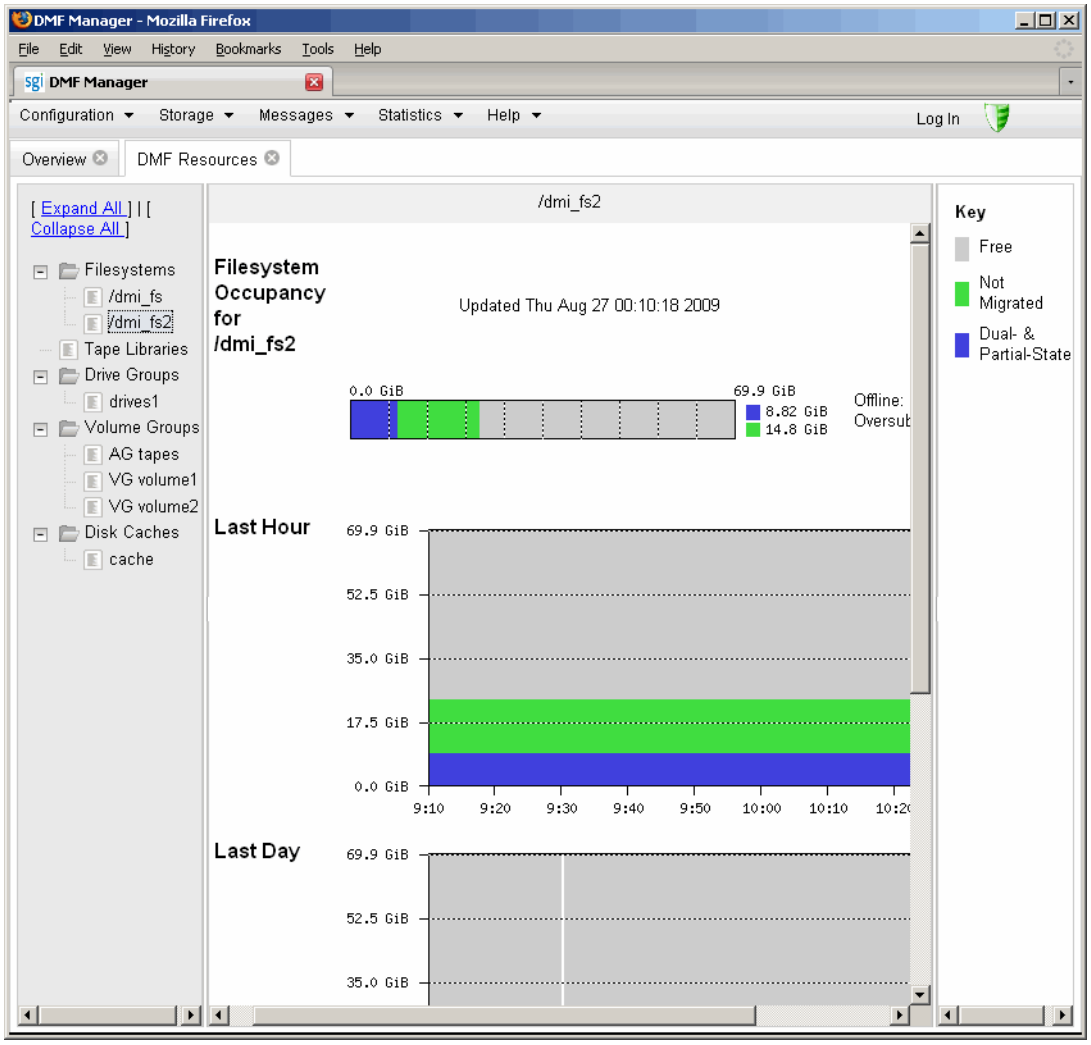

Figure 6-17 Filesystem Resource Graph

Drive Groups Folder

The reports in the **Drive Groups** folder provide information for each tape drive concerning its current state:

- Idle
- Busy
- Unavailable

When the drive is in use, it also shows the following:

- Activity (such as waiting)
- Purpose (such as recall)
- Details of the tape volume (such as volume name)

Note: This information is available only for DMF's tapes. Any other use, such as filesystem backups or direct tape use by users, is not shown.

Figure 6-18 shows information about the `drive1` drive group resource.

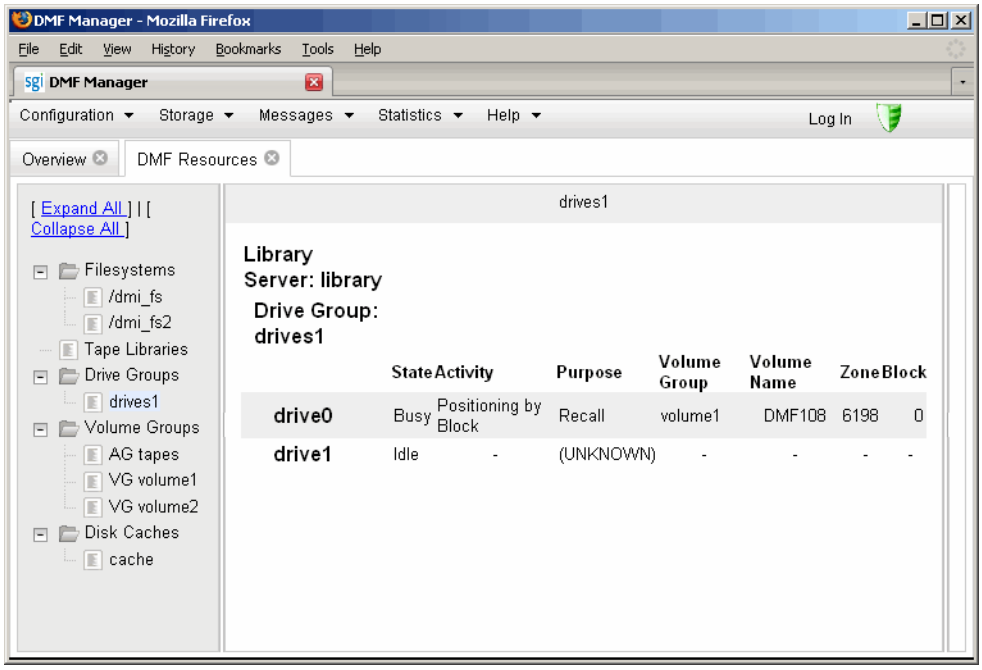

Figure 6-18 Drive Group Resource Information

Volume Groups Folder

The reports in the **Volume Groups** folder show the allocation group (AG) report (if there is an allocation group) and the slot and volume usage and states of each volume group (VG).

Figure 6-19 is an example of a volume group resource graph. It shows that the majority of the `volume2` volume group is either read-only or partial-state.

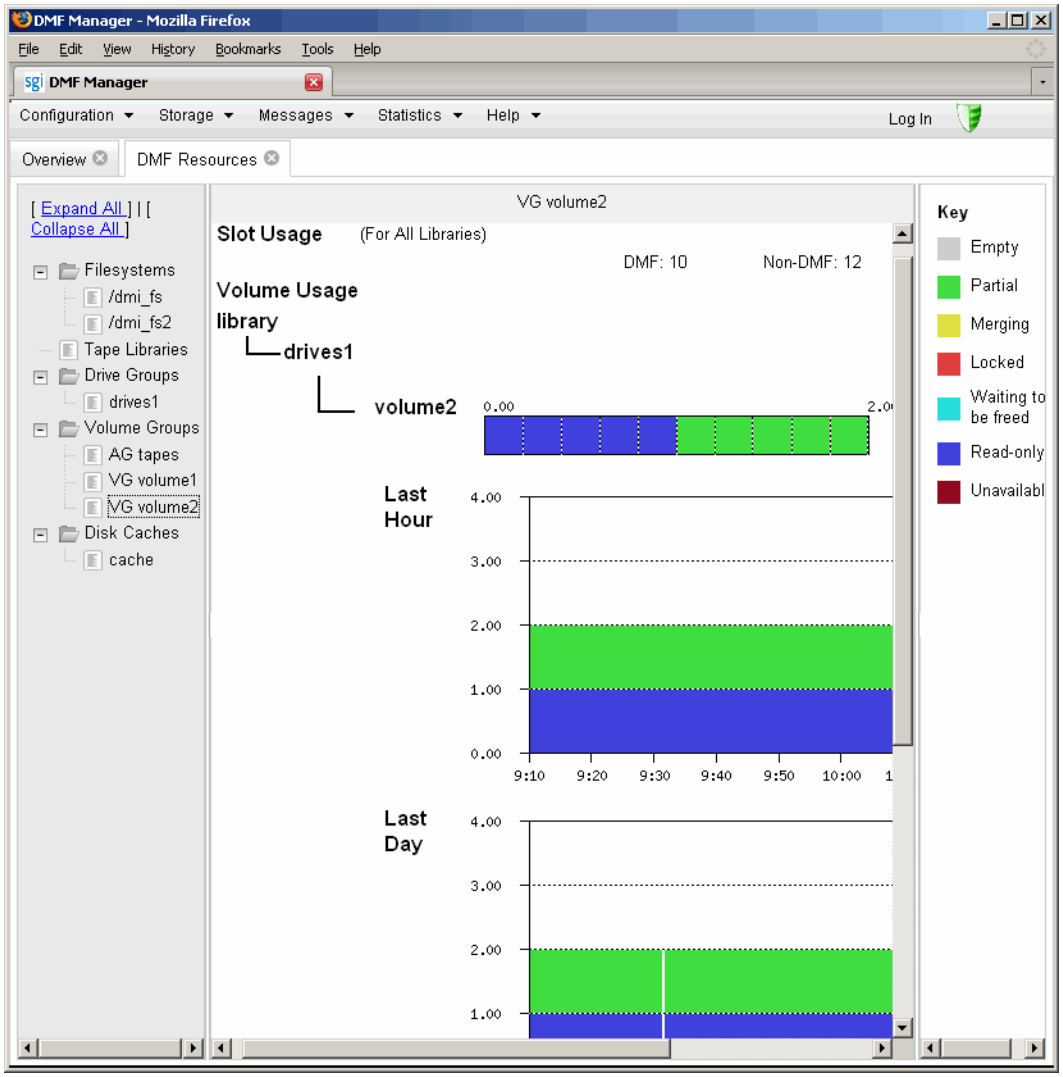

Figure 6-19 Volume Group Resource Graph

Disk Caches Folder

The reports in the **Disk Caches** folder show the cache occupancy:

- *Dual-resident* refers to cache files that have already been copied to back-end tape and can therefore be quickly removed from the cache if it starts filling
- *Not Dual-resident* refers to files that would have tape copies made before they could be removed, which is much slower

---

Note: The disk cache reports have similar issues to filesystem reports with regard to the frequency of updates, as described in "Filesystems Folder" on page 139.

Figure 6-20 is an example of a disk cache resource graph. It shows the amount of cache that the majority of the `cache` disk cache is not dual-resident.

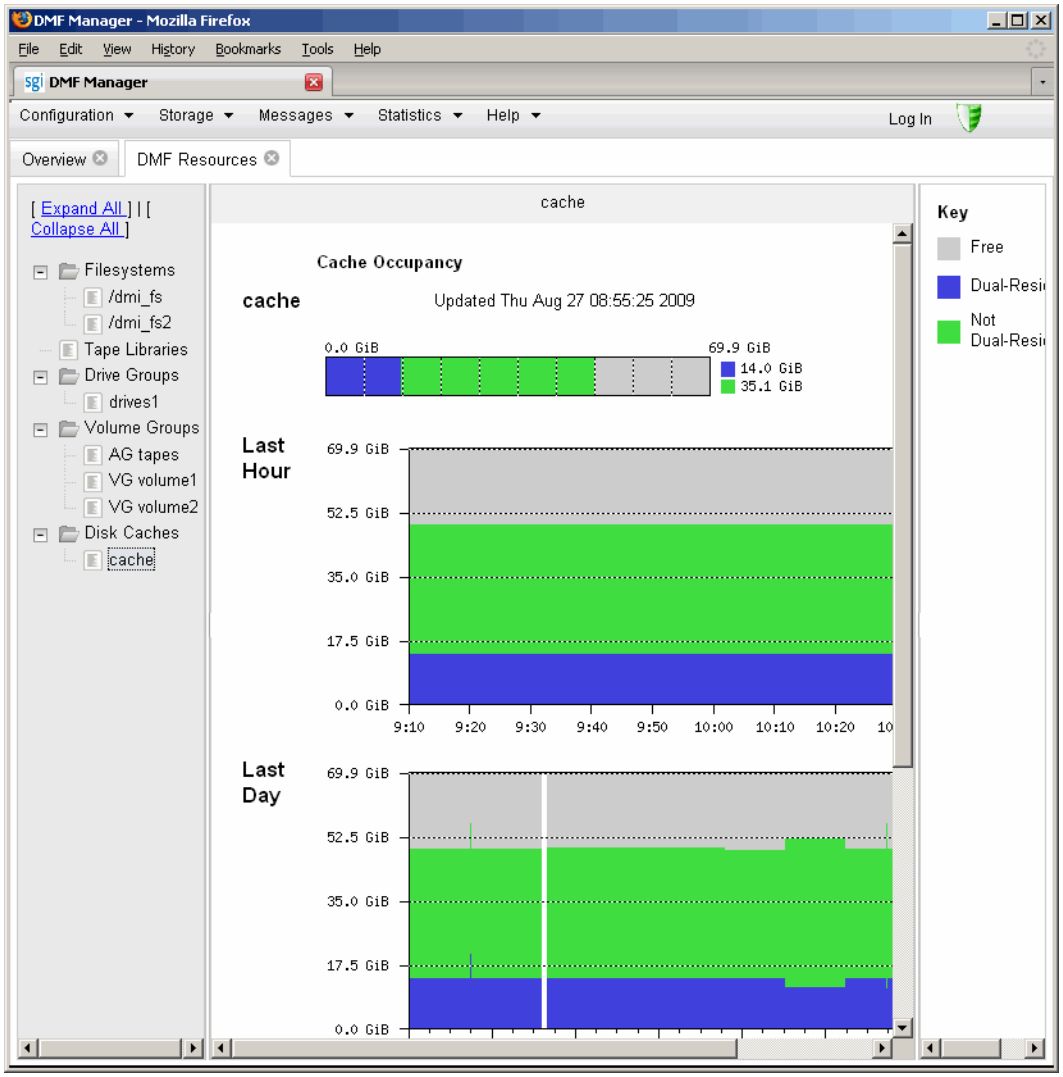

Figure 6-20 Disk Cache Resource Graph

Metrics Collected

Performance Co-Pilot™ continuously gathers performance metrics that are stored in `/var/lib/pcp-storage/archives`. Each month, a data reduction process is performed on the metric gathered for the month. This reduces the size of the archives while retaining a consistent amount of information.

Although the size of metric archives has a bounded maximum, this can still be quite large depending on the configuration of the server and how many clients access it. For example, a server with a large number of filesystems could generate up to 100 Mbytes of archives per day. You should initially allow around 2 GB of space for archive storage and monitor the actual usage for the first few weeks of operation.

DMF Configuration File

This chapter discusses the following:

Configuration Objects Overview

The DMF configuration file (`/etc/dmf/dmf.conf`) defines a set of configuration objects required by DMF. Each object is defined by using a sequence of parameters and definitions;this sequence is called a *stanza.* There is one stanza for each object.

The objects defined are as follows:

- The `base` object defines pathname and file size parameters necessary for DMF operation. See "`base` Object" on page 152.
- The `dmdaemon` object defines parameters necessary for `dmfdaemon`(8) operation. See "`dmdaemon` Object" on page 160.

- The `node` objects define machines that act as data movers when using the DMF Parallel Data Mover Option. There must be one `node` object for each data mover. The data movers may be the DMF server and the DMF parallel data mover nodes. In a high-availability (HA) environment, if the DMF server is to be used as a data mover, there must be `node` objects defined for the primary DMF server and the passive DMF server (DMF client machines are excluded.) See "`node` Object" on page 163.
- The `services` object defines parameters for `dmnode_service` and other DMF services. For DMF configurations using the Parallel Data Mover Option, multiple `services` objects may be defined. For basic DMF configurations, only one `services` object may be defined. (The `services` parameters all have defaults, so a `services` object is only required to change those defaults.) See "`services` Object" on page 167.
- The `taskgroup` objects define parameters necessary for automatic completion of specific maintenance tasks. See "`taskgroup` Object" on page 170.
- The `device` objects define parameters necessary for automatic use of tape devices. (Normally, the backup scripts would refer to a DMF drive group to define parameters necessary for accessing tape drives, but if they are to use drives that are not in use by DMF, you can use a `device` object to define these parameters.) See "`device` Object" on page 186.
- The `filesystem` object defines parameters related to DMF's use of that filesystem. See "`filesystem` Object" on page 187.
- The `policy` objects specify parameters to determine media-specific process (MSP) or volume group (VG) selection, automated space-management policies, and/or file weight calculations in automated space management. See "`policy` Object" on page 193.
- The following objects are related to a library server (LS):
  - The `libraryserver` object defines parameters relating to a tape library for an LS. See "`libraryserver` Object" on page 215.
  - The `drivegroup` object defines parameters relating to a pool of tape devices in a specific LS. See "`drivegroup` Object" on page 217.
  - The `volumegroup` object defines parameters relating to a pool of tape volumes mountable on the drives of a specific drive group that are capable of holding, at most, one copy of user files. See "`volumegroup` Object" on page 224.

  - The `resourcescheduler` object defines parameters relating to scheduling of tape devices in a drive group when requests from VGs exceed the number of devices available. See "`resourcescheduler` Object" on page 230.
  - The `resourcewatcher` object defines parameters relating to the production of files informing the administrator about the status of the LS and its components. See "`resourcewatcher` Object" on page 231.

- The `msp` object defines parameters necessary for an MSP's operation. See:
  - "FTP `msp` Object" on page 245
  - "Disk `msp` Object" on page 250
  - "Disk Cache Manager (DCM) `msp` Object" on page 254

Stanza Format

A stanza has the following general format:

```
define object_name
      TYPE            object_type
      parameter-1     values
 ...
      parameter-n     values
 enddef
```

The *object_name* varies by stanza:

- For the `base` object, it must be `base`
- For `filesystem` objects, it is the mount point
- For `node` objects, it must be the same as the output of the `hostname`(1) command
- For other objects, it is chosen by the administrator

The *object_type* value identifies the type (detailed in the following subsections). The parameters and their values depend on the type of the object.

The configuration file is case-sensitive and lines within it can be indented for readability. The fields can be separated by spaces and/or tabs. Blank lines and all commentary text between a hash character (#) and the end of that line are ignored. Except for comments, any line ending in a back-slash (\) continues onto the next line.

Note: Before placing a new configuration into production, it is important to check it by running dmcheck(8).

For a summary of the parameters discussed in this chapter, see Table 7-3 on page 261. For the most current set of parameters, see the dmf.conf(5) man page.

You can add site-specific parameters to any existing stanza or you can create a new stanza. You should choose parameter and stanza names that will not cause conflict with future SGI DMF parameters and stanzas. See "Start Site-Specific Configuration Parameters and Stanzas with "LOCAL_"" on page 79.

Note: The dmcheck command will point out parameters and stanzas that it does not recognize.

base Object

The base object's parameters define pathnames and file sizes necessary for DMF operation. It is expected that you will modify the pathnames, although those provided will work without modification. All pathnames must be unique.

Warning: Do not change the directory names while DMF is running (changing the directory names can result in data corruption or loss).

Parameter	Description
TYPE	base (required name for this type of object). There is no default.
ADMIN_EMAIL	Specifies the e-mail address to receive output from administrative tasks (see "Automated Maintenance Tasks" on page 89). The mail can include errors, warnings, and output from any configured tasks. You can specify a list of addresses, separated by spaces. When using the Parallel Data Mover Option, data movers (the DMF server node and the parallel data mover nodes) may send email to the ADMIN_EMAIL

addresses. Therefore, choose addresses that can receive email from any data mover in the configuration.

DIRECT_IO_MAXIMUM_SIZE — Specifies the maximum size of I/O requests when using O_DIRECT I/O to read from any primary filesystem or when migrating files down the hierarchy from the *STORE_DIRECTORY* of the disk cache manager (DCM, a disk MSP configured for *n*–tier capability). DIRECT_IO_MAXIMUM_SIZE is ignored for a particular filesystem or DCM store when DIRECT_IO_SIZE is specified in the configuration stanza for that filesystem or DCM. The legal range of values is 262144—18446744073709551615. The default value is 1048576.

EXPORT_METRICS — Enables DMF's use of the common arena for collecting DMF statistics for use by dmstat(8), dmarenadump(8), and other commands. This parameter is not configurable while DMF is running; to change the value, DMF must be stopped and restarted. If set to OFF, some statistics in DMF Manager cannot be displayed. This parameter may be set to OFF, ON, NO, or YES. The default is OFF.

HBA_BANDWIDTH — *(OpenVault only)* Specifies the default I/O bandwidth capacity of an HBA port that is connected to tape drives on the node. The value is in bytes per second. All of the HBA ports connected to tape drives on a node are assumed to have the same bandwidth capacity. If HBA_BANDWIDTH is not specified anywhere, the default is 1024000000000000. For a complete description, see "node Object" on page 163. An HBA_BANDWIDTH value specified in a node object overrides the default value specified in the base object. Also see BANDWIDTH_MULTIPLIER in "drivegroup Object" on page 217.

HOME_DIR — Specifies the base pathname for directories in which files related to the daemon database and LS database reside. The best practice is for *HOME_DIR* to be the mount point of a filesystem that is used only by DMF. In this way, it is much less likely that the filesystem will become full and cause DMF to abort. If you choose to

use *HOME_DIR* for storing HA files or scripts that must be visible on a failover platform, you must use naming conventions that will not likely conflict with present or future DMF files and you must ensure that the files do not cause the filesystem to become full. Performance characteristics of the *HOME_DIR* filesystem will impact DMF database transaction performance and may become a limiting factor in achievable DMF database transaction rates. When using the Parallel Data Mover Option, *HOME_DIR* must either be a CXFS filesystem or be in a CXFS filesystem.

For guidelines about the size of *HOME_DIR*, see "Configure DMF Administrative Filesystems and Directories Appropriately" on page 64.

Note: *HOME_DIR* must be on a separate physical device from *JOURNAL_DIR*.

`JOURNAL_DIR`

Specifies the base pathname for directories in which the journal files for the daemon database and LS database will be written. The best practice is for *JOURNAL_DIR* to be the mount point of a filesystem that is used only by DMF. In this way, it is much less likely that the filesystem will become full and cause DMF to abort. The appropriate size of this filesystem is a function of the expected daily DMF transaction activity and the number of days that journals are kept.

Note: *JOURNAL_DIR* must be on a separate physical device from *HOME_DIR*.

`JOURNAL_SIZE`

Specifies the maximum size (in bytes) of the database journal file before DMF closes it and starts a new file.

`LICENSE_FILE`

Specifies the full pathname of the file containing the licenses used by DMF. The default is `/etc/lk/keys.dat`. (There is no need to use this parameter if the default is being used.)

NODE_BANDWIDTH

(OpenVault only) Specifies the default I/O bandwidth capacity of the node. If NODE_BANDWIDTH is not specified anywhere, the default is 1024000000000000. For a complete description, see "node Object" on page 163. A NODE_BANDWIDTH value specified in a node object overrides the default value specified in the base object. Also see BANDWIDTH_MULTIPLIER in "drivegroup Object" on page 217.

OV_KEY_FILE

(OpenVault only) Specifies the file containing the OpenVault security keys used by DMF. It is usually located in *HOME_DIR* and called ovkeys. There is no default. When using the Parallel Data Mover Option, this file must be visible to the DMF server and all parallel data mover nodes, therefore it must be in a CXFS filesystem.

OV_SERVER

(OpenVault only) Specifies the hostname of the OpenVault server. This should only be used if it is not the same as the DMF server.

Note: More configuration steps are necessary to configure DMF to use OpenVault; see "OpenVault and LS Drive Groups" on page 235.

SERVER_NAME

Specifies the hostname of the machine on which the DMF server is running. In an HA configuration, SERVER_NAME must be the HA virtual hostname rather than the output of the hostname (1) command. This parameter is only required for HA configurations or configurations using the DMF Parallel Data Mover Option.

Note: If you change this parameter, you must copy the dmf.conf file manually to each parallel data mover node and then restart the DMF services. Do not change this parameter while DMF is running.

SPOOL_DIR

Specifies the base pathname for directories in which DMF log files are kept. The best practice is for

SPOOL_DIR to be the mount point of a filesystem that is used only by DMF. In this way, it is much less likely that the filesystem will become full and cause DMF to abort. The appropriate size of this filesystem is a function of the expected daily DMF transaction activity, the MESSAGE_LEVEL parameter setting, and the number of days that logs are kept. When using the Parallel Data Mover Option, *SPOOL_DIR* must either be a CXFS filesystem or be in a CXFS filesystem.

TMP_DIR — Specifies the base pathname for directories in which DMF puts temporary files for its own internal use. It is also used by DMF commands and scripts and is the directory used by default by the library server (LS) for caching files if the CACHE_DIR parameter is not defined. The best practice is for *TMP_DIR* to be the mount point of a filesystem that is used only by DMF. *TMP_DIR* filesystem performance will impact the performance of many of the internal DMF administrative tasks, particularly tasks that involve the need to sort DMF databases. When using the Parallel Data Mover Option, TMP_DIR must either be a CXFS filesystem or be in a CXFS filesystem.

Note: Many DMF operations that do analysis on the DMF database contents use *TMP_DIR* as their work directory. Because most of these involve large buffered I/O, SGI recommends that you configure *TMP_DIR* on a fast disk, with bandwidth at the RAID level.

When an MSP, LS, daemon, or configuration file object (such as the taskgroup object named dump_tasks in Example 7-10 on page 184) obtains a path such as HOME_DIR from the configuration file, the actual path used is the value of HOME_DIR plus the MSP/LS/daemon/object name appended as a subdirectory. For example, if the value of HOME_DIR was set to /dmf/home in the configuration file, and the taskgroup object named dump_tasks used a value of HOME_DIR/tapes for the DUMP_TAPES parameter, then the actual path for DUMP_TAPES would resolve to /dmf/home/dump_tasks/tapes.

Note: Do not use automated space management to manage the *HOME_DIR*, *SPOOL_DIR*, or *JOURNAL_DIR* directories, because DMF daemon processes will deadlock if files that they are actively using within these directories are migrated. The `dmcheck`(8) command reports an error if any of the *HOME_DIR*, *SPOOL_DIR*, or *JOURNAL_DIR* directories are also configured as DMF-managed filesystems. You should configure a `taskgroup` object for daemon tasks to manage old log files and journal files in these directories. See "`taskgroup` Object" on page 170 for more information.

Example 7-1 `base` object for Basic DMF

```
define  base
        TYPE            base
        ADMIN_EMAIL     root@dmfserver
        HOME_DIR        /dmf/home
        TMP_DIR         /dmf/tmp
        SPOOL_DIR       /dmf/spool
        JOURNAL_DIR     /dmf/journals
        JOURNAL_SIZE    10m
        OV_KEY_FILE     /dmf/home/ovkeys
enddef
```

In the above example:

- A new journal file will be created after the present file reaches 10 million bytes
- The `OV_KEY_FILE` parameter is necessary if OpenVault is used as the mounting service
- The hostname of the OpenVault server is the same as the DMF server, so `OV_SERVER` is not specified

Example 7-2 `base` object for DMF Using the Parallel Data Mover Option

```
define  base
        TYPE            base
        SERVER_NAME     server1
        ADMIN_EMAIL     root@dmfserver
        HOME_DIR        /dmf/home
        TMP_DIR         /dmf/tmp
        SPOOL_DIR       /dmf/spool
        JOURNAL_DIR     /dmf/journals
        JOURNAL_SIZE    10m
        OV_KEY_FILE     /dmf/home/ovkeys
enddef
```

In the above example:

- The `SERVER_NAME` parameter is required when using the Parallel Data Mover Option. The hostname of the machine that is running the DMF and OpenVault servers in this case is `server1` (so `OV_SERVER` is not specified).

- /dmf/tmp must either be a CXFS filesystem or be in a CXFS filesystem when using the Parallel Data Mover Option.
- The /dmf/spool directory must either be a CXFS filesystem or be in a CXFS filesystem when using the Parallel Data Mover Option.
- A new journal file will be created after the present file reaches 10 million bytes.
- OpenVault must be configured as the mounting service for tape drives that are used by parallel data mover nodes. The /dmf/home/ovkeys file must be visible to the DMF server node and all parallel data mover nodes, therefore it must be in a CXFS filesystem.

Example 7-3 base object for DMF Using the Parallel Data Mover Option in an HA Cluster

```
define  base
        TYPE            base
        SERVER_NAME     virtual-server
        ADMIN_EMAIL     root@dmfserver
        HOME_DIR        /dmf/home
        TMP_DIR         /dmf/tmp
        SPOOL_DIR       /dmf/spool
        JOURNAL_DIR     /dmf/journals
        JOURNAL_SIZE    10m
        OV_KEY_FILE     /dmf/home/ovkeys
enddef
```

In the above example:

- The SERVER_NAME parameter is required when using the Parallel Data Mover Option. Because this configuration is using HA, it must be set to the HA virtual hostname (in this case virtual-server), which corresponds to the HA_VIRTUAL_HOSTNAME parameter in the node objects for the DMF server.
- /dmf/tmp must either be a CXFS filesystem or be in a CXFS filesystem when using the Parallel Data Mover Option.
- The /dmf/spool directory must either be or be in a CXFS filesystem when using the Parallel Data Mover Option.
- OpenVault must be configured as the mounting service for tape drives that are used by parallel data mover nodes. The /dmf/home/ovkeys file must be visible to the DMF server and all parallel data mover nodes, therefore it must be in a CXFS filesystem.

- The hostname of the OpenVault server is the same as the DMF server, so OV_SERVER is not specified.

dmdaemon Object

The dmdaemon object defines configuration parameters necessary for the DMF daemon operation. It is expected that you will modify the values for the pathnames and MSP names.

Parameter **Description**

TYPE

dmdaemon (required name for this type of object). There is no default.

Note: This cannot be specified as dmfdaemon. It must be dmdaemon.

EXPORT_QUEUE

Instructs the daemon to export details of its internal request queue to *SPOOL_DIR*/daemon_exports every two minutes, for use by dmstat(8) and other utilities. On a busy system, the responsiveness of the daemon may be improved by disabling this feature. This parameter may be set to OFF, ON, NO, or YES. The default is OFF.

MESSAGE_LEVEL

Specifies the highest message level number that will be written to the daemon log. It must be an integer in the range 0--6; the higher the number, the more messages written to the log file. The default is 2. For more information on message levels, see Chapter 9, "Message Logs" on page 277.

MIGRATION_LEVEL

Sets the highest level of migration service allowed on all DMF filesystems (you can configure a lower service level for a specific filesystem). The value can be:

- none (no migration)
- user (requests from dmput or dmmigrate only)

- `auto` (automated space management)

The default is `auto`.

See "`policy` Object" on page 193 for information about configuring automated space management.

`MOVE_FS`

Specifies the scratch filesystem that is used by `dmmove`(8)to move files between MSPs or VGs. *MOVE_FS* must be the root of a DMAPI filesystem (mounted with `dmi,mtpt=`*/MOVE_FS*). The size of *MOVE_FS* is a function of expected `dmmove` activity. *MOVE_FS* must be mounted when a `dmmove` command is executed. The best practice when using *MOVE_FS* is for it to be dedicated to the `dmmove` function. (The `dmmove` command calculates the available space in *MOVE_FS* when selecting move candidates; if other processes are allocating space in *MOVE_FS*, those calculations can become inaccurate, causing errors.) When using the DMF Parallel Data Mover Option, if `MOVE_FS` is specified, it must be a CXFS filesystem.

`LS_NAMES`, `MSP_NAMES`

Names the LSs or MSPs used by the DMF daemon. You must specify either `LS_NAMES` or `MSP_NAMES`, but not both (however, the value of either parameter can be a mixture of both forms). There is no default.

The order of the values specified for this parameter is integral to the determination of the MSP/VG from which the DMF daemon attempts to recall an offline file. If the offline file has more than one copy, DMF uses a specific order when it attempts to recall the file. It searches for a good copy of the offline file in MSP or LS order, from the `dmdaemon` object's `MSP_NAMES` or `LS_NAMES` parameter. If one of those names refers to an LS, it searches for the copy in drive group order, from the `libraryserver` object's `DRIVE_GROUPS` parameter. It then searches for the copy in VG order from the `drivegroup` object's `VOLUME_GROUPS` parameter.

Note: Do not change these parameters while DMF is running.

PARTIAL_STATE_FILES

Enables or disables the DMF daemon's ability to produce partial-state files. The possible values are:

- ON or YES, which means that the daemon will correctly process put and get requests that would result in a partial-state file. The default is on.
- OFF or NO, which means that all put and get requests that require a change to the online status of the file will result in a file that is completely online or offline. That is, any put request that makes any part of the file offline will result in the entire file being made offline. Any get request that would result in any part of the file being brought back online will result in the entire file being brought back online.

RECALL_NOTIFICATION_RATE

Specifies the approximate rate, in seconds, at which regions of a file being recalled are put online. This allows for access to part of a file before the entire file is recalled. The default is 30 seconds. Specify a value of 0 if you want the user process to be blocked until the entire recall is complete. The optimum setting of this parameter is dependent on many factors and must be determined by trial and error. The actual rate at which regions being recalled are put online may vary from the value of RECALL_NOTIFICATION_RATE.

TASK_GROUPS

Names the task groups that contain tasks the daemon should run. They are configured as objects of TYPE taskgroup. There is no default. For more information, see "taskgroup Object" on page 170. SGI recommends that you use the task groups specified in the sample configuration file, changing the parameters as necessary for your site.

Example 7-4 dmdaemon object

```
define  daemon
        TYPE                    dmdaemon
        MOVE_FS                 /dmmove_dir
        LS_NAMES                lib1 ftp2
        TASK_GROUPS             daemon_tasks dump_tasks
enddef
```

In the above example:

- The name of the dmdaemon object is daemon.
- The dmmove command will use the /dmmove_dir filesystem (which can accept temporary files) to move files between VGs and/or MSPs.
- The names of the LSs are lib1 and ftp2.
- The daemon will run the tasks specified by the daemon_tasks and dump_tasks objects (see Example 7-9 on page 179 and Example 7-10 on page 184). In the example, daemon_tasks defines the tasks such as scanning and managing log files and journal files. The dump_tasks object defines tasks that back up DMF-managed filesystems.
- The MIGRATION_LEVEL level is not explicitly set, so the default of auto is used.

node Object

Note: The node object is only for DMF configurations using the Parallel Data Mover Option. Basic DMF configurations do not use the node object.

In a configuration using the DMF Parallel Data Mover Option, there must be one node object for each data mover. The data movers may be the DMF server and the DMF parallel data mover nodes. In a high-availability (HA) environment, if the DMF server is to be used as a data mover, there must be node objects defined for the primary DMF server and the passive DMF server (DMF client machines are excluded.) The name of a node object must match the name returned by hostname(1) on the machine.

Parameter	Description
TYPE	node (required name for this type of object). There is no default.
HA_VIRTUAL_HOSTNAME	Specifies the virtual hostname. In an HA environment, potential DMF server nodes must set this parameter to the same virtual hostname used for SERVER_NAME in the base object. Parallel data mover nodes should not define this parameter.
HBA_BANDWIDTH	*(OpenVault only)* Specifies the I/O bandwidth capacity of an HBA port that is connected to tape drives on the node. The value is in bytes per second. All of the HBA ports connected to tape drives on a node are assumed to have the same bandwidth capacity. When the mounting service is OpenVault, the LS uses this value when determining which tape drives to use. The maximum value is 1024000000000000. The minimum value is 0, which means that the HBA will not be used. The default value is 1024000000000000 or else the value assigned to this parameter in the base object. An HBA_BANDWIDTH value specified in a node object overrides the default value specified in the base object; see "base Object" on page 152. Also see BANDWIDTH_MULTIPLIER in "drivegroup Object" on page 217.
NODE_BANDWIDTH	*(OpenVault only)* Specifies the I/O bandwidth capacity of the node, in bytes per second. When the mounting service is OpenVault, the LS uses this value to calculate how many tape drives it can simultaneously use on a node. The maximum value is 1024000000000000. The minimum value is 0, which means that the node will not be used. The default value is 1024000000000000 or else the value assigned to this parameter in the base object. A NODE_BANDWIDTH value specified in a node object overrides the default value specified in the base object; see "base Object" on page 152. Also see BANDWIDTH_MULTIPLIER in "drivegroup Object" on page 217.
SERVICES	Specifies the name of the services object used to configure DMF services on this node. Multiple nodes

may refer to the same `services` object. For node-specific configuration, each node can refer to a different `services` object. If no `SERVICES` parameter is defined, the default values for the `services` object parameters are used.

Example 7-5 `node` Objects for DMF Using the Parallel Data Mover Option

```
define server1
        TYPE                    node
        SERVICES                server1_services
enddef

define pdm1
        TYPE                    node
        SERVICES                pdm1_services
enddef
```

In the above example:

- There are two data movers: the DMF server `server1` and the parallel data mover node `pdm1`.
- The DMF services on the `server1` node use the parameters defined in the `server1_services` object. The DMF services on the `pdm1` node use the parameters defined in the `pdm1_services` object.

Example 7-6 node Objects for DMF Using the Parallel Data Mover Option in an HA Cluster

```
define server1
        TYPE                    node
        HA_VIRTUAL_HOSTNAME     virtual-server
        SERVICES                dmfserver_services
enddef

define server2
        TYPE                    node
        HA_VIRTUAL_HOSTNAME     virtual-server
        SERVICES                dmfserver_services
enddef

define pdm1
        TYPE                    node
        SERVICES                pdm1_services
enddef
```

In the above example:

- The following nodes are data movers:
  - Either the primary DMF server `server1` or the passive DMF server `server2`
  - The parallel data mover node `pdm1`

  Note: At any given time, only one of the potential DMF server nodes (either `server1` or `server2`) may provide data mover functionality.

- The virtual hostname in the HA cluster is `virtual-server`. This is required in an HA cluster.
- The potential DMF server nodes provide the tasks that are described by the `dmfserver_services` object. The parallel data mover node provides the DMF services described by the `pdm1_services` object.

services Object

The services object defines parameters for dmnode_service and other DMF services. When using the Parallel Data Mover Option, multiple services objects may be defined. For basic DMF configurations, exactly one services object may be defined. (The services parameters all have defaults, so you only need to define a services object if you want to change those defaults.)

Parameter	Description
TYPE	services (required name for this type of object). There is no default.
MESSAGE_LEVEL	Specifies the highest message level number that will be written to the service logs. It must be an integer in the range 0--6; the higher the number, the more messages written to the log files. The default is 2. For more information on message levels, see Chapter 9, "Message Logs" on page 277.
NODE_ANNOUNCE_RATE	Specifies the rate in seconds at which the DMF server or parallel data mover node will contact the dmnode_service on the DMF server to announce its presence. This also determines the rate at which configuration changes are propagated to any DMF parallel data mover nodes. This value should be less than the value of NODE_TIMEOUT. The default is 20 seconds.
NODE_TIMEOUT	Specifies the number of seconds after which the data mover functionality on the DMF server or on a parallel data mover node will be considered inactive if it has not contacted the dmnode_service on the DMF server. This value should be larger than the value of NODE_ANNOUNCE_RATE. The default is 30 seconds.
SERVICES_PORT	Specifies the port number on which DMF starts a locator service, which DMF uses to locate other DMF services. It must be an integer in the range 1 through 65535. The default is 44333.

Note: If you change this parameter, you must copy the `dmf.conf` file manually to each parallel data mover node and then restart the DMF services.

Do not change this parameter while DMF is running.

`TASK_GROUPS` Names the task groups that contain tasks that should be run. They are configured as `taskgroup` objects. Unlike task groups specified for other objects, task groups defined by the `TASK_GROUPS` parameter for the `services` object will run on the DMF server and every parallel data mover node (rather than on just the DMF server). If you specify this parameter, you must specify the scripts to be run. For more information, see "`taskgroup` Object" on page 170.

SGI recommends that you use the task groups specified in the sample configuration file, changing the parameters as necessary for your site.

Example 7-7 `services` object for DMF Using the Parallel Data Mover Option

```
define server1_services
        TYPE                    services
        MESSAGE_LEVEL           2
        TASK_GROUPS             node_tasks
enddef

define pdm1_services
        TYPE                    services
        MESSAGE_LEVEL           4
        SERVICES_PORT           1111
        TASK_GROUPS             node_tasks
enddef
```

In the above example:

- Two services are defined:
  - `server1_services` (which applies to `server1` (as shown in Example 7-5 on page 165)
  - `pdm1_services` (which applies to `pdm1`, as also shown in Example 7-5)
- The `server1` services will log fewer messages than the `pdm1` services.
- The `pdm1` services use locator port `1111`. The `server1` services will use the default port.
- Both services use the tasks described by the `node_tasks` object.

Example 7-8 `services` object for DMF Using the Parallel Data Mover Option in an HA Cluster

```
define dmfserver_services
        TYPE                    services
        MESSAGE_LEVEL           2
        TASK_GROUPS             servernode_tasks
enddef

define pdm1_services
        TYPE                    services
        MESSAGE_LEVEL           4
        SERVICES_PORT           1111
        TASK_GROUPS             pdmnode_tasks
enddef
```

In the above example:

- Two services are defined:
  - `dmfserver_services`, which applies to `server1` and `server2` (as shown in Example 7-6 on page 166)
  - `pdm1_services`, which applies to `pdm1` (as shown in Example 7-6)
- The `dmfserver_services` services will log fewer messages than the `pdm1` services.
- The `pdm1` services use locator port `1111`. The `dmfserver_services` services will use the default port.
- The active DMF server (either `server1` or `server2`) will run the tasks defined by the `servernode_tasks` object.
- The DMF parallel data mover node `pdm1` will run the tasks defined by the `pdmnode_tasks` object.

`taskgroup` Object

This section discusses the following:

Overview of the `taskgroup` Object

You can configure `taskgroup` objects to manage how periodic maintenance tasks are performed. Many of these maintenance tasks are performed by the DMF daemon, while others are performed by LSs, drive groups, services, or DCMs. Table 7-1 summarizes the tasks.

Note: For more information about third party backups see, see "Using DMF-aware Third-Party Backup Packages" on page 354.

You can configure when each task should run. For some of the tasks, you must provide more information (such as destinations or retention times for output).

Table 7-1 Automated Maintenance Task Summary

Referencing Object Type	Task	Purpose	Parameters
dmdaemon	run_audit.sh	Audit databases	
	run_copy_databases.sh	Back up DMF databases	DATABASE_COPIES
	run_daily_drive_report.sh	Create a report about tape drives that have indicated they need cleaning	DRIVETAB
	run_daily_report.sh[1]	Create a report including information on managed filesystems (if run_filesystem_scan.sh has been run recently) and DCMs, and all LSs	
	run_daily_tsreport.sh	Create a report containing the output of the tsreport command, which reports tape drive alerts, errors, and statistics	DRIVETAB TSREPORT_OPTIONS
	run_filesystem_scan.sh	Run dmscanfs(8) on filesystems to collect file information for subsequent use by other scripts and programs	SCAN_FAST SCAN_FILESYSTEMS SCAN_FOR_DMSTAT SCAN_OUTPUT SCAN_PARAMS SCAN_PARALLEL

1 The run_compact_tape_report.sh and run_tape_report.sh tasks have been superseded by the run_daily_report.sh task.

Referencing Object Type	Task	Purpose	Parameters
	run_full_dump.sh	Full backup of filesystems (For restores, see dmxfsrestore(8))	DUMP_DEVICE DUMP_FILE_SYSTEMS DUMP_FLUSH_DCM_FIRST DUMP_INVENTORY_COPY DUMP_MAX_FILESPACE DUMP_MIGRATE_FIRST DUMP_RETENTION DUMP_TAPES DUMP_VSNS_USED DUMP_XFSDUMP_PARAMS
	run_hard_deletes.sh	Hard-delete files	Uses DUMP_RETENTION
	run_partial_dump.sh	Partial backup of filesystems	Uses parameters set for run_full_dump.sh
	run_remove_journals.sh	Remove old journal files	JOURNAL_RETENTION
	run_remove_logs.sh	Remove old log files	LOG_RETENTION
	run_scan_logs.sh	Scan recent log files for errors	
drivegroup	run_merge_mgr.sh	Merge sparse tapes	DATA_LIMIT THRESHOLD VOLUME_LIMIT
libraryserver	run_merge_stop.sh	Stop volume merges	
	run_tape_merge.sh	Merge sparse tapes	DATA_LIMIT THRESHOLD VOLUME_LIMIT
DCM msp	run_dcm_admin.sh	Routine disk cache manager (DCM) administration	
services	run_remove_logs.sh	Remove old log files	LOG_RETENTION

taskgroup Parameters

The taskgroup object parameters are as follows:

TYPE	taskgroup (required name for this type of object). There is no default.
DATABASE_COPIES	Specifies one or more directories into which the run_copy_databases.sh task will place a copy of the DMF databases. The run_copy_databases.sh task copies a snapshot of the current DMF databases to the directory with the oldest copy. If you specify multiple directories, you should spread the directories among multiple disk devices in order to minimize the chance of losing all the copies of the databases. There is no default.
DRIVETAB	Provides the name of a file that is used with the tsreport --drivetab option, which causes the run_daily_drive_report and run_daily_tsreport output to contain the more-readable drive name instead of the device name. By default, the device name is reported.
DUMP_DATABASE_COPY	Specifies the path to a directory where a snapshot of the DMF databases will be placed when do_predump.sh is run. The third-party backup application should be configured to backup this directory. If not specified, a snapshot will not be taken. (Third–party backup applications only).
DUMP_DEVICE	Specifies the name of the drive group in the configuration file that defines how to mount the tapes that the dump tasks will use.
DUMP_FILE_SYSTEMS	Specifies one or more filesystems to dump. If not specified, the tasks will dump all the DMF-managed user filesystems configured in the configuration file. Use this parameter only if your site needs different dump policies (such as different dump times) for different filesystems or wishes to back up filesystems that are not managed by DMF. It is safest not to specify a value for this parameter and therefore dump all filesystems configured for management by DMF.

DUMP_FLUSH_DCM_FIRST	If set to YES, specifies that the dmmigrate command is run before the dumps are done to ensure that all non-dual-resident files in the DCM caches are migrated to tape. If DUMP_MIGRATE_FIRST is also enabled, that is processed first. This parameter may be set to OFF, ON, NO or YES. The default is OFF.
DUMP_INVENTORY_COPY	Specifies the pathnames of one or more directories into which are copied the XFS inventory files for the backed-up filesystems. If you specify multiple directories, spreading the directories among multiple disk devices minimizes the chance of losing all copies of the inventory. The dump scripts choose the directory with the oldest inventory copy and copy the current one to it.
DUMP_MAX_FILESPACE	Specifies a maximum disk space used for files to be dumped, which may be larger or smaller than the length of the file. Regular files using more than this space are silently left out of the dump. This limit is not applied to offline or dual-state files. This value applies to all filesystems being dumped except for the backup of the DMF databases. The value, which is in bytes, may have a suffix that indicates the multiplier k (1000) m (1000000) g (1000000000) If this parameter is not provided, there is no limit.
DUMP_MIGRATE_FIRST	If set to YES, specifies that the dmmigrate command is run before the dumps are done to ensure that all migratable files in the DMF-managed user filesystems are migrated, thus reducing the number of tapes needed for the dump and making it run much faster. This parameter may be set to OFF, ON, NO or YES. The default is OFF.
DUMP_RETENTION	Specifies how long the backups of the filesystem will be kept before the tapes are reused. Valid values are a number followed by one of m[inutes], h[ours], d[ays] or w[eeks].

DUMP_TAPES

Specifies the path of a file that contains tape volume serial numbers (VSNs), one per line, for the dump tasks to use. Any text in that file after a # character is considered to be a comment.

Note: When an MSP, LS, daemon, or configuration file object (such as the `taskgroup` object example named `dump_tasks`) obtains a path such as `HOME_DIR` from the configuration file, the actual path used is the value of `HOME_DIR` plus the MSP/LS/daemon/object name appended as a subdirectory. In the above example, if the value of `HOME_DIR` was set to `/dmf/home` in the configuration file, then the actual path for `DUMP_TAPES` would resolve to `/dmf/home/dump_tasks/tapes`.

DUMP_VSNS_USED

Specifies a file in which the VSNs of tapes that are used are written. If this parameter is not provided, `/dev/null` is used, effectively disabling this feature.

DUMP_XFSDUMP_PARAMS

Passes parameters to the `xfsdump` program. The value is not checked for validity, so this parameter should be used with care. Make sure that there are no conflicts with the `xfsdump` parameters generated by the DMF scripts.

JOURNAL_RETENTION

Specifies the age at which the `run_remove_journals.sh` script will remove journals. Valid values are an integer followed by `m[inutes]`, `h[ours]`, `d[ays]`, or `w[eeks]`

LOG_RETENTION

Specifies the age at which the `run_remove_logs.sh` script will remove logs. Valid values are an integer followed by `m[inutes]`, `h[ours]`, `d[ays]`, or `w[eeks]`.

RUN_TASK

Specifies the scripts to be run. All of the `RUN_TASK` parameters have the same syntax in the configuration file:

`RUN_TASK $ADMINDIR/`*`script_name time_expression`*

DMF will equate $ADMINDIR to the appropriate directory, which is usually /usr/lib/dmf. When the *script_name* task is run, it is given the name of the object that requested the task as the first parameter and the name of the task group as the second parameter. The task itself may use the dmconfig(8) command to obtain further parameters from either of these objects. Some of the tasks defined by the RUN_TASK parameters require more information.

The *time_expression* defines when a task should be done. It is a schedule expression that has the following form:

[every *n period*] [at *hh*:*mm*[:*ss*] ...] [on *day* ...]

period is one of minute[s], hour[s], day[s], week[s], or month[s].

n is an integer.

day is a day of the month (1 through 31) or day of the week (sunday through saturday).

The following are examples of valid time expressions:

```
at 2:00
every 5 minutes
at 1:00 on tuesday
```

You may comment-out the RUN_TASK parameters for any tasks you do not want to run.

SCAN_FAST

Specifies for the run_filesystem_scan.sh script (such as in Example 7-9 on page 179) whether dmscanfs(8) will scan filesystems quickly (ON or YES) or will use its recursive option, which is much slower but results in pathnames being included in the output file (NO or OFF). The default is YES. If SCAN_FAST is disabled, the bfid2path output file is written in the daemon's *SPOOL_DIR* directory. The bfid2path file is optimized for use by dmstat(8).

SCAN_FILESYSTEMS

Specifies for the run_filesystem_scan.sh script the filesystems that dmscanfs will scan. The default is to scan all DMF-managed filesystems.

SCAN_FOR_DMSTAT

Specifies for the run_filesystem_scan.sh script whether the fhandle2bfid+path file is created (ON or YES) or not (OFF or NO). The fhandle2bfid+path file may be used by the dmemptytape(8) command. The default is YES.

SCAN_OUTPUT

Specifies for the run_filesystem_scan.sh script the name of the file into which dmscanfs will place output. The default is /tmp/dmscanfs.output.

SCAN_PARALLEL

Specifies for the run_filesystem_scan.sh script whether dmscanfs will scan filesystems in parallel. (YES or ON) or not (NO or OFF). The default is OFF.

Note: Setting this parameter to ON for a daemon task taskgroup may result in the filesystem scan completing in a shorter period of time, but it may also result in an unacceptable level of filesystem activity generated by the script that interferes with user processes.

SCAN_PARAMS

Specifies for the run_filesystem_scan.sh script other dmscanfs parameters, such as for example requesting a nondefault output format.

Note: SGI recommends that you do not include the -q option (which suppresses the dmscanfs header line) as a value for SCAN_PARAMS because it makes the output file harder to parse with general-purpose scripts. The run_daily_report.sh script requires that this header line be present.

TSREPORT_OPTIONS

Specifies for the run_daily_tsreport.sh script additional options that will be added to the end of the tsreport command line. For example, specifying

`--host` will add an additional column with the hostname to the report. (This parameter is optional).

Daemon Tasks

The following example shows the task groups for daemon tasks and describes the specific information required by the scripts.

Example 7-9 `taskgroup` Object for Daemon Tasks

```
define daemon_tasks
        TYPE                taskgroup
        RUN_TASK            $ADMINDIR/run_filesystem_scan.sh at 2:00

        RUN_TASK            $ADMINDIR/run_daily_report.sh at 3:00

        RUN_TASK            $ADMINDIR/run_daily_drive_report.sh at 4:00

        RUN_TASK            $ADMINDIR/run_audit.sh every day \
                               at 23:00

        RUN_TASK            $ADMINDIR/run_scan_logs.sh at 00:01

        RUN_TASK            $ADMINDIR/run_remove_logs.sh every \
                               day at 1:00

        RUN_TASK            $ADMINDIR/run_daily_tsreport.sh every \
                               day at 5:00

        LOG_RETENTION       4w

        RUN_TASK            $ADMINDIR/run_remove_journals.sh every \
                               day at 1:00
        JOURNAL_RETENTION   4w

        RUN_TASK            $ADMINDIR/run_copy_databases.sh \
                            every day at 3:00 12:00 21:00
        DATABASE_COPIES     /save/dmf_home /alt/dmf_home
enddef
```

In the above example:

- The name of this task group is daemon_tasks. This can be any name you like, but it must be the same as the name provided for the TASK_GROUPS parameter of the dmdaemon object. See Example 7-4 on page 163.
- $ADMINDIR in the path will be substituted with /usr/lib/dmf. When the task is run, it will be given the name of the object that requested the task as the first parameter and the name of the task group (in this case, daemon_tasks) as the second parameter.

- The scripts specify the following:
  - The `run_filesystem_scan.sh` task runs `dmscanfs` (8)on filesystems specified by `SCAN_FILESYSTEMS` (by default, all DMF-managed filesystems) writing the output to a file specified by `SCAN_OUTPUT` (by default `/tmp/dmscanfs.output`).

    This file, if it exists, is used by `run_daily_report.sh` and `dmstat`(8) and may be of use to site-written scripts or programs. Although DMF does not require this file, the output from `run_daily_report.sh` and `dmstat` will be incomplete if it is unavailable.

    You must specify the time at which the `run_filesystem_scan.sh` script is run, in this case 2:00 AM.

    Because `SCAN_FAST` is not set, its default value of `YES` means that pathnames will not be included in the output file and the `bfid2path` file will not be created.

    Because `SCAN_FOR_DMSTAT` (a misnomer) is not specified, its default value of `YES` means that the `fhandle2bfid+path` file is created in the daemon's *SPOOL_DIR* directory.
  - The `run_daily_report.sh` task reports on DCMs and managed filesystems (if `run_filesystem_scan.sh` has been run recently) and on all LSs. You must specify the time at which this report will run, in this case 3:00 AM.
  - The `run_daily_drive_report.sh` task generates a report showing tape drives that have requested or required cleaning since the report was last run. If the time that the report was last run cannot be determined, or if this is the first time that the report was run, the reporting period is the previous 24 hours. You must specify the time at which the report will run, in this case 4:00 AM.

    The report uses information that the program `dmtscopy` copies from files in `/var/spool/ts/pd/log` to the directory *SPOOL_DIR*`/tspdlogs`. Only events from files in *SPOOL_DIR*`/tspdlogs` are reported. Information is not reported from tape drives that are not used with `ts`.
  - The `run_audit.sh` task runs `dmaudit`. For this task, provide a *time_expression*. In this case, the script will run each day at 11:00 PM.

    If it detects any errors, the `run_audit.sh` task mails the errors to the e-mail address defined by the `ADMIN_EMAIL` parameter of the `base` object (described in "`base` Object" on page 152).

– The `run_scan_logs.sh` task scans the DMF log files for errors. For this task, provide a *time_expression*. In this case, the script will run at 12:01 AM.

  If the task finds any errors, it sends e-mail to the e-mail address defined by the `ADMIN_EMAIL` parameter of the `base` object.

– The `run_remove_logs.sh` task removes logs that are older than the value you provide by specifying the `LOG_RETENTION` parameter. You also provide a *time_expression* to specify when you want the `run_remove_logs.sh` to run. In the example, log files more than 4 weeks old are deleted each day at 1:00 A.M.

– The `run_daily_tsreport.sh` task generates a report containing the output of the `tsreport` command. The reporting period covers the time since the task was last run. If that cannot be determined, the reporting period is the previous 24 hours. In the example, the report will run every day at 5:00 A.M.

  The report uses information that the program `dmtscopy` copies from files in `/var/spool/ts/pd/log` to the directory *SPOOL_DIR*`/tspdlogs`. Only events from files in *SPOOL_DIR*`/tspdlogs` are reported. Information is not reported from tape drives that are not used with `ts`.

  The task uses the following options for the `tsreport` command:

```
--noversion
--options
--wide
--tapestats
--drivestats
--errors
--tapealert
--startdate
--starttime
```

– The `run_remove_journals.sh` task removes journals that are older than the value you provide by specifying the `JOURNAL_RETENTION` parameter. You also provide a *time_expression* to specify when you want the `run_remove_journal.sh` to run. In the example, journal files more than 4 weeks old are deleted each day at 1:00 A.M.

Note: The `run_remove_journals.sh` and `run_remove_logs.sh` tasks are not limited to the daemon logs and journals; they also clear the logs and journals for MSPs and LSs.

– The `run_copy_databases.sh` task makes a copy of the DMF databases. For this task, specify a value for *time_expression* and `DATABASE_COPIES`. In this case, the script will run each day at 3:00 AM, 12:00 noon, and 9:00 PM.

  The task copies a snapshot of the current DMF databases to the directory specified by `DATABASE_COPIES` that contains the oldest copy. In the example, the copy would be made to either `/save/dmf_home` or `/alt/dmf_home`. Integrity checks are done on the databases before the copy is saved. If the checks fail, the copy is not saved, and the task sends e-mail to the address defined by the `ADMIN_EMAIL` parameter of the `base` object.

Dump Tasks

You can configure `taskgroup` object parameters to manage how the daemon completes the following tasks to back up the DMF-managed filesystems:

- Fully back up DMF-managed filesystems (the `run_full_dump.sh` task)
- Partially back up DMF-managed filesystems (the `run_partial_dump.sh` task)
- Hard-delete files no longer on backup tape (the `run_hard_deletes.sh` task)
- Manage the data from the filesystem dumps (the `DUMP_TAPES`, `DUMP_RETENTION`, `DUMP_DEVICE`, `DUMP_MIGRATE_FIRST`, `DUMP_INVENTORY_COPY`, `DUMP_FILE_SYSTEMS`, and `DUMP_VSNS_USED` parameters).

For each of these tasks, you can configure when the task is run. To manage the tapes, you must provide information such as:

- Tape and device names
- Retention times for output
- Whether to migrate files before dumping the filesystem
- Locations for inventory files

Table 7-1 on page 172 provides a summary of automated maintenance tasks.

The following example configures a `taskgroup` object named `dump_tasks`. You can give the `taskgroup` object any name you like, but do not change the script names. You may comment-out the `RUN_TASK` parameters for any tasks you do not want to run.

Example 7-10 `taskgroup` Object for Dump Tasks

```
define  dump_tasks
        TYPE                    taskgroup
        RUN_TASK                $ADMINDIR/run_full_dump.sh on \
                                  sunday at 00:01
        RUN_TASK                $ADMINDIR/run_partial_dump.sh on \
                                  monday tuesday wednesday thursday \
                                  friday saturday at 00:01
        RUN_TASK                $ADMINDIR/run_hard_deletes.sh \
                                  at 23:00
        DUMP_TAPES              HOME_DIR/tapes
        DUMP_RETENTION          4w
        DUMP_DEVICE             SILO_2
        DUMP_MIGRATE_FIRST      yes
        DUMP_INVENTORY_COPY     /save/dump_inventory
enddef
```

In the above example:

- The name of this task group is `dump_tasks`. This can be any name you like, but it must be the same as the name provided for the `TASK_GROUPS` parameter of the `dmdaemon` object. See Example 7-4 on page 163.
- `$ADMINDIR` in the path will be substituted with `/usr/lib/dmf`. When the task is run, it will be given the name of the object that requested the task as the first parameter and the name of the task group (in this case, `dump_tasks`) as the second parameter.
- The `RUN_TASK` scripts specify the following:
  - The `run_full_dump.sh` task runs a full backup of DMF-managed filesystems at intervals specified by the *time_expression*. In this case, the full backup is run each week on Sunday morning one minute after midnight.
  - The `run_partial_dump.sh` task backs up only those files in DMF-managed filesystems that have changed since the time a full backup was completed. The backups are run at intervals specified by the *time_expression*. In the example, it is run each day of the week except Sunday, at one minute after midnight.
  - The `run_hard_deletes.sh` task removes from the DMF databases any files that have been deleted but can no longer be restored because the backup tapes have been recycled (that is, the task hard-deletes the files). The backup tapes are recycled at the time interval set by the `DUMP_RETENTION` parameter

described in the next step. For more information on hard-deleting files, see "Soft- and Hard-Deletes" on page 348.

– The other parameters in Example 7-10 determine how the data from the filesystem dumps will be managed:

  • The file `HOME_DIR/tapes` contains the VSNs of tapes that can be used by the dump tasks (where `HOME_DIR` is specified in the `base` object, see "`base` Object" on page 152)

  • The backups will be kept for 4 weeks

  • The drive group that defines how to mount the tapes is `SILO_2`

  • The `dmmigrate` command will be run before the dumps are taken

  • The XFS inventory files will be copied into `/save/dump_inventory`

Node Tasks

The following example shows the task groups for node tasks (when using the Parallel Data Mover Option) and describes the specific information required by the scripts.

Example 7-11 `taskgroup` Object for Node Tasks

```
define  node_tasks
    TYPE              taskgroup
    RUN_TASK          $ADMINDIR/run_remove_logs.sh every day at 1:00
    LOG_RETENTION     4w
enddef
```

In the above example:

• The name of this task group is `node_tasks`. This can be any name you like, but it must be the same as the name provided for the `TASK_GROUPS` parameter of the `services` object. See "`services` Object" on page 167.

• The `run_remove_logs.sh` task removes logs that are older than the value you provide by specifying the `LOG_RETENTION` parameter. You also provide a *time_expression* to specify when you want the `run_remove_logs.sh` to run.

  In the example, log files more than 4 weeks old are deleted each day at 1:00 A.M.

When using the Parallel Data Mover Option, you should define the `run_remove_logs.sh` task for the `taskgroup` that applies to the `node` object rather than for the `taskgroup` that applies to the `dmdaemon` object.

Note: The `run_remove_logs.sh` task is the only task available for `service` objects.

`device` Object

Normally, a `drivegroup` object is used to define the tape devices to be used by a `taskgroup` object (such as the example `dump_tasks`), with the DMF LS and the backup scripts sharing the same devices. However, if backups are to use different drives from those in use by DMF, they should be defined by a `device` object. The parameters you define are based on the mounting service you intend to use.

The following parameters are common to all `device` objects:

Parameter	Description
`TYPE`	`device` (required name for this type of object). There is no default.
`MOUNT_SERVICE`	Specifies the mounting service. Possible values are `openvault` and `tmf`. You must use `openvault` for those drive groups that contain tape drives on parallel data mover nodes. The default is `openvault`.
`MOUNT_SERVICE_GROUP`	Specifies the name by which the object's devices are known to the mounting service. For TMF, this is the device group name that would be used with the `-g` option on the `tmmnt` command. For OpenVault, this is the OpenVault `drivegroup` name that would be listed by the `ov_drivegroup` command. If this parameter is not specified, the `device` object's name is used.
`OV_ACCESS_MODES`	Specifies the OpenVault access mode. The default is `readwrite` when migrating and `readonly` when recalling. The only possible value you can specify for this parameter is `readwrite` if you want to force the access to always be `readwrite`. (Other OpenVault

	access modes are not configurable in DMF; DMF always uses `rewind` and `variable`.)
`OV_INTERCHANGE_MODES`	Specifies a list of interchange mode names that control how data is written to tape. This can be used to control whether the device compresses data as it is written. This optional parameter is applied when a tape is mounted or rewritten.
`TMF_TMMNT_OPTIONS`	Specifies command options that should be added to the `tmmnt` command when mounting a tape. DMF uses the `-Z` option to `tmmnt`, so options controlling block size and label parameters are ignored. Use `-g` if the group name is different from the `device` object's name. Use `-i` to request compression.

`filesystem` Object

You must have a `filesystem` object for each filesystem on which DMF can operate:

- *Managed filesystems* are DMAPI-mounted XFS or CXFS filesystems on which DMF can migrate or recall files. (When using the Parallel Data Mover Option, they must be CXFS.) The object parameters specify the level of migration for the filesystem, I/O options, and (if applicable) policies for MSP selection, file weighting, and automatic space management.
- *Unmanaged filesystems* are POSIX filesystems (such as Lustre) that are not managed by DMF but from which you can efficiently copy files to secondary storage via the `dmarchive`(1) command. They do not support space management, migrations, or recalls. The `MIGRATION_LEVEL` parameter must be set to `archive`.

The `filesystem` object parameters are as follows:

Parameter	**Description**
`TYPE`	`filesystem` (required name for this type of object). There is no default.
`BUFFERED_IO_SIZE`	Specifies the size of I/O requests when reading from this filesystem using buffered I/O. The legal range of values is `4096` through `16777216`. The default is `262144`. However, this parameter is ignored when

recalling files if USE_UNIFIED_BUFFER is set to on (which is the default.)

DIRECT_IO_SIZE — Specifies the size of I/O requests when reading from this filesystem using direct I/O. The legal range of values is 65536 through 18446744073709551615. The default value depends on the filesystem's configuration, but will not exceed the value of DIRECT_IO_MAXIMUM_SIZE defined in the base object. This parameter is ignored if the filesystem does not support direct I/O. For more information about direct I/O, see O_DIRECT in the open(2) man page.

MAX_MANAGED_REGIONS — Sets the maximum number of managed regions that DMF will assign to a file on a per-filesystem basis. You can set MAX_MANAGED_REGIONS to any number that is less than the actual number of regions that will fit in a filesystem attribute. For XFS and CXFS filesystems, that number is 3275.

By default, DMF allows a DMF attribute to contain the maximum number of managed regions that will still allow the attribute to fit completely inside the inode, based on inode size and attr type. The default value for a filesystem object that does not have a MAX_MANAGED_REGIONS parameter is calculated at filesystem mount time. This value is chosen to ensure that the DMF attribute will fit inside the inode, assuming that no other attribute (such as an ACL) is already occupying the inode's attribute space. Table 4-1 lists the default maximum file regions. This parameter does not apply to filesystems with a MIGRATION_LEVEL of archive.

Caution: You should use MAX_MANAGED_REGIONS cautiously. If you set this parameter to a value that is larger than the default maximum (see Table 4-1 on page 87), the DMF attribute may not fit inside the inode. If there are many files with DMF attributes outside of the inode, filesystem scan times can increase greatly. To avoid this problem, SGI recommends that a file that has exceeded the maximum default file regions be made offline (that is, having a single region) as soon as possible after the online data has been accessed.

MESSAGE_LEVEL

Specifies the highest message level number that will be written to the automated space management log (autolog). It must be an integer in the range 0-6; the higher the number, the more messages written to the log file. The default is 2. This parameter applies only to filesystems with a MIGRATION_LEVEL of auto.

For more information on message levels, see Chapter 9, "Message Logs" on page 277.

MIGRATION_LEVEL

Sets the level of migration service for the filesystem for migration to offline media. (Recall from offline media is not affected by the value of MIGRATION_LEVEL.) Valid values are:

- archive (only for DMF direct archiving via the dmarchive command)
- none (no migration)
- user (only user-initiated migration using the dmput or dmmigrate commands)
- auto (automated space management, default)

The migration level actually used for the filesystem is the lesser of the MIGRATION_LEVEL of the dmdaemon object and this value. If you do not want automated space management for a filesystem, set

MIGRATION_LEVEL to user or none. The default is auto.

When using the Parallel Data Mover Option, all DMF-managed filesystems (that is, filesystems where DMF can migrate or recall files) must be CXFS filesystems.

MIN_ARCHIVE_SIZE

Specifies the minimum file size required for the dmarchive command to copy data directly between an unmanaged filesystem and DMF secondary storage. Files smaller than this size will instead be copied between the unmanaged filesystem and a DMF-managed filesystem before being possibly migrated or recalled from DMF secondary storage. The legal range of values is 1 through 18446744073709551615. The default is 1. This parameter applies only to filesystems with a MIGRATION_LEVEL value of archive.

MIN_DIRECT_SIZE

Determines whether direct or buffered I/O is used when reading from this filesystem. If the number of bytes to be read is smaller than the value specified, buffered I/O is used, otherwise direct I/O is used. The legal range of values is 0 (direct I/O is always used) through 18446744073709551615 (direct I/O is never used). The default is 0. This parameter is ignored if the filesystem does not support direct I/O or is a real-time filesystem. For more information about direct I/O, see O_DIRECT in the open(2) man page.

Note: Buffered I/O is always used when writing to a filesystem.

POLICIES	Specifies the names of the configuration objects defining policies for this filesystem. Policies are defined with policy objects (see "policy Object"). The POLICIES parameter is required; there is no default value. A policy can be unique to each DMF-managed filesystem or it can be reused numerous times. This parameter does not apply to filesystems with a MIGRATION_LEVEL of archive.
POSIX_FADVISE_SIZE	Specifies the number of bytes after which DMF will call posix_fadvise()with advice POSIX_FADV_DONTNEED when recalling files. The minimum value is 0, which means that posix_fadvise is never used. The maximum value is 18446744073709551615. The default is 100000000, which will call posix_fadvise after each 100,000,000 bytes (approximately) it has written to the file. DMF does not synchronize the file at this point. If POSIX_FADVISE_SIZE is set to a nonzero value, DMF will also call posix_fadvise when a region is made online. Setting this parameter to a small, nonzero value may have an adverse affect on performance. See the posix_fadvise(2) man page for more information.
TASK_GROUPS	Names the task groups that contain tasks the daemon should run when MIGRATION_LEVEL is set to auto. They are configured as taskgroup objects. There is no default. There are no defined tasks for filesystems.
USE_UNIFIED_BUFFER	Determines how DMF manages its buffers when recalling files on this filesystem. The value can be one of the following: • on, which means that DMF will use the same buffer for reading and writing and BUFFERED_IO_SIZE is ignored when recalling files. Setting the value to on will cause the size of I/O requests to be small when recalling data from a disk, DCM, or FTP MSP. The default setting is on. • off, which means that DMF uses separate buffers for reading and writing during recall. That is, DMF

reads data from its backing store (such as tape) into a buffer and then copies the data into another buffer for writing. An additional thread for writing is also used.

The following example defines a `filesystem` object named `/c`.

Example 7-12 `filesystem` Object

```
define  /c
        TYPE                    filesystem
        MIGRATION_LEVEL         user
        POLICIES                fs_msp
enddef
```

In the example above:

- The `define` parameter must have a value that is the mount point of the filesystem you want DMF to manage, in this case `/c`. Do not use the name of a symbolic link.
- Only user-initiated migration will be used for migration to offline media.
- The migration policy is set by the `policy` object named `fs_msp`. See "`policy` Object" on page 193.

The following example defines a `filesystem` object for an unmanaged filesystem named `/lustrefs`:

Example 7-13 `filesystem` Object for DMF Direct Archiving

```
define  /lustrefs
        TYPE                    filesystem
        MIGRATION_LEVEL         archive
        MIN_ARCHIVE_SIZE        262144
enddef
```

In the example above:

- The `define` parameter must have a value that is the mount point of the unmanaged filesystem, in this case `/lustrefs`. Do not use the name of a symbolic link.
- File data in `/lustrefs` can be can be copied directly to secondary storage by users via the `dmarchive` command.
- Files that are smaller than 262144 bytes are never archived via `dmarchive` but instead will be copied to a DMF-managed filesystem before being possibly migrated or recalled from DMF secondary storage.

`policy` Object

This section discusses the following:

Functions of `policy` Parameters

A `policy` object specifies behavior for managing the following:

- A user filesystem
- A DCM `STORE_DIRECTORY`

The `policy` object parameters specify the following functions:

- "Automated Space Management Overview" on page 194
- "File Weighting Overview" on page 194
- "MSP/VG Selection Overview" on page 195

For details about the parameters, see:

- "User Filesystem `policy` Parameters" on page 196
- "DCM `STORE_DIRECTORY` `policy` Parameters" on page 201

Automated Space Management Overview

DMF lets you automatically monitor filesystems and migrate data as needed to prevent filesystems from filling. This capability is implemented by the `dmfsmon`(8) daemon. After the `dmfsmon` daemon has been initiated, it will begin to monitor the DMF-managed filesystem in order to maintain the level of free space specified in the configuration file.

Note: Ideal values for these parameters are highly site-specific, based largely on filesystem sizes and typical file sizes.

File Weighting Overview

When DMF is conducting automated space management, it derives an ordered list of files (called a *candidate list*) and migrates or frees files starting at the top of the list. The ordering of the candidate list is determined by weighting factors that are defined by parameters in the configuration file. You can use the file weighting parameters multiple times to specify that different files should have different weights.

For more details, see Chapter 10, "Automated Space Management" on page 279.

MSP/VG Selection Overview

DMF can be configured to have many MSPs/VGs. Each MSP/VG manages its own set of volumes. The MSP/VG selection parameters let you migrate files with different characteristics to different MSPs /VGs. You can use the MSP/VG selection parameters multiple times to specify that different files should have different MSP/VG selection values.

Rules for `policy` Parameters

This section discusses the following:

- "User Filesystem Rules" on page 195
- "DCM `STORE_DIRECTORY` Rules" on page 196

User Filesystem Rules

The rules for a `policy` object that is migrating a user filesystem are as follows:

- The `POLICIES` parameter for a `filesystem` object must specify one and only one MSP/VG selection policy.
- The `TYPE` parameter is required for any `policy` object:

Parameter	Description
`TYPE`	`policy` (required name for this type of object). There is no default.

- If the `MIGRATION_LEVEL` for a `filesystem` object is `auto`, the `POLICIES` parameter for that object must specify one and only one space-management policy.
- You do not need to specify a weighting policy if the default values are acceptable.
- Providing the above rules are followed, you can have many different combinations of policies. For example, you could configure one policy that defines all three categories of policy parameters (automated space management, MSP/VG selection, and file weighting) and share that policy among all the filesystems, or you could configure any number of individual MSP/VG selection policies and space-management policies (including weighting parameters) that you can apply to one or more filesystems.

DCM STORE_DIRECTORY Rules

The rules for a `policy` object that is managing a DCM `STORE_DIRECTORY` are as follows:

- The `TYPE` parameter is required for any `policy` object:

Parameter	Description
`TYPE`	`policy` (required name for this type of object). There is no default.

- If the `MIGRATION_LEVEL` for a `filesystem` object is `auto`, the `POLICIES` parameter for that object must specify one and only one space-management policy.
- You do not need to specify a weighting policy if the default values are acceptable.
- You can configure one policy that defines all three categories of policy parameters (automated space management, file weighting, and VG selection) and share that policy among all the filesystems. Alternatively, you might create a VG selection policy for all filesystems and a space-management policy (including weighting parameters) for all filesystems.
- The DCM supports the concept of *dual-residence*, which means that a cache-resident copy of a migrated file has already been copied to tape and can therefore be released quickly in order to prevent the cache filling, without any need to first copy it to tape. It is analogous to a dual-state file that is managed by the standard disk MSP and has equivalent policy parameters to control it.
- The age and space weighting parameters refer to the copies in the cache, not the originals in the managed filesystem.

User Filesystem `policy` Parameters

This section discusses the following:

Automated Space Management Parameters for a User Filesystem

The following parameters control automated space management for a user filesystem:

Parameter	Description
FREE_DUALSTATE_FIRST	When set to on, specifies that dmfsmon will first free dual-state and partial-state files before freeing files it must migrate. The default is off.
FREE_SPACE_DECREMENT	Specifies the percentage of filesystem space by which dmfsmon will decrement FREE_SPACE_MINIMUM if it cannot find enough files to migrate so that the value is reached. The decrement is applied until a value is found that dmfsmon can achieve. If space later frees up, the FREE_SPACE_MINIMUM is reset to its original value. Valid values are in the range 1 through the value of FREE_SPACE_TARGET. The default is 2.
FREE_SPACE_MINIMUM	Specifies the minimum percentage of free filesystem space that dmfsmon maintains. dmfsmon will begin to migrate files when the available free space for the filesystem falls below this percentage value. This parameter is required; there is no default.
FREE_SPACE_TARGET	Specifies the percentage of free filesystem space that dmfsmon will try to achieve if free space reaches or falls below FREE_SPACE_MINIMUM. This parameter is required; there is no default.
MIGRATION_TARGET	Specifies the percentage of filesystem capacity that DMF maintains as a reserve of dual-state files whose online space can be freed if free space reaches or falls below FREE_SPACE_MINIMUM. dmfsmon tries to make sure that this percentage of the filesystem is migrated, migrating, or free after it runs to make space available. This parameter is required; there is no default.
SITE_SCRIPT	Specifies the site-specific script to execute when dmfsfree, dmdskfree, or dmfsmon is run. If it returns a zero exit status, dmfsfree, dmdskfree, or dmfsmon continue their normal processing. If nonzero, they return immediately, using this value as their own

exit status. See dmfsfree(8) or dmdskfree(8) for further details. This parameter is optional.

For more details, see Chapter 10, "Automated Space Management" on page 279.

See also:

- "User Filesystem Rules" on page 195
- "Functions of policy Parameters" on page 194

File Weighting Parameters for a User Filesystem

The following parameters control file weighting for a user filesystem:

Parameter	Description
AGE_WEIGHT	Specifies a floating-point constant and floating-point multiplier to use when calculating the weight given to a file's age, calculated as follows: *constant* + (*multiplier* * *file_age_in_days*) If DMF cannot locate values for this parameter, it uses a floating point constant of 1 and a floating point multiplier of 1. The AGE_WEIGHT parameter accepts an optional when clause, which contains a conditional expression. See "when clause" on page 205. The AGE_WEIGHT parameter also accepts an optional ranges clause, which specifies the ranges of a file for which the parameter applies. See "ranges clause" on page 206.
SPACE_WEIGHT	Specifies a floating-point constant and floating-point multiplier to use when calculating the weight given to a file's size, calculated as follows: *constant* + (*multiplier* * *file_disk_space_in_bytes*) If DMF cannot locate values for this parameter, it uses a floating point constant of 0 and a floating point multiplier of 0.

For a partial-state (PAR) file, *file_disk_space_in_bytes* is the amount of space occupied by the file at the time of evaluation.

The SPACE_WEIGHT parameter accepts an optional when clause, which contains a conditional expression. See "when clause" on page 205.

The SPACE_WEIGHT parameter also accepts an optional ranges clause, which specifies the ranges of a file for which the parameter applies. See "ranges clause" on page 206.

See also:

- "User Filesystem Rules" on page 195
- "Functions of policy Parameters" on page 194

MSP/VG Selection Parameters for a User Filesystem

The following parameters control MSP/VG selection for a user filesystem:

Parameter	Description
SELECT_MSP, SELECT_VG	Specifies the MSPs/VGs to use for migrating a file. **Note:** The parameters are not used for defining which MSP/VG to use for recalls; for that, see the definitions of the LS_NAMES, MSP_NAMES, DRIVE_GROUPS, and VOLUME_GROUPS parameters. The SELECT_MSP and SELECT_VG names are equivalent. VGs, disk MSPs, and FTP MSPs may be specified by either parameter. You can list as many MSP/VG names as you have msp or volumegroup objects defined (separate the names with white space). A copy of the file will be migrated to each MSP/VG listed. The special MSP/VG name none means that the file will not be migrated. If no SELECT_MSP or SELECT_VG parameter applies to a file, it will not be migrated. The parameters are processed in the order that they appear in the policy. These parameters allow conditional expressions based on the value of a file tag. See "Customizing DMF" on page 93. The root user on the DMF server can override the selection specified in these parameters through the use of dmput -V or with libdmfusr.so calls. If site–defined policies are in place, they may also override these parameters. There is no default.

See also:

- "User Filesystem Rules" on page 195
- "Functions of `policy` Parameters" on page 194

DCM `STORE_DIRECTORY` `policy` Parameters

This section discusses the following:

- "Automated Space Management Parameters for a DCM `STORE_DIRECTORY`" on page 201
- "File Weighting Parameters for a DCM `STORE_DIRECTORY`" on page 202
- "VG Selection Parameters for a DCM `STORE_DIRECTORY`" on page 204

See also "Functions of `policy` Parameters" on page 194.

Automated Space Management Parameters for a DCM `STORE_DIRECTORY`

The following parameters control automated space management for a DCM `STORE_DIRECTORY`:

Parameter	Description
`DUALRESIDENCE_TARGET`	Specifies the percentage of DCM cache capacity that DMF maintains as a reserve of dual-resident files whose online space can be freed if free space reaches or falls below `FREE_SPACE_MINIMUM`. The `dmdskmsp` process tries to ensure that this percentage of the filesystem is copied to tape, is currently being copied to tape, or is free after it runs `dmdskfree` to make space available. This parameter is required for a DCM; there is no default. (It does not apply to user filesystems.)
`FREE_DUALRESIDENT_FIRST`	When set to `on`, specifies that `dmfsmon` will first free dual-resident files before freeing files it must migrate. The default is `off`.
`FREE_SPACE_DECREMENT`	Specifies the percentage of filesystem space by which `dmfsmon` will decrement `FREE_SPACE_MINIMUM` if it cannot find enough files to migrate so that the value is reached. The decrement is applied until a value is

found that dmfsmon can achieve. If space later frees up, the FREE_SPACE_MINIMUM is reset to its original value. Valid values are in the range 1 through the value of FREE_SPACE_TARGET. The default is 2.

FREE_SPACE_MINIMUM	Specifies the minimum percentage of free filesystem space that dmfsmon maintains. dmfsmon will begin to migrate files when the available free space for the filesystem falls below this percentage value. This parameter is required; there is no default.
FREE_SPACE_TARGET	Specifies the percentage of free filesystem space that dmfsmon will try to achieve if free space reaches or falls below FREE_SPACE_MINIMUM. This parameter is required; there is no default.
SITE_SCRIPT	Specifies the site-specific script to execute when dmfsfree, dmdskfree, or dmfsmon is run. If it returns a zero exit status, dmfsfree, dmdskfree, or dmfsmon continue their normal processing. If nonzero, they return immediately, using this value as their own exit status. See dmfsfree(8) or dmdskfree(8) for further details. This parameter is optional.

See also:

- "DCM STORE_DIRECTORY Rules" on page 196
- "Functions of policy Parameters" on page 194

File Weighting Parameters for a DCM STORE_DIRECTORY

The policy parameters for file weighting are as follows:

Parameter	Description
CACHE_AGE_WEIGHT	Specifies a floating-point constant and floating-point multiplier to use when calculating the weight given to a file's age, calculated as follows: *constant* + (*multiplier* * *file_age_in_days*)

If DMF cannot locate values for this parameter, it uses a floating point constant of `1` and a floating point multiplier of `1`.

Note: This parameter refers to the copies in the cache, not the originals in the managed filesystem.

The `CACHE_AGE_WEIGHT` parameter accepts an optional `when` clause, which contains a conditional expression. See "`when` clause" on page 205.

`CACHE_SPACE_WEIGHT` Specifies a floating-point constant and floating-point multiplier to use when calculating the weight given to a file's size, calculated as follows:

constant `+` (*multiplier* `*` *file_disk_space_in_bytes*)

If DMF cannot locate values for this parameter, it uses a floating point constant of `0` and a floating point multiplier of `0`.

For a partial-state (`PAR`) file, *file_disk_space_in_bytes* is the amount of space occupied by the file at the time of evaluation.

The `CACHE_SPACE_WEIGHT` parameter accepts an optional `when` clause, which contains a conditional expression. See "`when` clause" on page 205.

See also:

- "DCM `STORE_DIRECTORY` Rules" on page 196
- "Functions of `policy` Parameters" on page 194

VG Selection Parameters for a DCM STORE_DIRECTORY

The following parameter controls VG selection for a DCM STORE_DIRECTORY:

Parameter	Description
SELECT_LOWER_VG	Defines which VGs should maintain tape-based copies of files in the cache, and under what conditions that would define dual-residence. **Note:** The parameter is not used for defining which VG to use for recalls; for that, see the definitions of the LS_NAMES, MSP_NAMES, DRIVE_GROUPS, and VOLUME_GROUPS parameters. You can list as many VG names as you have volumegroup objects defined (separate the names with white space). A copy of the file will be migrated to each VG listed. The special VG name none means that the file will not be migrated. If no SELECT_LOWER_VG parameter applies to a file, it will not be migrated. The parameters are processed in the order that they appear in the policy. These parameters allow conditional expressions based on the value of a file tag. See "Customizing DMF" on page 93. The root user on the DMF server can override the selection specified in these parameters through the use of dmput -V or with libdmfusr.so calls. If site–defined policies are in place, they may also override these parameters. There is no default.

See also:

- "DCM STORE_DIRECTORY Rules" on page 196
- "Functions of policy Parameters" on page 194

when clause

The file weighting and MSP/VG selection parameters accept an optional when to restrict the set of files to which that parameter applies. It has the following form:

when *expression*

expression can include any of the following simple expressions:

Expression	Description
age	Specifies the number of days since last modification or last access of the file, whichever is more recent.
gid	Specifies the group ID or group name of the file.
sitefn	Invokes a site-defined policy function once for each file being considered, and is replaced by the return code of the function. This is only applicable to the AGE_WEIGHT, SPACE_WEIGHT, SELECT_MSP, and SELECT_VG parameters in a filesystem's policy stanza. For more information, see Appendix C, "Site-Defined Policy Subroutines and the sitelib.so Library" on page 433.

sitetag	Specifies a site-determined number associated with a file by the dmtag(1) command, in the range 0—4294967295. For example:
	`sitetag = 27`
	`sitetag in (20-40, 5000, 4000000000)`

size	Specifies the logical size of the file, as shown by ls -l.
softdeleted	Specifies whether or not the file corresponding to a cached copy has been soft deleted; only applicable to the CACHE_AGE_WEIGHT, CACHE_SPACE_WEIGHT, and SELECT_LOWER_VG parameters in a DCM policy stanza. Values are false and true.

space	Specifies the number of bytes the file occupies on disk (always a multiple of the block size, which may be larger or smaller than the length of the file). For a partial-state (PAR) file, the value used is the space that the file occupies on disk at the time of evaluation.
uid	Specifies the user ID or user name of the file.

Combine expressions by using and, or, and ().

Use the following operators to specify values:

```
=
!=
>
<
>=
<=
in
```

The following are examples of valid expressions:

```
space < 10m             (space used is less than 10 million bytes)
uid <= 123              (file's user ID is less than or equal to 123)
gid = 55                (file's group ID is 55)
age >= 15               (file's age is greater than or equal to 15 days)
space > 1g              (space used is greater than 1 billion bytes)
uid in (chris, 10 82-110 200)
                        (file owner's user name is chris or
                        the file owner's UID is  10, in the range 82-110, or 200)
(gid = 55 or uid <= 123) and age < 5
                        (file's age is less than 5 days and its
                        group ID is 55 or its user ID is less than or equal to 123)
```

ranges clause

The AGE_WEIGHT and SPACE_WEIGHT parameters accept an optional ranges clause to restrict the ranges of a file for which a parameter applies. Example 7-15, page 212 shows an example of a policy that contains ranges clauses.

Note: The `ranges` clause is not valid with the `CACHE_AGE_WEIGHT` or `CACHE_SPACE_WEIGHT` parameters.

The clause has the following form, where *byteranges* is one or more byte ranges:

`ranges` *byteranges*

Each byte range consists of a set of numbers that indicate byte positions. (You can also use `BOF` or `bof` to indicate the first byte in the file and `EOF` or `eof` to indicate the last byte in the file.) Each byte range is separated by a comma and can have one of the following forms:

- A specification of two byte positions, where *first* specifies the first byte in the range and *last* specifies the last byte in the range:

  first`:`*last*

  If unsigned, *first* and *last* count from the beginning of the file; if preceded by a minus sign (`-`), they count backwards from the end of the file.

  The first byte in the file is byte `0` or `BOF` and the last byte is `-0` or `EOF`. Therefore, `BOF:EOF` and `0:-0` both define a range covering the entire file.

  For example:

  – `ranges 0:4095` specifies the first 4096 bytes of the file

  – `ranges -4095:EOF` specifies the last 4096 bytes of the file

- A specification of the size of the range, starting at a given point, where *first* is a byte position as above and *size* is the number of bytes in the range, starting at *first*:

  first`+`*size*

  For example, the following indicates bytes 20 through 29:

  ```
  ranges 20+10
  ```

  If *size* is preceded by a minus sign, it specifies a range of *size* bytes ending at *first*. For example, the following indicates bytes 11 through 20:

  ```
  ranges 20+-10
  ```

- A specification of the size of the range only (without a colon or plus symbol), assumed to start at the end of file (when preceded by a minus sign) or beginning of file:

  -size
  size

  For example, the following specifies the last 20 bytes in the file:

  ```
  ranges -20
  ```

The *first*, *last*, or *size* values can be of the following forms:

- A hexadecimal number: `0x`*n*
- A based number: *base#n*
- A decimal number with an optional trailing scaling character. The decimal number may include a decimal point (`.`) and exponent. The trailing scaling character may be one of the following (all of which are powers of 1000, not 1024):

  `k` or `K` for 1 thousand
  `m` or `M` for 1 million
  `g` or `G` for 1 billion
  `t` or `T` for 1 trillion
  `p` or `P` for 1 quadrillion

Note: DMF may round byte ranges and join nearby ranges if necessary. If a range is given a negative weight, rounding may cause additional bytes to be ineligible for automated space management.

Do not use a `ranges` clause when partial-state files are disabled in DMF. Specifying many ranges for a file is discouraged, as it can cause the time and memory used by automated space management to grow. DMF has an upper limit on the number of regions that can exist within a file; this can sometimes cause a range to be given an effective lower weight than what was specified in the configuration file. This might happen if the file is already partial-state and the range with largest weight cannot be made offline (`OFL`) because that would create too many regions. If the file has too many regions to make the range offline, but it could be made offline at the same time as a range with lower weight, it will be given the lower weight. If more than one range in the middle of a file is not a candidate for automatic migration, the limit on

the number of regions may make it impossible to automatically free other regions of the file.

`policy` Configuration Procedures

The following procedures explain how to create policies for automated space management (including file weighting) and MSP/VG selection:

- "Automated Space-Management Procedure" on page 209
- "MSP/VG Selection Procedure" on page 213

Automated Space-Management Procedure

Procedure 7-1 Configuring Objects for Automated Space Management

The following steps explain pertinent information for configuring a `policy` object:

1. Ensure that `define` has a value you set previously in the `POLICIES` parameter of a `filesystem` object.
2. Ensure that `TYPE` is set to `policy`.
3. Configure automated space management as follows:
   a. Set `MIGRATION_TARGET` to an integer percentage of total filesystem space. DMF attempts to maintain this percentage as a reserve of space that is free or occupied by dual-state files that can be deleted if the filesystem free space reaches or falls below `FREE_SPACE_MINIMUM`.
   b. Configure `FREE_SPACE_TARGET` to an integer percentage of total filesystem space. DMF will try to achieve this level of free space when free space reaches or falls below `FREE_SPACE_MINIMUM`.
   c. Configure `FREE_SPACE_MINIMUM` to an integer percentage of the total filesystem space that DMF must maintain as free. DMF will begin to migrate files when the available free space for the configured filesystem reaches or falls below this percentage value.
   d. Configure `FREE_DUALSTATE_FIRST` to be `on` if you want DMF to free the space used by dual-state or partial-state files before it migrates and frees regular files. The default is `off`.

4. Configure the age and size weighting factors associated with a file when it is evaluated for migration as follows:

   a. The syntax of the parameter is a floating-point constant followed by a floating-point multiplier. The age weight is calculated as follows:

      constant + (*multiplier* x *age_in_days*)

      Add a `when` clause to select which files should use these values. DMF checks each `AGE_WEIGHT` parameter in turn, in the order that they occur in the configuration file. If the `when` clause is present and no `ranges` clause is present, DMF determines whether the file matches the criteria in the clause. If no `when` clause is present, a match is assumed. If the file matches the criteria, the file weight is calculated from the parameter values. If they do not match, the next instance of that parameter is examined.

      An `AGE_WEIGHT` of `1 1.0` is used if no `AGE_WEIGHT` applies for a file.

      In Example 7-14 below, files that have been accessed or modified within the last 10 days have a weight of 0. File migration likelihood increases with the length of time since last access because the file will have a greater weight. The final line specifies that files that have not been accessed or modified in 120 days or more have a far greater weight than all other files.

   b. The syntax of the parameter is a floating-point constant followed by a floating-point multiplier. Calculate the space weight as follows:

      constant + (*multiplier* x *file_disk_space_in_bytes*)

      In Example 7-14, the size of the file does not affect migration because all files have `SPACE_WEIGHT` of `0`.

      A default of `0 0.0` is used if no `SPACE_WEIGHT` applies for a file.

   c. Configure negative values to ensure that files are never automatically migrated. For example, you might want to set a minimum age for migration. The following parameter specifies that files that have been accessed or modified within 1 day are never automatically migrated:

      ```
      AGE_WEIGHT -1    0.0    when age <= 1
      ```

The following parameter specifies that small files are never automatically migrated:

```
SPACE_WEIGHT -1    0    when space <= 4k
```

d. If partial-state files are enabled on your machine (meaning that you have the PARTIAL_STATE_FILES configuration file parameter set to on and have the appropriate kernel installed, according to the information in the DMF release note), you can use the ranges clause to select ranges of a file.

DMF checks each AGE_WEIGHT parameter in turn, in the order that they occur in the configuration file. As described in step 4a above, DMF checks the when clause, if present, to see if the file matches the criteria. If the file matches and a ranges clause is present, DMF determines if that range has already been weighted. If it has not been weighted, the specified range is given the weight calculated from the parameter values. DMF examines the next instance of the parameter until all ranges in the file have been assigned a weight.

In Example 7-15 below, note the following about the AGE_WEIGHT parameters:

- If a file is owned by UID 624 and is 1004096 bytes long, the first 4096 bytes are given an AGE_WEIGHT of -1. The remaining 1000000 bytes are given an AGE_WEIGHT based on the age of the file; based on this weight, automated space management may select this file to be migrated. DMF migrates the entire file before changing its state to OFL, DUL, or PAR. Automated space management may also choose to put the last 1000000 bytes of the file offline based on the weight of that range; the first 4096 bytes will not be eligible for being put offline by automated space management.
- If a file is owned by UID 321, the first and last 4096 bytes of it will not be eligible for being put offline by automated space management, similar to the above situation.
- If a file is owned by UID 956, the policy in Example 7-15 would give the entire file an AGE_WEIGHT based on its age.

SPACE_WEIGHT parameters are evaluated similarly.

Note: DMF calculates the size weight and age weight separately. If either value is less than zero, the file is **not** automatically migrated and the file or range is **not** automatically freed. Otherwise, the two values are summed to form the file's or range's weight.

Example 7-14 `policy` Object for Automated Space Management

```
define  fs_space
        TYPE                    policy
        MIGRATION_TARGET        50
        FREE_SPACE_TARGET       10
        FREE_SPACE_MINIMUM      5
        FREE_DUALSTATE_FIRST    off

        AGE_WEIGHT 0    0.00    when age < 10
        AGE_WEIGHT 1    0.01    when age < 30
        AGE_WEIGHT 10   0.05    when age < 120
        AGE_WEIGHT 50   0.1

        SPACE_WEIGHT  0  0
enddef
```

Example 7-15 `policy` Object for Automated Space Management Using Ranges

```
define  fs2_space
        TYPE                    policy
        MIGRATION_TARGET        50
        FREE_SPACE_TARGET       10
        FREE_SPACE_MINIMUM      5
        FREE_DUALSTATE_FIRST    off

        AGE_WEIGHT -1.  0.00 ranges 0:4095 when uid=624
        AGE_WEIGHT -1   0    ranges 0:4095,-4095:EOF when uid=321
        AGE_WEIGHT 1    0.01    when age < 30
        AGE_WEIGHT 10   0.05    when age < 120
        AGE_WEIGHT 50   0.1

        SPACE_WEIGHT  0  0
enddef
```

Example 7-16 defines a `policy` object for MSP/VG selection.

Example 7-16 `policy` Object for an FTP MSP

```
define  fs_msp
        TYPE                    policy
        SELECT_MSP none         when space < 65536
        SELECT_MSP cart1 cart2  when gid = 22
        SELECT_MSP cart1        when space >= 50m
        SELECT_VG cart2
enddef
```

MSP/VG Selection Procedure

Procedure 7-2 Configuring Objects for MSP/VG Selection

The following steps explain pertinent information for configuring the above `policy` object:

1. Ensure that `define` has a value that you set previously in the `POLICIES` parameter of the `filesystem` object.
2. Ensure that `TYPE` is set to `policy`.
3. Ensure that the MSP/VG names you specify as the first value of the `SELECT_MSP` or `SELECT_VG` parameter is either:
   - The name of an MSP you set previously in the `MSP_NAMES` or `LS_NAMES` parameter of the `dmdaemon` object
   - The name of a VG that is a component of an LS named in that same parameter

   There is no default.
4. Configure MSP/VG selection criteria as follows:
   a. If you want to select an MSP/VG based on file size, use parameters such as the following, which send large files to `cart1` and small files to `cart2`:

      ```
      SELECT_MSP cart1        when space >= 50m
      SELECT_MSP cart2        when space >= 65536
      ```

      The order of the `SELECT` statements is important. The first `SELECT` statement that applies to the file is honored. For example, if the order of the statements above were reversed, a 50m file would be migrated to `cart2`, because the check for greater than or equal to (`>=`) 65536 would be done first, and it would be true.

b. If you want certain files to be copied to more than one MSP/VG, use syntax such as the following, which migrates all files that have a group ID of 22 to both of the configured MSPs or VGs:

```
SELECT_MSP cart1 cart2  when gid = 22
```

Separate multiple MSP/VG names with a blank space.

c. If you want to ensure that some files are never migrated, you can designate the MSP/VG selection as `none`. The following line from the sample file ensures that files smaller than 65,536 bytes are not migrated:

```
SELECT_MSP none          when space < 65536
```

Note: The `space` expression references the number of bytes the file occupies on disk, which may be larger or smaller than the length of the file. For example, you might use the following line in a policy:

```
SELECT_VG none when space < 4096
```

Your intent would be to restrict files smaller than 4 Kbytes from migrating.

However, this line may actually allow files as small as 1 byte to be migrated, because while the amount of data in the file is 1 byte, it will take 1 block to hold that 1 byte. If your filesystem uses 4–Kbyte blocks, the space used by the file is 4096, and it does not match the policy line.

To ensure that files smaller than 4 Kbytes do not migrate, use the following line:

```
SELECT_VG none when space <= 4096
```

(You could use either `SELECT_VG` or `SELECT_MSP` in these examples.)

LS Objects

Multiple objects are required to configure an LS. This section discusses the following:

libraryserver Object

The entry for an LS, one for each tape library, has the following parameters:

Parameters	Description
TYPE	libraryserver (required name for this type of object). There is no default.
CACHE_DIR	Specifies the directory in which the VG stores chunks while merging them from sparse tapes. If you do not specify this parameter, DMF uses the value of TMP_DIR from the base object. If you use the Parallel Data Mover Option and specify CACHE_DIR, it must either be or be in a CXFS filesystem.
CACHE_SPACE	Specifies the amount of disk space (in bytes) that dmatls can use when merging chunks from sparse tapes. During merging, small chunks from sparse tapes are cached on disk before being written to a tape. The default is 0, which causes all files to be merged via sockets. **Note:** The zone size influences the required cache space. See ZONE_SIZE in "volumegroup Object" on page 224.
COMMAND	Specifies the binary file to execute in order to initiate the LS. This value must be dmatls.
DISCONNECT_TIMEOUT	Specifies the number of seconds after which the LS will consider a mover process to have exited if it cannot

communicate with the process. Likewise, mover processes will use this value to determine if the LS has exited. The default value is 10 seconds.

DRIVE_GROUPS

Names one or more drive groups containing drives that the LS can use for mounting and unmounting volumes. They are configured as drivegroup objects. This parameter must be configured. There is no default.

The order of the values specified for this parameter is integral to the determination of the MSP/VG from which the DMF daemon attempts to recall an offline file. If the offline file has more than one copy, DMF uses a specific order when it attempts to recall the file. It searches for a good copy of the offline file in MSP or LS order, from the dmdaemon object's MSP_NAMES or LS_NAMES parameter. If one of those names refers to an LS, it searches for the copy in drive group order, from the libraryserver object's DRIVE_GROUPS parameter. It then searches for the copy in VG order from the drivegroup object's VOLUME_GROUPS parameter.

Note: Do not change this parameter while DMF is running.

MAX_CACHE_FILE

Specifies the largest chunk (in bytes) that will be merged using the merge disk cache. Larger files are transferred directly via a socket from the read child to the write child. The default is 25% of the CACHE_SPACE value. Valid values are 0 through the value of CACHE_SPACE.

MESSAGE_LEVEL

Specifies the highest message level number that will be written to the LS log, which includes messages from the LS's components. It must be an integer in the range 0-6; the higher the number, the more messages written to the log file. The default is 2.

RUN_TASK

See the description of RUN_TASK in "taskgroup Parameters" on page 174. Also see "Automated Maintenance Tasks" on page 89.

TASK_GROUPS	Names the task groups that contain tasks the LS should run. They are configured as `taskgroup` objects. There is no default.
WATCHER	Names the resource watcher that the LS should run. The default is no watcher. (A corresponding `resourcewatcher` object is required only if the default parameters are unacceptable. See "`resourcewatcher` Object" on page 231.)

`drivegroup` Object

The entry for a `drivegroup` object, one for each pool of interchangeable drives in a single library, has the following parameters:

Parameter	Description
TYPE	`drivegroup` (required name for this type of object). There is no default.
BANDWIDTH_MULTIPLIER	*(OpenVault only)* Specifies a floating point number used to adjust the amount of bandwidth that the LS assumes a drive in this drive group will use. The value is used when scheduling drives, which allows the administrator to adjust for the affects of compression. The default value is `1`, which means no compression. The minimum value is `.1` and the maximum is `1000`. The `node` object parameters `HBA_BANDWIDTH` and `NODE_BANDWIDTH` are related to this parameter; see "`node` Object" on page 163.
BLOCK_SIZE	Specifies the maximum block size to use when writing from the beginning of a tape. The block size in the VOL record of the LS database is updated with this value and is later used when reading or appending to a tape. DMF supports block sizes ranging from 4096 through 2097152 bytes. DMF will use direct I/O to tapes when possible. On some architectures, direct I/O cannot be used if the block size is larger than 524288; in this case, buffered I/O will be used instead. The default

maximum size is dependent on your device configuration, with DMF setting it as follows:

AMPEX™ DIS/DST	1199840
DLT™	131072
HP® ULTRIUM 4	524288
HP ULTRIUM 3	524288
HP ULTRIUM 2	262144
IBM® 03590B1A	16384
IBM 03590E1A	32768
IBM 03590H1A	16384
IBM 03592E05	131072
IBM 03592E06	262144
IBM ULTRIUM-TD1	131072
IBM ULT3580-TD1	131072
IBM ULTRIUM-TD2	262144
IBM ULT3580-TD2	262144
IBM ULTRIUM-TD3	262144
IBM ULT3580-TD3	262144
IBM ULTRIUM-TD4	524288
IBM ULT3580-TD4	524288
IBM ULTRIUM-HH4	524288
QUANTUM SDLT600	131072
QUANTUM SDLT320	131072
QUANTUM SuperDLT1	131072
SEAGATE ULTRIUM	262144
SONY SDX-700C	131072
SONY SDZ-100	131072
SONY SDZ-130	262144
SONY SDZ-200	524288
SONY SDZ-230	524288
STK™ 9840	126976
STK T9840B	126976
STK T9840C	262144
STK T9840D	262144
STK T9940A	262144
STK T9940B	262144
STK T10000A	524288
STK T10000B	524288
(Other drives)	65536

DRIVE_MAXIMUM	Specifies the maximum number of drives within this drive group that the LS is allowed to attempt to use simultaneously. This can be more or less than the number of drives the LS can physically detect. The maximum is 100; the default is 100 for drive groups. If a negative value is specified for DRIVE_MAXIMUM, the drive group uses the sum of the number of available drives and DRIVE_MAXIMUM.
DRIVE_SCHEDULER	Names the resource scheduler that the drive group should run for the scheduling of tape drives. They are configured as resourcescheduler objects. The default is a resource scheduler of default type and parameters. For the defaults, see "resourcescheduler Object" on page 230.
DRIVES_TO_DOWN	Specifies an integer value that controls the number of "bad" drives the drive group is allowed to try to configure down. When more than this number are down, whether due to the drive group or to external influences such as the system administrator, the drive group does not attempt to disable any more drives. The default of 0 prevents the drive group from disabling any drives.
LABEL_TYPE	Specifies the label type used when writing tapes from the beginning. Possible values are: • nl (no label) • sl (standard label, for IBM tapes) • al (ANSI label) The default is al.
MAX_MS_RESTARTS	Specifies the maximum number of times DMF can attempt to restart the mounting service (TMF or OpenVault) without requiring administrator intervention. The default and recommended values are 1 for TMF and 0 for OpenVault.
MOUNT_SERVICE	Specifies the mounting service. Possible values are openvault and tmf. You must use openvault for

	those drive groups that contain tape drives on parallel data mover nodes. The default is `openvault`.
`MOUNT_SERVICE_GROUP`	Specifies the name by which the drive group's devices are known to the mounting service. For TMF, this is the device group name that would be used with the `-g` option on the `tmmnt` command. For OpenVault, this is the drive group name that is specified by the `ov_drivegroup` command.
`MOUNT_TIMEOUT`	Specifies the maximum number of minutes to wait for a tape to be mounted. (The default is `0`, which means forever.) If a tape mount request waits for longer than this period of time, the drive group attempts to stop and restart provided that the `MAX_MS_RESTARTS` parameter allows it. Do not make this value too restrictive, as any non-LS tape activity (including `xfsdump`) can legitimately delay a VG's tape mount, which could result in this timeout being exceeded.
`MSG_DELAY`	Specifies the number of seconds that all drives in the drive group can be down before an e-mail message is sent to the administrator and an error message is logged. The default is `0`, which means that as soon as DMF notices that the mounting service is up and all of the drives are configured down, it will e-mail a message.
`OV_ACCESS_MODES`	Specifies the OpenVault access modes. The default is `readwrite` when migrating and `readonly` when recalling. The only possible value you can specify for this parameter is `readwrite` if you want to force the access to always be `readwrite`. (Other OpenVault access modes are not configurable in DMF: DMF always uses `rewind` and `variable`.)
`OV_INTERCHANGE_MODES`	*(OpenVault only)* Specifies a list of names to be provided to OpenVault for the `firstmount` clause when mounting a tape. Use `compression` to request compression. By default, this list is empty.

POSITIONING

Specifies how the tape should be positioned. The values can be:

- `skip`, which means use tape mark skipping to the zone.
- `direct`, which means use block ID seek capability to the zone if the block ID is known.
- `data`, which means the same as `direct` when the tape is being written. When the tape is being read, `data` means that the read child will try to determine the block ID of the data being read, and use the block ID seek capability to position there.

The default depends on the type of drive, and is either `direct` or `data`. If data positioning is specified for a drive whose default is `direct`, the block ID is calculated by adding an estimate of the number of blocks from the start of the zone to the data being recalled and the block ID of the start of the zone. Not all drives use this format for block ID.

POSITION_RETRY

Specifies the level of retry in the event of a failure during zone positioning. The values can be:

- `aggressive`, which means the VG can try more costly and time-consuming alternatives
- `lazy`, which means the VG retries if a reasonably fast alternative means of positioning is available (default)
- `none`, which means there will be no retry

If the VG is unable to position to a zone, all recalls for files with data in that zone are aborted by the VG (though not by DMF if a copy exists in another VG).

The default is `lazy`, to give the best overall recall time. If you are having trouble getting data from tape, you might want to try `aggressive`.

READ_ERR_MAXIMUM

Specifies the maximum number of I/O errors that will be tolerated when reading from tape to recall a file. The

	legal range of values is 2 through 100000. The default is 5000. The value of READ_ERR_MAXIMUM should be greater than the value of READ_ERR_MINIMUM. See the description of READ_ERR_TIMEOUT.
READ_ERR_MINIMUM	Specifies the minimum number of I/O errors that will be tolerated when reading from tape to recall a file. The legal range of values is 1—100000. The default is 10. See the description of READ_ERR_TIMEOUT.
READ_ERR_TIMEOUT	Specifies the number of seconds that can elapse since the first I/O error was seen when reading from tape to recall a file. READ_ERR_TIMEOUT, READ_ERR_MINIMUM, and READ_ERR_MAXIMUM together determine how many I/O errors will be tolerated when reading from tape to recall a file. If the number of consecutive I/O errors is greater than READ_ERR_MAXIMUM, or if the number of consecutive I/O errors is greater than READ_ERR_MINIMUM and the elapsed number of seconds since the first error was seen is greater than READ_ERR_TIMEOUT, the recall will fail. The legal values for READ_ERR_TIMEOUT are 300 through 3600 seconds. The default is 600 seconds.
READ_IDLE_DELAY	Specifies the number of seconds an idle tape LS read child (dmatrc) can wait before being told to exit. If other DMF requests are waiting for a tape drive, the read child may be told to exit before READ_IDLE_DELAY seconds have passed. The default is 5 seconds.
REINSTATE_DRIVE_DELAY	Specifies the number of minutes after which a drive that was configured down by the drive group will be automatically reinstated and made available for use again. A value of 0 means it should be left disabled indefinitely. The default is 1440 (one day).
REINSTATE_VOLUME_DELAY	Specifies the number of minutes after which a volume that had its HLOCK flag set by DMF will be automatically reinstated and made available for use again. A value of 0 means they should be left disabled indefinitely. The default is 1440 (one day).

REWIND_DELAY	Specifies the number of seconds an idle tape LS read child (dmatrc) can wait before rewinding. If other DMF requests are waiting for a tape drive, the read child may rewind before READ_IDLE_DELAY seconds have passed. The maximum value that you can specify is the value of READ_IDLE_DELAY. If READ_IDLE_DELAY is not specified, the maximum value that you can specify is the default value of READ_IDLE_DELAY. The default is the minimum of {2, READ_IDLE_DELAY/2}. If an idle read child must rewind the tape before the drive can be used to service other DMF requests, that will delay the servicing of those requests; therefore you should use caution when increasing this parameter.
RUN_TASK	See the description of RUN_TASK in "taskgroup Parameters" on page 174. Also see "Automated Maintenance Tasks" on page 89.
TASK_GROUPS	Names the task groups that contain tasks the drive group should run. They are configured as taskgroup objects. There is no default.
TMF_TMMNT_OPTIONS (TMF MOUNT_SERVICE only)	Specifies command options that should be added to the tmmnt command when mounting a tape. DMF uses the -Z option to tmmnt to ignore options controlling block size and label parameters. Use the BLOCK_SIZE and LABEL_TYPE drive group parameters instead. There is no need for a -g option here. If it is provided, it must match the value of the MOUNT_SERVICE_GROUP parameter. To request compression, use -i. Options that are ignored are -a, -b, -c, -D, -f, -F, -l, -L, -n, -o, -O, -p, -P, -q, -R, -t, -T, -U, -v, -V, -w, -x, and -X.
VERIFY_POSITION	Specifies whether the LS write child should (prior to writing) verify that the tape is correctly positioned and that the tape was properly terminated by the last use. The default is to verify. Specifying no or off turns verification off; anything else ensures verification.
VOLUME_GROUPS	Names the VGs containing volumes that can be mounted on any of the drives within this drive group.

They are configured as volumegroup objects. This parameter must be configured. There is no default.

The order of the values specified for this parameter is integral to the determination of the MSP/VG from which the DMF daemon attempts to recall an offline file. If the offline file has more than one copy, DMF uses a specific order when it attempts to recall the file. It searches for a good copy of the offline file in MSP or LS order, from the dmdaemon object's MSP_NAMES or LS_NAMES parameter. If one of those names refers to an LS, it searches for the copy in drive group order, from the libraryserver object's DRIVE_GROUPS parameter. It then searches for the copy in VG order from the drivegroup object's VOLUME_GROUPS parameter.

Note: Do not change this parameter while DMF is running.

WRITE_CHECKSUM — Specifies that tape block should be checksummed before writing. If a tape block has a checksum, it is verified when read. The default is on.

volumegroup Object

There must be a volumegroup object for each pool of tape volumes of the same type. It must be usable on the drives of the associated drive group and capable of holding at most one copy of user files. A volumegroup object has the following parameters:

Parameter	Description
TYPE	volumegroup (required name for this type of object). There is no default.
ALLOCATION_GROUP	Specifies the allocation group that serves as a source of additional volumes if a VG runs out of media. Normally, one allocation group is configured to serve multiple VGs. As a volume's hfree flag is cleared (see HFREE_TIME below) in a VG, it is immediately returned to the allocation group subject to the

restrictions imposed by the configuration parameters ALLOCATION_MAXIMUM and ALLOCATION_MINIMUM. The administrator must ensure that volumes in the allocation group are mountable on drives in the same drive group as any VG that references the allocation group. It is an error to assign an ALLOCATION_GROUP name that is the same as an existing VG name. The ALLOCATION_GROUP defines a logical pool of volumes rather than an actual operational VG.

Allocation groups have no configurable parameters or configuration stanzas of their own; a reference to them in a VG's ALLOCATION_GROUP parameter is all that is needed to activate them. A VG that does not define the ALLOCATION_GROUP option will not use an allocation group.

ALLOCATION_MAXIMUM — Specifies the maximum size in number of volumes to which a VG can grow by borrowing volumes from its allocation group. The minimum value is 0 and the maximum and default are infinity. (That is, the default is that there is no maximum; the VG can keep borrowing from the allocation group until the allocation group runs out.) If the VG already contains ALLOCATION_MAXIMUM or more volumes, no additional volumes are borrowed from the allocation group. If no allocation group is defined, this parameter is meaningless.

ALLOCATION_MINIMUM — Specifies the minimum size in number of volumes to which a VG can shrink by returning volumes to its allocation group. The minimum value is 0, which is the default, and the maximum is the value of ALLOCATION_MAXIMUM. If the VG already contains ALLOCATION_MINIMUM or fewer volumes, no additional volumes are returned to the allocation group. If no allocation group is defined, this parameter is meaningless.

DRIVE_MAXIMUM — Specifies the maximum number of drives within this drive group that this VG is allowed to use simultaneously. The value actually used is the least of the drivegroup object's DRIVE_MAXIMUM, this

volumegroup object's DRIVE_MAXIMUM, and the number of drives that the drive group can physically detect. The maximum is 100; the default is the drivegroup object's DRIVE_MAXIMUM.

HFREE_TIME

Specifies the minimum number of seconds that a tape no longer containing valid data must remain unused before the VG overwrites it. The default value is 172800 seconds (2 days) and the minimum allowed value is 0.

When an LS removes all data from a tape, it sets the hfree (hold free tape) flag bit in the tape's VOL record in the LS database to prevent that tape from being immediately reused. The next time that the LS scans the database for volumes after HFREE_TIME seconds have passed, the LS clears the hfree flag, allowing the tape to be rewritten. If HFREE_TIME is set to 0, the LS will never clear hfree, so an unused tape will not be reused until you clear its hfree flag manually. For a description of how to set and clear the hfree flag manually, see the dmvoladm man page.

MAX_CHUNK_SIZE

Specifies that the VG should break up large files into chunks no larger than this value (specified in bytes) as it writes data to tape. If a file is larger than this size, it is broken up into pieces of the specified size, and, depending on other activity, more than one write child may be used to write the data to tape. If MAX_CHUNK_SIZE is 0 (the default), the VG breaks a file into chunks only when an end of volume is reached.

MAX_PUT_CHILDREN

Specifies the maximum number of write child (dmatwc) processes that will be simultaneously scheduled for the VG. The maximum value is the value of DRIVE_MAXIMUM for the VG's owning drive group. The minimum value is 1. The default is the same as the value that the volumegroup object uses for DRIVE_MAXIMUM; if the value specified in the configuration file exceeds this default, the default is used.

MERGE_CUTOFF	Specifies a limit at which the VG will stop scheduling tapes for merging. This number refers to the sum of the active and queued children generated from gets, puts, and merges. The default value for this option is the value used by the volumegroup object for DRIVE_MAXIMUM. This means that if sparse tapes are available, the VG will create DRIVE_MAXIMUM number of children, thus using tape resources efficiently. However, if any recall requests arrive for that VG, they will be started before new merges. Setting this number below DRIVE_MAXIMUM, in effect, reserves some tape units for recalls at the expense of merge efficiency. Setting this number above DRIVE_MAXIMUM increases the priority of merges relative to recalls. The minimum value that you can specify is 2.
MIN_VOLUMES	Specifies the minimum number of unused volumes that can exist in the LS database for this VG without operator notification. If the number of unused volumes falls below MIN_VOLUMES, the operator is asked to add new volumes. The default is 10; the minimum is 0. If a VG has an allocation group configured, MIN_VOLUMES is applied to the sum of the number of unused volumes in the VG and in its allocation group subject to any ALLOCATION_MAXIMUM restrictions.
PUTS_TIME	Specifies the minimum number of seconds a VG waits after it has requested a drive for a write child before it tells a lower priority child to go away. The default is 3600 seconds.
READ_TIME	Specifies the interval, in seconds, after which the VG will evaluate whether a read child should be asked to go away (even if it is in the middle of recalling a file) so that a higher priority child can be started. If READ_TIME is 0, the VG will not do this evaluation. The default is 0.
RUN_TASK	See the description of RUN_TASK in "taskgroup Parameters" on page 174. Also see "Automated Maintenance Tasks" on page 89.
TASK_GROUPS	Names the taskgroup objects that contain tasks the VG should run. There is no default.

`TIMEOUT_FLUSH`	Specifies the number of minutes after which the VG will flush files to tape. The default is `120` minutes.

`ZONE_SIZE`

Specifies approximately how much data the write child should put in a zone. The write child adds files and chunks to a zone until the data written equals or exceeds this value, at which time it writes a tape mark and updates the data base. The VG also uses zone size to determine when to start write children, and the number of write children to start. By default, the unit of measure is bytes, but you can specify other units of measure as follows: `m` for megabytes, `g` for gigabytes. The default is `50000000` bytes. For more information about zone size, also see "Media Concepts" on page 303.

Note: It is critical that the zone size is appropriate for the tape speed and average data compression rate at your site. A value that is too small can cause poor write performance because a tape mark is written at the end of each zone; a value that is too large can reduce parallelism when migrating files. See "Improve Tape Drive Performance with an Appropriate Zone Size" on page 70.

The zone size influences the required cache space. The value for the `CACHE_SPACE` parameter should be at least twice the value used for `ZONE_SIZE`. Increasing the `ZONE_SIZE` value without also increasing `CACHE_SPACE` could cause tape merging to become inefficient. Tape merges could have problems if the `ZONE_SIZE` value is larger than the `CACHE_SPACE` value. For more information about `CACHE_SPACE`, see "`libraryserver` Object" on page 215.

`resourcescheduler` Object

The entries for a `resourcescheduler` object, one for each drive group in a single library, has the following parameters:

Parameter	Description
`TYPE`	`resourcescheduler` (required name for this type of object). There is no default.
`ALGORITHM`	Specifies the resource scheduling algorithm to be used. Two are supplied: a simple one called `fifo` ("first-in, first out"), and a more flexible one called `weighted_roundrobin` (default).

Other parameters are specific to a particular resource scheduling algorithm. There are no parameters for `fifo`. For `weighted_roundrobin`, the following apply:

Parameter	Description
`PENALTY`	Reduces the priority of requests from a VG that is not the next one preferred by the round-robin algorithm. It is a multiplier in the range `0.0`—`1.0`. Low values result in the urgency assigned by the VG being totally or partially ignored, and high values mean that the urgency is more important than selecting one whose turn ought to be next. The default is `0.7`.
`WEIGHT`	Assigns a weighting to one or more VGs. The ratio of these weightings to each other (within the one drive group) determines the number of opportunities the VG has to obtain drives when they are needed. The weightings are integers in the range `1`—`99` and they need not be unique. For efficiency reasons, small numbers are preferred, especially if large numbers of VGs are defined. Usually, there are multiple `WEIGHT` lines in the configuration, and a given VG might appear on more than one of them. In such cases, the sum of the weights is used as the effective weight for that VG. Any VGs that do not appear on a `WEIGHT` line are assigned the default of `5`. If there are no `WEIGHT` lines, all VGs will use this default, resulting in a strict round-robin behavior.

`WEIGHT` has the following format:

`WEIGHT` *weight vg1 vg2 ...*

`resourcewatcher` Object

The entry for a `resourcewatcher` object is needed only if you wish to change its parameter defaults; a reference to a resource watcher by the `libraryserver` object is sufficient to activate it.

The `resourcewatcher` object has the following parameters:

Parameter	Description
`TYPE`	`resourcewatcher` (required name for this type of object). There is no default.
`HTML_REFRESH`	Specifies the refresh rate (in seconds) of the generated HTML pages. The default is 60.

Example of Configuring an LS

Example 7-17 defines an LS containing a default resource watcher and two drive groups:

- Drive group `dg1` contains two VGs (the primary VG `vg_prim` and the secondary VG `vg_sec`) sharing an allocation group. A resource scheduler is defined to give primary `vg_prim` twice the priority of secondary `vg_sec` when competing for drives. The `volumegroup` objects are slightly different, reflecting that the secondary VG is usually write-only.
- Drive group `dg2` contains a single VG, `vg_ul4`.

Note: Example 7-17 does not use all of the possible options for configuring a `libraryserver` object.

Example 7-17 `libraryserver` Object

```
define  ls1
                TYPE                    libraryserver
                COMMAND                 dmatls
                DRIVE_GROUPS            dg1 dg2
                CACHE_SPACE             500m
                TASK_GROUPS             ls_tasks
                WATCHER                 rw
enddef

define  dg1
                TYPE                    drivegroup
                VOLUME_GROUPS           vg_prim vg_sec
                MOUNT_SERVICE           openvault
                MOUNT_SERVICE_GROUP     ultrium3grp
                OV_INTERCHANGE_MODES    compression
                DRIVE_SCHEDULER         rs
                DRIVES_TO_DOWN          2
                REINSTATE_DRIVE_DELAY   60
enddef

define  dg2
                TYPE                    drivegroup
                VOLUME_GROUPS           vg_ul4
                MOUNT_SERVICE           openvault
                MOUNT_SERVICE_GROUP     ultrium4grp
                OV_INTERCHANGE_MODES    compression
                DRIVES_TO_DOWN          2
                REINSTATE_DRIVE_DELAY   60
enddef

define  rs
                TYPE                    resourcescheduler
                WEIGHT                  10      vg_prim
                WEIGHT                  5       vg_sec
enddef

define  vg_prim
                TYPE                    volumegroup
                ALLOCATION_GROUP        ag_ult3
```

```
enddef

define  vg_sec
                TYPE                    volumegroup
                ALLOCATION_GROUP        ag_ult3
                DRIVE_MAXIMUM           2
enddef

define vg_ul4
                TYPE                    volumegroup
enddef
```

The steps in Procedure 7-3 explain pertinent information for configuring each of the `libraryserver` objects in Example 7-17.

Procedure 7-3 Configuring an LS and Its Components

1. Ensure that `define` parameter has a value that you set previously in the `LS_NAMES` or `MSP_NAMES` parameter of the `dmdaemon` object. There is no default.
2. Set `TYPE` to `libraryserver`.
3. Set `COMMAND` is set to `dmatls`.
4. Specify a `DRIVE_GROUPS` parameter that names a collection of interchangeable tape drives. In this example, there are two such groups. There is no default.
5. Set the `CACHE_SPACE` parameter to tell the LS how much disk space it can use when merging chunks from sparse tapes. The LS can merge tapes more efficiently if it can stage most of the files to disk. The default for `CACHE_SPACE` is `0`, which causes all data to be transferred by sockets.

   The tape zone size influences the required cache space; you should configure the `CACHE_SPACE` parameter to be at least twice the value used for `ZONE_SIZE`. For more information on zone sizes, see "`volumegroup` Object" on page 224.
6. Configure the `TASK_GROUPS` parameter to the names of the objects used to define how periodic maintenance tasks are completed. There is no default. For more information, see "LS Tasks" on page 240.
7. To observe LS operation through a web browser, specify a resource watcher by defining the `WATCHER` parameter. You need only a reference to the resource watcher's name.

Note: Define a `resourcewatcher` object only if you want to change its default parameters. See "`resourcewatcher` Object" on page 231.

Assuming that `SPOOL_DIR` was set in the base object to be `/dmf/spool`, the URL to use for this example is `file://dmf/spool/ls/_rw/ls.html`. Text files are generated in the same directory as the HTML files.

8. Define the drive groups referenced in step 4. There is no `COMMAND` line; a drive group is not an independent program, but a component of an LS.
9. Define the VGs using the drives managed by each drive group with the `VOLUME_GROUPS` parameter.
10. Specify the use of OpenVault. Because OpenVault is the default mounting service, this line can be omitted.
11. Specify the name that the mounting service uses to refer to this group of drives. When using OpenVault, the `MOUNT_SERVICE_GROUP` line specifies the OpenVault drive group to be used.

Note: OpenVault uses the same term as does DMF to describe a group of interchangeable tape devices, but the two uses are separate. Their names need not match, though it may be less confusing if they do.

If using TMF, the `MOUNT_SERVICE_GROUP` line names the TMF device group name.

12. Use the `OV_INTERCHANGE_MODES` and `TMF_TMMNT_OPTIONS` lines to specify that the drives (OpenVault and TMF, respectively) should be used in compression mode.
13. For drive group `dg1`, override the default resource scheduler behavior by referring to an object called `rs`, to be defined in step 15 below.
14. Allow each drive group to configure at most two drives down temporarily for 60 minutes for recovery from I/O errors if the drives are faulty and if doing so will result in a more reliable operation. When this happens, the administrator is e-mailed so that maintenance can be performed.
15. In the `rs` object, specify that when there are more requests for tape drives than there are drives in the drive group, VG `vg_prim` is to be given access twice as

often as vg_sec. The ratio of the numbers is important, but the exact values are not.

16. Define the VGs. The VOLUME_GROUPS parameter of the drivegroup object and the SELECT_LS or SELECT_MSP lines in the filesystem objects refer to them.

17. Define an allocation group for tape type Ultrium 3 called ag_ult3. Allocation groups have no configurable parameters, so they have no defining object; just a reference is sufficient. Use of an allocation group is optional. In this example, no allocation group is defined for tape type Ultrium 4.

18. Include any other VG parameters that you require. For example, the definition of the vg_sec object specifies that it can use at most two tape drives, so that other drives in the dg1 drive group will be immediately available for use by vg_prim when it needs them.

OpenVault and LS Drive Groups

Procedure 7-4 describes the steps you must take to configure OpenVault for a drive group.

Procedure 7-4 Configuring DMF to Use OpenVault

The following procedure describes how to make OpenVault and DMF work together using ov_admin.

Note: The procedure that follows assumes that before you complete the steps described, the OpenVault server is configured and all drives and libraries are configured and OpenVault is running. SGI recommends that the OpenVault server be the same as the DMF server.

---

1. On the OpenVault server, add DMF as an OpenVault application. Use ov_admin and select the menu option that allows you to manage applications. Create the DMF application with a name of dmf (in lowercase).

2. Grant the DMF application access to any cartridge groups that DMF will use. (This can also be done at a later time.) See step 5 for more information.

3. Grant the DMF application access to any drive groups DMF will use. (This can also be done at a later time.) See step 6 for more information.

4. Add both a privileged and unprivileged application instance for the DMF application. Enter a wildcard ("*") for the host and application's instance name as shown here:

```
Enter the name of the Host where an instance of Application "dmf" will run: *

Enter the Application's instance name or "*": *
```

   To prevent the application from being used from unauthorized hosts, enter a security key when prompted.

5. Add DMF as a valid application to appropriate cartridge groups.

   The `ov_admin` script allows you to specify the cartridge groups when the DMF application is created or, after creation of the DMF application, you can choose the menu option that allows you to manage cartridge groups.

   If for some reason you cannot use the `ov_admin` script, you can enter the command manually, as follows:

   `ov_cartgroup -a -G` *tape_group* `-A dmf`

6. Add the DMF application as a valid user to appropriate OpenVault drive groups. The OpenVault drive groups that DMF uses must contain only fungible drives. That is, the drives in the OpenVault drive group must have identical characteristics and accessibility, so that any volume that can be mounted and written on one of the drives can also be mounted and read on any of the other drives within the group. Failure to provide identical mounting and accessibility characteristics to all drives in an OpenVault drive group used by an LS might result in tape mount failures.

   Choose the appropriate item from the `ov_admin` menu. If for some reason you cannot use the `ov_admin` script, you can enter the command manually, as follows:

   `ov_drivegroup -a -G` *drive_group* `-A dmf`

7. Configure the base object for use with OpenVault:

```
define  base
        TYPE            base
        HOME_DIR        /dmf/home
.
.
.
        OV_KEY_FILE     /dmf/home/ovkeys
        OV_SERVER       hostname_ONLY_if_different_from_DMF_server
enddef
```

   a. Configure the OV_KEY_FILE parameter name of the key file that holds security information for OpenVault. It is usually located in *HOME_DIR* and called ovkeys. When using the Parallel Data Mover Option, this file must be visible to the DMF server and all parallel data mover nodes, therefore it must be in a CXFS filesystem.

   b. Configure OV_SERVER only if the OpenVault server is not running on the same node as the DMF server.

8. Use the dmov_keyfile (8)command to create the file defined by the OV_KEY_FILE parameter. This command will prompt you for the privileged and unprivileged keys that you defined in step 4.

9. Configure the LS's drivegroup object for use with OpenVault. In the drivegroup object, use the following steps:

   a. Configure the MOUNT_SERVICE parameter to be openvault.

   b. Configure the MOUNT_SERVICE_GROUP parameter to be the name of the OpenVault drive group, as seen in the output from the ov_stat -d command.

   c. Configure the OV_INTERCHANGE_MODES parameter to be a list of interchange mode names that control how data is written to tape. You can use this to control whether the device compresses data as it is written. This parameter is optional.

      To specify that you want data compressed, use:

      ```
      OV_INTERCHANGE_MODES     compression
      ```

To force all tapes to be written as DLT4000, use:

```
OV_INTERCHANGE_MODES      DLT4000
```

This parameter is applied when a tape is first used or rewritten.

10. Make the appropriate cartridges accessible to the allocation groups, VGs, or filesystem backup scripts by assigning the cartridges to the DMF application in OpenVault. Do the following:

    - To find out which drives are in each drive group:

```
# ov_dumptable -n -d'|' -c DriveGroupName,DriveName,LibraryName DRIVE
ultrium3grp|drive1|lib1
ultrium3grp|drive2|lib1
ultrium4grp|drive3|lib1
ultrium4grp|drive4|lib1
```

    - To find out which cartridge types each drive can mount:

```
# ov_dumptable -n -d'|' -c DriveName,CartridgeTypeName DCPCAPABILITY | sort -u
drive1|Ultrium1-100
drive1|Ultrium2-200
drive1|Ultrium3-400
drive2|Ultrium1-100
drive2|Ultrium2-200
drive2|Ultrium3-400
drive3|Ultrium2-200
drive3|Ultrium3-400
drive3|Ultrium4-800
drive4|Ultrium2-200
drive4|Ultrium3-400
drive4|Ultrium4-800
```

      In this example, any Ultrium4–800 cartridges can only be used in the `ultrium4grp` drive group.

    - To find out the possible cartridge groups:

      # **ov_cartgroup -s -A dmf**

    - Do one of the following to make both DMF and OpenVault aware of the cartridges to be mounted:

Caution: All cartridges that DMF mounts via OpenVault must have the correct cartridge type. Failure to correctly specify the cartridge type can result in errors when reading and writing data. Contact your SGI service representative if you have questions about cartridge type specification.

– If you already have tapes defined in your LS database or in a `DUMP_TAPES` file but OpenVault is not aware of them, and every cartridge in the given LS, VG, or task group is of the same cartridge type, you can tell OpenVault about these tapes by entering one of the following:

```
dmov_makecarts -g cartgroup -t carttype lsname
dmov_makecarts -g cartgroup -t carttype -v vg1,vg2 lsname
dmov_makecarts -g cartgroup -t carttype taskgroupname
```

You can replace any of the references to a VG previously mentioned with an allocation group. If the `-v` parameter is omitted, all VGs and allocation groups in the specified LS will be processed. Tapes will be added to the file controlling the `run_full_dump.sh` and `run_partial_dump.sh` scripts by specifying the name of the task group that refers to them.

– If you have tapes that neither DMF nor OpenVault is aware of, you can import them by cartridge type into OpenVault and add them to DMF by VG, allocation group, or task group by entering one of the following:

```
dmov_loadtapes -l library -g cartgroup -t carttype vgname
dmov_loadtapes -l library -g cartgroup -t carttype agname
dmov_loadtapes -l library -g cartgroup -t carttype taskgroupname
```

This command will invoke a `vi`(1) session. In the `vi`(1) session, delete any cartridges that you do **not** want added to the LS database. All cartridges that are left in the `vi` session file must be of the same cartridge type, the type you specified with the `-t` option. Tapes will be added to the file controlling the `run_full_dump.sh` and `run_partial_dump.sh` scripts by specifying the name of the task group which refers to them.

– If neither of the above cases apply, you can manually configure the cartridges. The following commands can be useful in this effort:

- Use `ov_stat` to list cartridges in a library. For example:

  ```
  ov_stat -s -L library
  ```

- Use `ov_lscarts` to list information on cartridges known to OpenVault. For example:

  ```
  ov_lscarts -f '.*'
  ```

- Use `ov_import` and `dmvoladm` to add the unmanaged cartridges to OpenVault and DMF, and use `vi` to edit the task group in the file specified by the `DUMP_TAPES` parameter in the `taskgroup` stanza in the `dmf.conf` file.

TMF and LS Drive Groups

Use one of the following `dmvoladm`(8) commands to add tapes to the LS database:

```
dmvoladm  -l lsname  -c 'create vsn001-vsn010 vg vgname'
dmvoladm  -l lsname  -c 'create vsn001-vsn010 vg agname'
```

An allocation group is specified by the `vg` option, just like a VG.

There is no special procedure to inform TMF of a tape's existence. TMF assumes that every tape it deals with is in the library or can be provided by an operator, as needed.

LS Tasks

You can configure parameters for how the LS daemon performs the following maintenance tasks:

- Merging sparse tapes with the `run_tape_merge.sh` task and the `THRESHOLD`, `VOLUME_LIMIT`, and `DATA_LIMIT` parameters
- Stopping volume merges at a specified time with the `run_merge_stop.sh` task

Table 7-1 on page 172 provides a summary of automated maintenance tasks.

For each of these tasks, you can configure when the task is run. For merging sparse tapes, you must provide more information such as what determines that a tape is sparse and how many tapes can be merged at one time.

Note: The `run_remove_journals.sh` and `run_remove_logs.sh` tasks are configured as part of the `taskgroup` object for daemon tasks, but these tasks also clear the MSP/LS logs and journals. These tasks are described in "`taskgroup` Object" on page 170.

The `run_daily_drive_report.sh`, `run_daily_tsreport.sh`, and `run_daily_report.sh` tasks should be configured as part of the `taskgroup` object for `dmdaemon` tasks. This is because there could be multiple LSs for which `run_daily_drive_report.sh` and `run_daily_tsreport.sh` create reports, and `run_daily_report.sh` reports on other things besides LS information (such as information about the DMF-managed filesystems).

The following example explains how to define a `taskgroup` object for LS tasks named `libraryserver_tasks`. (You can change `libraryserver_tasks` to be any name you like.) The example assumes that the LS using this task has only one VG. For information about volume merging when an LS has multiple VGs, see step 3 of Procedure 7-5, page 242.

Do not change the pathnames or task names.

You may comment-out the `RUN_TASK` parameters for any tasks you do not want to run.

Example 7-18 `taskgroup` Object for LS Tasks

```
define libraryserver_tasks
 TYPE           taskgroup
 RUN_TASK         $ADMINDIR/run_tape_merge.sh on \
                   monday wednesday friday at 2:00
  THRESHOLD        50
# VOLUME_LIMIT     20
# DATA_LIMIT       5g
  RUN_TASK         $ADMINDIR/run_merge_stop.sh at 5:00
```

Procedure 7-5 Configuring a `taskgroup` Object

1. Define the object to have the same name that you provided for the `TASK_GROUPS` parameter of the `libraryserver` object. In the example, it is named `libraryserver_tasks`.

2. Ensure that `TYPE` is set to `taskgroup`. There is no default.

3. Configure the `RUN_TASK` parameters. DMF substitutes `$ADMINDIR` in the path with the `/usr/lib/dmf` directory. When the task is run, it is given the name of the object that requested the task as the first parameter and the name of the task group (in this case `msp_tasks`) as the second parameter. The task itself may use the `dmconfig`(8) command to obtain further parameters from either of these objects.

   The `RUN_TASK` parameters require that you provide a *time_expression*. See the parameter description in "`taskgroup` Object" on page 170.

   The following steps specify the information you must provide for the tasks to run correctly:

   a. The `run_tape_merge.sh` task merges sparse tapes. Specify the criteria that DMF uses to determine that a tape is sparse, as follows:

      - Use the `THRESHOLD` parameter to set an integer percentage of active data on a tape. DMF will consider a tape to be sparse when it has less than this percentage of data that is still active.

      - Use the `VOLUME_LIMIT` parameter to set the maximum number of tape volumes that can be selected for merging at one time. In the example, this is commented out, so there is no maximum limit.

      - Use the `DATA_LIMIT` parameter to set the maximum amount of data (in bytes) that should be selected for merging at one time. In the example, this is commented out, so there is no maximum limit.

      For the LSs, you can configure volume merging as part of the `libraryserver` object's `TASK_GROUPS` parameter or as part of a `RUN_TASK` parameter in the VG object. If it is configured as part of the `libraryserver` object's `TASK_GROUPS` parameter, volumes from any of the VGs in that LS may be marked as sparse. This can lead to drive scheduling and cache usage conflicts. To avoid this problem, configure volume merging as part of the `volumegroup` object and ensure there is no overlap in the times that the various merge tasks run.

As this might become cumbersome when there are large numbers of VGs configured, an alternative has been provided to `run_tape_merge.sh`, called `run_merge_mgr.sh`. This script establishes the needs of the VGs for more tapes, using their `MIN_VOLUMES` parameters as a guide to expected requirements. The script processes the most urgent requests first, minimizing interference with the production workload. To use this script, perform the following steps:

i. Define a `taskgroup` object, which is referred to by the `drivegroup` object (not the `volumegroup` or `libraryserver` object).

ii. Specify a `RUN_TASK` parameter for `run_merge_mgr.sh` in the `taskgroup` object and (optionally) another for `run_merge_stop.sh` . You can also specify `MESSAGE_LEVEL`, `THRESHOLD`, `VOLUME_LIMIT`, and `DATA_LIMIT` parameters.

iii. Ensure that the `libraryserver` object that refers to this drive group has a `resourcewatcher` object defined via the `WATCHER` parameter.

iv. For each `volumegroup` object, confirm that the value of its `MIN_VOLUMES` parameter is realistic.

b. Use the `run_merge_stop.sh` task to shut down volume merging at a time you specify by using a *time_expression*. This task is an alternative to using the `VOLUME_LIMIT` and `DATA_LIMIT` parameters to stop merging at specified points. In the example, the limit parameters are commented out because `run_merge_stop.sh` is used to control volume merging.

LS Database Records

After you have added the LS information to the configuration file, use the `dmvoladm`(8) command with the `-m` option to create any missing directories with the proper labels and to create volume (VOL) and catalog (CAT) records in the LS database.

You can follow the steps in Procedure 7-6 for each LSs you have defined.

Caution: Each LS must have a unique set of volume serial numbers.

Procedure 7-6 Creating LS Database Records

The following procedure is shown as an example that assumes you have an LS called `ls1`. This LS contains a VG named `vg_pri`.

1. Enter the following command and it will respond as shown:

   ```
   % dmvoladm -m ls1
   dmvoladm: at rdm_open - created database libsrv_db
   adm: 1>
   ```

   The response is an informational message indicating that `dmvoladm` could not open an existing LS database, so it is creating a new and empty one. You should get this message the first time you use `dmvoladm` for an LS, but never again. The next line (`adm:1>`) is the prompt for `dmvoladm` directives.

2. Assume that you will use 200 tapes with standard labels `VA0001` through `VA0200`. After the prompt, enter the following directive:

   ```
   adm:1> create VA0001-VA0200 vg vg_pri
   ```

   Note: You are specifying the VG `vg_pri` for the tapes being added. It is also valid to specify an allocation group name instead of a VG name.

   After entering this directive, you will receive 200 messages, one for each entry created, beginning with the following:

   ```
   VSN VA0001 created.
   VSN VA0002 created.
   ```

3. List all of the tape VSNs in the newly created library:

   ```
   adm:2> list all
   ```

4. Complete setting up the LS:

   ```
   adm:3> quit
   ```

MSP Objects

This section discusses the following:

FTP msp Object

To enable a file transfer protocol (FTP) MSP, include a name for it on the MSP_NAMES or LS_NAMES parameter in the dmdaemon object and define an msp object for it in the DMF configuration file.

DMF has the capability to use an FTP MSP to convert a non-DMF fileserver to DMF with a minimal amount of down time for the switch over, and at a site-determined pace. Contact your customer service representative for information about technical assistance with fileserver conversion.

An FTP msp object has the following parameters (see also Procedure 7-8, page 253):

Parameter	Description
TYPE	msp (required name for this type of object). There is no default.
CHILD_MAXIMUM	Specifies the maximum number of child processes the MSP is allowed to fork. The default is 4; the maximum is 100.
COMMAND	Specifies the binary file to execute in order to initiate this MSP. For the FTP MSP, this value must be dmftpmsp.
FTP_ACCOUNT	Specifies the account ID to use when migrating files to the remote system.
FTP_COMMAND	Specifies additional commands to send to the remote system. There may be more than one instance of this parameter.
FTP_DIRECTORY	Specifies the directory to use on the remote system.

FTP_HOST	Specifies the Internet hostname of the remote machine on which files are to be stored.
FTP_PASSWORD	Specifies the file containing the password to use when migrating files to the remote system. This file must be owned by root and be only accessible by root.
FTP_PORT	Specifies the port number of the FTP server on the remote system. The default value is the value configured for ftp in the services file.
FTP_USER	Specifies the user name to use when migrating files to the remote system.
GUARANTEED_DELETES	Specifies the number of child processes that are guaranteed to be available for processing delete requests. If CHILD_MAXIMUM is nonzero, its value must be greater than the sum of GUARANTEED_DELETES and GUARANTEED_GETS. The default is 1.
GUARANTEED_GETS	Specifies the number of child processes that are guaranteed to be available for processing dmget(1) requests. If CHILD_MAXIMUM is nonzero, its value must be greater than the sum of GUARANTEED_DELETES and GUARANTEED_GETS. The default is 1.
IMPORT_DELETE	Specifies if the MSP should honor hard-delete requests from the DMF daemon. This parameter may be set to OFF, ON, NO or YES. The default is OFF. This parameter applies only if IMPORT_ONLY is set to on. Set IMPORT_DELETE to on if you wish files to be deleted on the destination system when hard deletes are processed.
IMPORT_ONLY	Specifies that the MSP is used for importing only. Set this parameter ON when the data is stored as a bit-for-bit copy of the file and needs to be available to DMF as part of a conversion. The MSP will not accept dmput(1) requests when this parameter is enabled. The MSP will, by default, ignore hard-delete requests when this parameter is enabled. When the DMF daemon recalls a file from an IMPORT_ONLY MSP, it makes the file a regular file

rather than a dual-state file, and it soft-deletes the MSP's copy of the file.

`MESSAGE_LEVEL` — Specifies the highest message level number that will be written to the MSP log. It must be an integer in the range `0`—`6`; the higher the number, the more messages written to the log file. The default is `2`. For more information on message levels, see Chapter 9, "Message Logs" on page 277.

`MVS_UNIT` — Defines the storage device type on an MVS™ system. This must be specified when the destination is an MVS system. Valid values are `3330`, `3350`, `3380`, and `3390`.

`NAME_FORMAT` — Specifies the strings that form a template to create names for files stored on remote machines in the *STORE_DIRECTORY*. For a list of possible strings, see Table 7-2.

The default is `%u/%b` (*username/bfid*). This default works well if the remote machine runs an operating system based on UNIX. The default may not work at all if the remote machine runs an operating system that is not based on UNIX or if a given user has a large number of files. The date- and time-related strings allow sites with very large numbers of files to spread them over a large number of directories, in order to minimize subsequent access times.

Using the `%b` specification will guarantee a unique filename.

The `NAME_FORMAT` must include `%b` or `%2`, `%3`, `%4` in some combination.

The default size allotted to the `NAME_FORMAT` value in the daemon database base record is 34 bytes. This is large enough to accommodate the default for `NAME_FORMAT` if the user name is 8 or fewer characters (the `%b` value is always 24 characters). If you choose a set of strings that will evaluate to a field that is larger than 34 bytes, you may want to consider increasing the size of this record; see "Daemon Database Record Length" on page 87.

TASK_GROUPS	Names the task groups that contain tasks the MSP should run. They are configured as taskgroup objects. There is no default.
WRITE_CHECKSUM	Specifies that the DMF MSP's copy of the file should be checksummed before writing. If the file has been checksummed, it is verified when read. The default is on.

The MSP checks the DMF configuration file just before it starts child processes. If the DMF configuration file changed, it is reread.

If CHILD_MAXIMUM is nonzero, its value must be greater than the sum of GUARANTEED_DELETES and GUARANTEED_GETS.

The parameters COMMAND, FTP_HOST, FTP_USER, FTP_PASSWORD, and FTP_DIRECTORY must be present.

The MVS_UNIT parameter affects only IBM machines; they are further described in the dmf.conf(5) man page.

Note: The MSP will not operate if the FTP_PASSWORD file is readable by anyone other than root.

Table 7-2 NAME_FORMAT Strings

String	Evaluates To
%1	First 32 bits of the bit-file identifier (BFID) in lowercase hexadecimal. This is always 8 pad characters (00000000).
%2	Second 32 bits of the BFID in lowercase hexadecimal.
%3	Third 32 bits of the BFID in lowercase hexadecimal.
%4	Fourth 32 bits of the BFID in lowercase hexadecimal.
%b	BFID in hexadecimal (least-significant 24 characters). This does not contain the 8 pad characters found in the 8 most-significant characters of the full BFID.
%u	User name of the file owner.
%U	User ID of the file owner.

String	Evaluates To
%g	Group name of the file.
%G	Group ID of the file.
%%	Literal % character.
%d	Current day of month (2 characters).
%H	Current hour (2 characters).
%m	Current month (2 digits).
%M	Current minute (2 digits).
%S	Current second (2 digits).
%y	Last 2 digits of the current year (such as 03 for 2003).

The following example defines an FTP MSP.

Example 7-19 msp Object for an FTP MSP

```
define  ftp
        TYPE                    msp
        COMMAND                 dmftpmsp
        FTP_HOST                fileserver
        FTP_USER                dmf
        FTP_ACCOUNT             dmf.disk
        FTP_PASSWORD            /dmf/ftp/password
        FTP_DIRECTORY           ftpmsp
        FTP_COMMAND             umask 022
enddef
```

Procedure 7-7 Configuring an ftp Object

The following steps explain pertinent information for configuring an ftp object that uses a NAME_FORMAT of %u/%b:

1. Ensure that define has a value that you set previously in the MSP_NAMES or LS_NAMES parameter of the dmdaemon object. There is no default.
2. Set TYPE to msp.
3. Set COMMAND to dmftpmsp.

4. Set FTP_USER to the user name to use on the remote FTP server during session initialization.
5. Set FTP_HOST to fileserver.
6. Set the FTP_ACCOUNT parameter (if necessary) to the account to use on the remote FTP server during session initialization. Most FTP servers do not need account information. When account information is required, its nature and format will be dictated by the remote machine and will vary from operating system to operating system. There is no default.
7. Set the FTP_PASSWORD parameter to the name of the file containing the password to be used on the remote FTP server during session initialization. This file must be owned by root and only be accessible by root. In the example, the password for the user dmf on fileserver is stored in the file /dmf/ftp/password. There is no default.
8. Set the FTP_DIRECTORY parameter to the directory into which files will be placed on the remote FTP server. There is no default.
9. If necessary, specify commands to the remote machine's FTP daemon. In the example, the umask for files created is set to 022 (removes write permission for group and other). There is no default.

Disk msp Object

Note: The parameters differ for a disk cache manager (DCM), which is a disk MSP configured for *n*–tier capability. See "Disk Cache Manager (DCM) msp Object" on page 254.

To enable a disk MSP, include a name for it on the MSP_NAMES or LS_NAMES parameter in the dmdaemon object and define an msp object for it in the DMF configuration file.

As with the FTP MSP, you can use a disk MSP to convert a non-DMF fileserver to DMF with a minimal amount of down time for the switch over, and at a site-determined pace. Contact your customer service representative for information about technical assistance with fileserver conversion.

A disk msp object has the following parameters:

Parameter	Description
TYPE	msp (required name for this type of object). There is no default.
CHILD_MAXIMUM	Specifies the maximum number of child processes the MSP is allowed to fork. The legal range is 0—100, the default is 4.
COMMAND	Specifies the binary file to execute in order to initiate this MSP. For the disk MSP, this value must be dmdskmsp.
DSK_BUFSIZE	Specifies the transfer size in bytes used when reading from and writing to files within the disk MSP's *STORE_DIRECTORY*. The value must be in the range 4096—16000000 (16 million). The default is 131072 when writing and 1000000 when reading.
FADV_SIZE_MSP	Specifies the size of files in the MSP's STORE_DIRECTORY for which posix_fadvise() will be called with advice POSIX_FADV_DONTNEED. If the file is larger than FADV_SIZE_MSP bytes, the call is made following migration to the MSP or following recall of the entire file. The minimum value is 0, which means that posix_fadvise() will always be called, and the maximum value is 9223372036854775807. The default is 10000000.
GUARANTEED_DELETES	Specifies the number of child processes that are guaranteed to be available for processing delete requests. The default is 1.
GUARANTEED_GETS	Specifies the number of child processes that are guaranteed to be available for processing dmget(1) requests. The default is 1.
IMPORT_DELETE	Applies only if IMPORT_ONLY is set to on. Set IMPORT_DELETE to on if you wish files to be deleted in *STORE_DIRECTORY* when hard deletes are processed.
IMPORT_ONLY	Specifies the MSP is used for importing only. Set this parameter on when the data is stored as a bit-for-bit copy of the file and needs to be available to DMF as part of a conversion. The MSP will not accept dmput(1) requests when this parameter is enabled. By default,

the MSP will ignore hard delete requests when this parameter is enabled.

MESSAGE_LEVEL — Specifies the highest message level number that will be written to the MSP log. It must be an integer in the range 0—6; the higher the number, the more messages written to the log file. The default is 2. For more information on message levels, see Chapter 9, "Message Logs" on page 277.

NAME_FORMAT — Specifies the strings that form a template to create names for files stored on remote machines in the *STORE_DIRECTORY*. For a list of possible strings, see Table 7-2 on page 248.

The default is %u/%b (*username/bfid*). This default works well if the remote machine runs an operating system based on UNIX. The default may not work at all if the remote machine runs an operating system that is not based on UNIX or if a given user has a large number of files. The date- and time-related strings allow sites with very large numbers of files to spread them over a large number of directories, in order to minimize subsequent access times.

Using the %b specification will guarantee a unique filename.

The NAME_FORMAT must include %b or %2, %3, %4 in some combination.

The default size allotted to the NAME_FORMAT value in the daemon database base record is 34 bytes. This is large enough to accommodate the default for NAME_FORMAT if the user name is 8 or fewer characters (the %b value is always 24 characters). If you choose a set of strings that will evaluate to a field that is larger than 34 bytes, you may want to consider increasing the size of this record; see "Daemon Database Record Length" on page 87.

STORE_DIRECTORY — Specifies the directory used to hold files for disk MSPs. In order to avoid data corruption in the event of a system crash, the mount point of this directory must be

mounted with the `dirsync` option. See the `mount`(8) man page for a description of how to set the `dirsync` option.

Note: In the calculation used when measuring an MSP's actual amount of data stored versus the amount allowed to be stored by the DMF license, if the `STORE_DIRECTORY` parameter defined for that MSP does not define the root directory of a filesystem, or if other subdirectories of that filesystem are used by other users or processes to store data, the amount of stored capacity being charged to that MSP may exceed the actual amount of data being managed by that MSP. See the `dmusage`(8) man page and "Determining DMF Capacity" on page 49.

`TASK_GROUPS`	Names the task groups that contain tasks the MSP should run. They are configured as `taskgroup` objects. There is no default.
`WRITE_CHECKSUM`	Specifies that the DMF MSP's copy of the file should be checksummed before writing. If the file has been checksummed, it is verified when read. The default is `on`.

The following example describes setting up a disk `msp` object:

Example 7-20 `msp` Object for a Disk MSP

```
define  dsk
        TYPE                    msp
        COMMAND                 dmdskmsp
        CHILD_MAXIMUM           8
        GUARANTEED_DELETES      3
        GUARANTEED_GETS         3
        STORE_DIRECTORY         /remote/dir
enddef
```

Procedure 7-8 Configuring a Disk `msp` Object

The following steps explain pertinent information for configuring a disk `msp` object named `dsk`:

1. Ensure that define has a value that you set previously in the MSP_NAMES or LS_NAMES parameter of the dmdaemon object.
2. Set TYPE to msp.
3. Set COMMAND to dmdskmsp.
4. Set the CHILD_MAXIMUM parameter to the maximum number of child processes you want this MSP to be able to fork. The default is 4. The example allows 8.
5. Set the GUARANTEED_DELETES parameter to the number of child processes that are guaranteed to be available for processing delete requests. The default is 1. The example allows 3.
6. Set the GUARANTEED_GETS parameter to the number of child processes that are guaranteed to be available for processing dmget requests. The default is 1. The example allows 3.
7. Set the STORE_DIRECTORY parameter to the directory where files will be stored. This parameter is required; there is no default.

Disk Cache Manager (DCM) msp Object

A disk cache manager (DCM) is a disk MSP that is configured for *n*–tier capability. To enable a DCM, include a name for it on the MSP_NAMES or LS_NAMES parameter in the dmdaemon object and define an msp object for it in the DMF configuration file.

Note: The parameters differ for a disk MSP that is not a DCM. See "Disk msp Object" on page 250.

As with the FTP MSP, you can use a DCM to convert a non-DMF fileserver to DMF with a minimal amount of down time for the switch over, and at a site-determined pace. Contact your customer service representative for information about technical assistance with fileserver conversion.

A DCM msp object has the following parameters:

Parameter	Description
TYPE	msp (required name for this type of object). There is no default.

BUFFERED_IO_SIZE	Specifies the size of I/O requests for buffered I/O when migrating files downward in the hierarchy from *STORE_DIRECTORY* of this DCM. The legal range of values is 4096—16777216. The default is 262144.
CHILD_MAXIMUM	Specifies the maximum number of child processes that the DCM is allowed to fork. The legal range is 0—100, the default is 4.
COMMAND	Specifies the binary file to execute in order to initiate this MSP. For the DCM, this value must be dmdskmsp.
DIRECT_IO_SIZE	Specifies the size of I/O requests for direct I/O when migrating files downward in the hierarchy from the *STORE_DIRECTORY* of this DCM. The legal range of values is 65536 through 18446744073709551615. The default depends on the filesystem, but will not exceed the value of DIRECT_IO_MAXIMUM_SIZE defined in the base object. For more information about direct I/O, see O_DIRECT in the open(2) man page.
DSK_BUFSIZE	Specifies the transfer size in bytes used when reading from and writing to files within the DCM *STORE_DIRECTORY*. The value must be in the range 4096—16000000 (16 million). The default is 131072 when writing and 1000000 when reading.
FADV_SIZE_MSP	Specifies the size of files in the MSP's STORE_DIRECTORY for which posix_fadvise() will be called with advice POSIX_FADV_DONTNEED. If the file is larger than FADV_SIZE_MSP bytes, the call is made following migration to the MSP or following recall of the entire file. The minimum value is 0, which means that posix_fadvise() will always be called, and the maximum value is 9223372036854775807. The default is 10000000.
GUARANTEED_DELETES	Specifies the number of child processes that are guaranteed to be available for processing delete requests. The default is 1.

GUARANTEED_GETS
: Specifies the number of child processes that are guaranteed to be available for processing `dmget`(1) requests. The default is `1`.

MESSAGE_LEVEL
: Specifies the highest message level number that will be written to the MSP log. It must be an integer in the range `0`—`6`; the higher the number, the more messages written to the log file. The default is `2`. For more information on message levels, see Chapter 9, "Message Logs" on page 277.

MIGRATION_LEVEL
: Specifies the level of migration service for the DCM, as in `filesystem` objects. Valid values are:

  - `none` (no flushing to a lower VG)
  - `user` (only requests from `dmmigrate` or a manually invoked `dmdskfree`)
  - `auto` (automated space management)

  The default is `auto`.

MIN_DIRECT_SIZE
: Determines whether direct or buffered I/O is used when migrating files downward in the hierarchy from the *STORE_DIRECTORY* of this DCM. If the number of bytes to be read is smaller than the value specified, buffered I/O is used, otherwise direct I/O is used. The legal range of values is `0` (direct I/O is always used) through `18446744073709551615` (direct I/O is never used). The default is `0`.

  Note: For real-time filesystems, this parameter is ignored.

  For more information about direct I/O, see `O_DIRECT` in the `open`(2) man page.

NAME_FORMAT
: Specifies the strings that form a template to create names for files stored on remote machines in the *STORE_DIRECTORY*. For a list of possible strings, see Table 7-2 on page 248.

The default is %u/%b (*username/bfid*). This default works well if the remote machine runs an operating system based on UNIX. The default may not work at all if the remote machine runs an operating system that is not based on UNIX or if a given user has a large number of files. The date- and time-related strings allow sites with very large numbers of files to spread them over a large number of directories, in order to minimize subsequent access times.

Using the %b specification will guarantee a unique filename.

The NAME_FORMAT must include %b or %2, %3, %4 in some combination.

The default size allotted to the NAME_FORMAT value in the daemon database base record is 34 bytes. This is large enough to accommodate the default for NAME_FORMAT if the user name is 8 or fewer characters (the %b value is always 24 characters). If you choose a set of strings that will evaluate to a field that is larger than 34 bytes, you may want to consider increasing the size of this record; see "Daemon Database Record Length" on page 87.

POLICIES — Specifies the names of the configuration objects defining policies for this filesystem. The configuration stanza must contain at least one POLICIES parameter and the configuration stanza for that parameter must contain a SELECT_LOWER_VG parameter.

PRIORITY_PERIOD — Specifies the number of minutes after which a migrating file gets special treatment.

Normally, if there is insufficient room in the *STORE_DIRECTORY* for a file, the DCM will attempt to make room, while continuing to store files that will fit. If a file has not been stored into the *STORE_DIRECTORY* within PRIORITY_PERIOD, however, the DCM will stop trying to store other files until either sufficient room has been made or it has determined that room cannot be made. This behavior

may change in the future. The legal range of values is 1—2000000; the default is 120 minutes (2 hours).

STORE_DIRECTORY — Specifies the directory used to hold files for DCMs, which must be the mount point of a dedicated XFS or CXFS filesystem mounted with DMAPI enabled. In order to avoid data corruption in the event of a system crash, this directory must be mounted with the dirsync option. See the mount(8) man page for a description of how to set the dirsync option.

In addition, when using the Parallel Data Mover Option, the directory must be a CXFS filesystem. See "Filesystem Mount Options" on page 85 for instructions.

Note: In the calculation used when measuring a DCM's actual amount of data stored versus the amount allowed to be stored by the DMF license, if the STORE_DIRECTORY parameter defined for that DCM does not define the root directory of a filesystem, or if other subdirectories of that filesystem are used by other users or processes to store data, the amount of stored capacity being charged to that DCM may exceed the actual amount of data being managed by that DCM. See the dmusage(8) command and "Determining DMF Capacity" on page 49

TASK_GROUPS — Names the task groups that contain tasks the DCM should run. They are configured as taskgroup objects. There is no default.

WRITE_CHECKSUM — Specifies that the DCM's copy of the file should be checksummed before writing. If the file has been checksummed, it is verified when read. The default is on.

To work with the DCM, the msp object requires the following:

Note: A DCM also requires a task group that runs the run_dcm_admin.sh script during off-peak hours to perform routine maintenance for the DCM.

The default size allotted to the NAME_FORMAT value in the daemon database base record is 34 bytes. This is large enough to accommodate the default for NAME_FORMAT if the user name is 8 or fewer characters (the %b value is always 24 characters). If you choose a set of strings that will evaluate to a field that is larger than 34 bytes, you may want to consider increasing the size of this record; see "Daemon Database Record Length" on page 87.

When using a DCM, dmdskmsp will no longer fail if the *STORE_DIRECTORY* is full. Instead, it will queue the requests and wait to fulfill them until after dmdskfree has freed the required space.

Following is a sample of the configuration stanzas with some explanatory notes below. Many of these parameters have defaults and can be omitted if they are appropriate.

Example 7-21 Configuration Stanzas Associated with a DCM

```
define  daemon
        TYPE                    dmdaemon
        LS_NAMES                dcm_msp ls                          # [See note 1]
        ...                                                         # [See note 2]
enddef

define  msp_policy
        TYPE                    policy
        SELECT_MSP              dcm_msp copy2 when space > 4096 # [See note 3]
        ...                                                         # [See note 2]
enddef

define  dcm_msp
        TYPE                    msp
        COMMAND                 dmdskmsp
        STORE_DIRECTORY         /dcm_cache                          # [See note 4]
        CHILD_MAXIMUM           10                                  # [See note 5]
        POLICIES                dcm_policy
        TASK_GROUPS             dcm_tasks
enddef

define  dcm_policy
        TYPE                    policy                              # [See note 6]

        FREE_SPACE_MINIMUM      10
        FREE_SPACE_TARGET       70
```

```
        DUALRESIDENCE_TARGET    90
        FREE_SPACE_DECREMENT    1
        FREE_DUALRESIDENT_FIRST on

        CACHE_AGE_WEIGHT           1       .1
        CACHE_SPACE_WEIGHT         1       .1

        SELECT_LOWER_VG         none when uid = 0
        SELECT_LOWER_VG         vg1 when space > 1G
        SELECT_LOWER_VG         vg2
enddef

define  dcm_tasks
        TYPE                    taskgroup
        RUN_TASK                $ADMINDIR/run_dcm_admin.sh at 22:00:10
enddef
```

Notes referred to in the preceding example:

1. The DCM must be specified before the LSs that contain its lower VGs. (Otherwise, all recalls will attempt to come directly from tape.)
2. Other parameters essential to the use of this stanza but not relevant to the DCM have been omitted.
3. The DCM and its lower VGs should be considered to act as a single high-speed VG logically maintaining only one copy of a migrated file. You should always have a second copy of all migrated files, which is the purpose of `copy2` in this example. It would probably be a tape VG, but could be any type of MSP other than a DCM.

   The copy that resides in the DCM *STORE_DIRECTORY* is not to be considered a permanent copy of the file in terms of the safety of the file's data. It can be deleted at any time, though never before a copy of it exists in one of the `SELECT_LOWER_VG` VGs.
4. The mount point of a **dedicated** DMAPI-mounted filesystem.
5. Any other parameters applicable to a disk MSP may also be used, with the exception of `IMPORT_ONLY` and `IMPORT_DELETE`.
6. Several parameters in DCM policies have functions that are analogous to those in disk MSP policies; see "Rules for `policy` Parameters" on page 195 and "User Filesystem `policy` Parameters" on page 196.

Summary of the Configuration File Parameters

Table 7-3 alphabetically lists the DMF configuration file parameters discussed in this chapter.

Note: For the complete list of parameters, see the `dmf.conf`(5) man page.

Table 7-3 DMF Configuration File Parameters

Parameter	Section Discussed In
ADMIN_EMAIL	"`base` Object" on page 152
AGE_WEIGHT	"File Weighting Parameters for a User Filesystem" on page 198
ALGORITHM	"`resourcescheduler` Object" on page 230
ALLOCATION_GROUP	"`volumegroup` Object" on page 224
ALLOCATION_MAXIMUM	"`volumegroup` Object" on page 224
ALLOCATION_MINIMUM	"`volumegroup` Object" on page 224
BANDWIDTH_MULTIPLIER	"`drivegroup` Object" on page 217
BLOCK_SIZE	"`drivegroup` Object" on page 217
BUFFERED_IO_SIZE	"`filesystem` Object" on page 187 "Disk Cache Manager (DCM) `msp` Object" on page 254
CACHE_AGE_WEIGHT	"File Weighting Parameters for a DCM `STORE_DIRECTORY`" on page 202
CACHE_DIR	"`libraryserver` Object" on page 215
CACHE_SPACE	"`libraryserver` Object" on page 215
CACHE_SPACE_WEIGHT	"File Weighting Parameters for a DCM `STORE_DIRECTORY`" on page 202
CHILD_MAXIMUM	"FTP `msp` Object" on page 245 "Disk `msp` Object" on page 250 "Disk Cache Manager (DCM) `msp` Object" on page 254

Parameter	Section Discussed In
COMMAND	"libraryserver Object" on page 215 "FTP msp Object" on page 245 "Disk msp Object" on page 250 "Disk Cache Manager (DCM) msp Object" on page 254
DATABASE_COPIES	"taskgroup Object" on page 170
DATA_LIMIT	"LS Tasks" on page 240
DIRECT_IO_MAXIMUM_SIZE	"base Object" on page 152
DIRECT_IO_SIZE	"filesystem Object" on page 187 "Disk Cache Manager (DCM) msp Object" on page 254
DISCONNECT_TIMEOUT	"libraryserver Object" on page 215
DRIVE_GROUPS	"libraryserver Object" on page 215
DRIVE_MAXIMUM	"drivegroup Object" on page 217 "volumegroup Object" on page 224
DRIVE_SCHEDULER	"drivegroup Object" on page 217 "volumegroup Object" on page 224
DRIVES_TO_DOWN	"drivegroup Object" on page 217
DSK_BUFSIZE	"Disk msp Object" on page 250 "Disk Cache Manager (DCM) msp Object" on page 254
DUALRESIDENCE_TARGET	"Automated Space Management Parameters for a DCM STORE_DIRECTORY" on page 201 "Disk Cache Manager (DCM) msp Object" on page 254
DUMP_DATABASE_COPY	"taskgroup Object" on page 170
DUMP_DEVICE	"taskgroup Object" on page 170
DUMP_FILE_SYSTEMS	"taskgroup Object" on page 170
DUMP_FLUSH_DCM_FIRST	"taskgroup Object" on page 170
DUMP_INVENTORY_COPY	"taskgroup Object" on page 170
DUMP_MAX_FILESPACE	"taskgroup Object" on page 170
DUMP_MIGRATE_FIRST	"taskgroup Object" on page 170
DUMP_RETENTION	"taskgroup Object" on page 170
DUMP_TAPES	"taskgroup Object" on page 170

Parameter	Section Discussed In
DUMP_VSNS_USED	"taskgroup Object" on page 170
DUMP_XFSDUMP_PARAMS	"taskgroup Object" on page 170
EXPORT_METRICS	"base Object" on page 152
EXPORT_QUEUE	"dmdaemon Object" on page 160
FADV_SIZE_MSP	"Disk msp Object" on page 250 "Disk Cache Manager (DCM) msp Object" on page 254
FREE_DUALRESIDENT_FIRST	"Automated Space Management Parameters for a DCM STORE_DIRECTORY" on page 201
FREE_DUALSTATE_FIRST	"Automated Space Management Parameters for a User Filesystem" on page 197
FREE_SPACE_DECREMENT	"Automated Space Management Parameters for a User Filesystem" on page 197 "Automated Space Management Parameters for a DCM STORE_DIRECTORY" on page 201
FREE_SPACE_MINIMUM	"Automated Space Management Parameters for a User Filesystem" on page 197 "Automated Space Management Parameters for a DCM STORE_DIRECTORY" on page 201
FREE_SPACE_TARGET	"Automated Space Management Parameters for a User Filesystem" on page 197 "Automated Space Management Parameters for a DCM STORE_DIRECTORY" on page 201
FTP_ACCOUNT	"FTP msp Object" on page 245
FTP_COMMAND	"FTP msp Object" on page 245
FTP_DIRECTORY	"FTP msp Object" on page 245
FTP_HOST	"FTP msp Object" on page 245
FTP_PASSWORD	"FTP msp Object" on page 245
FTP_PORT	"FTP msp Object" on page 245
FTP_USER	"FTP msp Object" on page 245

Parameter	Section Discussed In
GUARANTEED_DELETES	"FTP msp Object" on page 245 "Disk msp Object" on page 250 "Disk Cache Manager (DCM) msp Object" on page 254
GUARANTEED_GETS	"FTP msp Object" on page 245 "Disk msp Object" on page 250 "Disk Cache Manager (DCM) msp Object" on page 254
HA_VIRTUAL_HOSTNAME	"node Object" on page 163
HBA_BANDWIDTH	"base Object" on page 152 "node Object" on page 163
HFREE_TIME	"volumegroup Object" on page 224
HOME_DIR	"base Object" on page 152
HTML_REFRESH	"resourcewatcher Object" on page 231
IMPORT_DELETE	"FTP msp Object" on page 245 "Disk msp Object" on page 250
IMPORT_ONLY	"FTP msp Object" on page 245 "Disk msp Object" on page 250
JOURNAL_DIR	"base Object" on page 152
JOURNAL_RETENTION	"taskgroup Object" on page 170
JOURNAL_SIZE	"base Object" on page 152
LABEL_TYPE	"drivegroup Object" on page 217
LICENSE_FILE	"base Object" on page 152
LOG_RETENTION	"taskgroup Object" on page 170
LS_NAMES	"dmdaemon Object" on page 160
MAX_CACHE_FILE	"libraryserver Object" on page 215
MAX_CHUNK_SIZE	"volumegroup Object" on page 224
MAX_MANAGED_REGIONS	"filesystem Object" on page 187
MAX_MS_RESTARTS	"drivegroup Object" on page 217
MAX_PUT_CHILDREN	"volumegroup Object" on page 224
MERGE_CUTOFF	"volumegroup Object" on page 224

Parameter	Section Discussed In
MESSAGE_LEVEL	"dmdaemon Object" on page 160 "services Object" on page 167 "filesystem Object" on page 187 "libraryserver Object" on page 215 "FTP msp Object" on page 245 "Disk msp Object" on page 250 "Disk Cache Manager (DCM) msp Object" on page 254 Chapter 9, "Message Logs" on page 277
MIGRATION_LEVEL	"dmdaemon Object" on page 160 "filesystem Object" on page 187 "FTP msp Object" on page 245 "Disk msp Object" on page 250 "Disk Cache Manager (DCM) msp Object" on page 254
MIGRATION_TARGET	"Automated Space Management Parameters for a User Filesystem" on page 197
MIN_ARCHIVE_SIZE	"filesystem Object" on page 187
MIN_DIRECT_SIZE	"filesystem Object" on page 187 "Disk Cache Manager (DCM) msp Object" on page 254
MIN_VOLUMES	"volumegroup Object" on page 224
MOUNT_SERVICE	"device Object" on page 186 "drivegroup Object" on page 217
MOUNT_SERVICE_GROUP	"device Object" on page 186 "drivegroup Object" on page 217
MOUNT_TIMEOUT	"drivegroup Object" on page 217
MOVE_FS	"dmdaemon Object" on page 160
MSG_DELAY	"drivegroup Object" on page 217
MSP_NAMES	"dmdaemon Object" on page 160
MVS_UNIT	"FTP msp Object" on page 245
NAME_FORMAT	"FTP msp Object" on page 245 "Disk msp Object" on page 250 "Disk Cache Manager (DCM) msp Object" on page 254
NODE_ANNOUNCE_RATE	"services Object" on page 167

Parameter	Section Discussed In
NODE_BANDWIDTH	"base Object" on page 152 "node Object" on page 163
NODE_TIMEOUT	"services Object" on page 167
OV_ACCESS_NODES	"device Object" on page 186 "drivegroup Object" on page 217
OV_INTERCHANGE_MODES	"device Object" on page 186 "drivegroup Object" on page 217
OV_KEY_FILE	"base Object" on page 152
OV_SERVER	"base Object" on page 152
PARTIAL_STATE_FILES	"dmdaemon Object" on page 160
PENALTY	"resourcescheduler Object" on page 230
POLICIES	"filesystem Object" on page 187 "Disk Cache Manager (DCM) msp Object" on page 254
POSITIONING	"drivegroup Object" on page 217
POSITION_RETRY	"drivegroup Object" on page 217
POSIX_FADVISE_SIZE	"filesystem Object" on page 187
PRIORITY_PERIOD	"Disk Cache Manager (DCM) msp Object" on page 254
PUTS_TIME	"volumegroup Object" on page 224
READ_ERR_MAXIMUM	"drivegroup Object" on page 217
READ_ERR_MINIMUM	"drivegroup Object" on page 217
READ_ERR_TIMEOUT	"drivegroup Object" on page 217
READ_IDLE_DELAY	"drivegroup Object" on page 217
READ_TIME	"volumegroup Object" on page 224
RECALL_NOTIFICATION_RATE	"dmdaemon Object" on page 160
REINSTATE_DRIVE_DELAY	"drivegroup Object" on page 217
REINSTATE_VOLUME_DELAY	"drivegroup Object" on page 217
REWIND_DELAY	"drivegroup Object" on page 217

Parameter	Section Discussed In
RUN_TASK	"Automated Maintenance Tasks" on page 89 "taskgroup Object" on page 170 "libraryserver Object" on page 215 "drivegroup Object" on page 217 "volumegroup Object" on page 224
SCAN_FAST	"taskgroup Object" on page 170
SCAN_FILESYSTEMS	"taskgroup Object" on page 170
SCAN_FOR_DMSTAT	"taskgroup Object" on page 170
SCAN_OUTPUT	"taskgroup Object" on page 170
SCAN_PARALLEL	"taskgroup Object" on page 170
SCAN_PARAMS	"taskgroup Object" on page 170
SELECT_LOWER_VG	"VG Selection Parameters for a DCM STORE_DIRECTORY" on page 204
SELECT_MSP	"MSP/VG Selection Parameters for a User Filesystem" on page 200
SELECT_VG	"MSP/VG Selection Parameters for a User Filesystem" on page 200
SERVER_NAME	"base Object" on page 152
SERVICES	"node Object" on page 163
SERVICES_PORT	"services Object" on page 167
SITE_SCRIPT	"Disk Cache Manager (DCM) msp Object" on page 254 "Automated Space Management Parameters for a User Filesystem" on page 197 "Automated Space Management Parameters for a DCM STORE_DIRECTORY" on page 201
SPACE_WEIGHT	"File Weighting Parameters for a User Filesystem" on page 198
SPOOL_DIR	"base Object" on page 152
STORE_DIRECTORY	"Disk msp Object" on page 250 "Disk Cache Manager (DCM) msp Object" on page 254

Parameter	Section Discussed In
TASK_GROUPS	"dmdaemon Object" on page 160 "services Object" on page 167 "taskgroup Object" on page 170 "filesystem Object" on page 187 "libraryserver Object" on page 215 "drivegroup Object" on page 217 "volumegroup Object" on page 224 "FTP msp Object" on page 245 "Disk msp Object" on page 250 "Disk Cache Manager (DCM) msp Object" on page 254
TSREPORT_OPTIONS	"taskgroup Parameters" on page 174
THRESHOLD	"LS Tasks" on page 240
TIMEOUT_FLUSH	"volumegroup Object" on page 224
TMF_TMMNT_OPTIONS	"device Object" on page 186 "drivegroup Object" on page 217
TMP_DIR	"base Object" on page 152
TYPE	"base Object" on page 152 "dmdaemon Object" on page 160 "node Object" on page 163 "services Object" on page 167 "taskgroup Object" on page 170 "device Object" on page 186 "filesystem Object" on page 187 "policy Object" on page 193 "libraryserver Object" on page 215 "drivegroup Object" on page 217 "volumegroup Object" on page 224 "resourcescheduler Object" on page 230 "resourcewatcher Object" on page 231 "FTP msp Object" on page 245 "Disk msp Object" on page 250 "Disk Cache Manager (DCM) msp Object" on page 254
USE_UNIFIED_BUFFER	"filesystem Object" on page 187
VERIFY_POSITION	"drivegroup Object" on page 217
VOLUME_GROUPS	"drivegroup Object" on page 217

Parameter	Section Discussed In
VOLUME_LIMIT	"LS Tasks" on page 240
WATCHER	"libraryserver Object" on page 215
WEIGHT	"resourcescheduler Object" on page 230
WRITE_CHECKSUM	"drivegroup Object" on page 217 "FTP msp Object" on page 245 "Disk msp Object" on page 250 "Disk Cache Manager (DCM) msp Object" on page 254
ZONE_SIZE	"volumegroup Object" on page 224

Note: The run_tape_merge.sh and run_merge_stop.sh tasks and their associated parameters can be specified in the volumegroup object.

- "Automated Space Management Parameters for a DCM STORE_DIRECTORY" on page 201
- "File Weighting Parameters for a DCM STORE_DIRECTORY" on page 202
- "VG Selection Parameters for a DCM STORE_DIRECTORY" on page 204

Parallel Data Mover Option Configuration

This chapter discusses the following:

Parallel Data Mover Option Configuration Procedure

If you are running DMF with the Parallel Data Mover Option, do the following:

Procedure 8-1 Configuring DMF for the Parallel Data Mover Option

1. Configure the DMF configuration file (`/etc/dmf/dmf.conf`) on the DMF server according to the instructions in "Configuration Objects Overview" on page 149. Ensure that a `node` object is defined in `dmf.conf` for the parallel data mover node that is being added.
2. Copy the `dmf.conf` file from the DMF server to the DMF parallel data mover node.

   Note: Do not edit the `dmf.conf` file on the parallel data mover node.

3. Install the **SGI DMF Parallel Data Mover** YaST pattern on the parallel data mover node. See the *SGI InfiniteStorage Software Platform* release note for more information.
4. Configure CXFS according to the instructions in the *CXFS 6 Administration Guide for SGI InfiniteStorage*.
5. Include the DMF parallel data mover node as a CXFS client, such as by creating a `cxfs_admin autoconf` rule. For more information, see the `cxfs_admin` chapter's information about the `autoconf` command in the CXFS administration guide and the `cxfs_admin`(8) man page.

For example, for two parallel data mover nodes named pdm1 and pdm2 in a CXFS cluster named mycluster:

```
# cxfs_admin -c "create autoconf rule_name=pdm1rule policy=allowed \
hostname=pdm1 enable_node=true" -i mycluster
# cxfs_admin -c "create autoconf rule_name=pdm2rule policy=allowed \
hostname=pdm2 enable_node=true" -i mycluster
```

After you have finished creating or modifying all of the desired autoconf rules, you must unlock all cxfs_admin sessions in order for nodes to be automatically configured. (The automatic configuration process must have access to the cxfs_admin lock.)

If a node you refer to in an autoconf rule has previously been part of the CXFS cluster, or if the node fails to join the CXFS membership, you must reboot the node.

6. Ensure the following:

   - The file containing the OpenVault security keys used by DMF must be visible to the DMF server and all parallel data mover nodes. The OV_KEY_FILE parameter in dmf.conf specifies the name and path of this file. See "base Object" on page 152.
   - The CXFS filesystems defined by the following DMF parameters are configured to be mounted only on the primary DMF server, the passive DMF server (if applicable), and each parallel data mover node:

     CACHE_DIR
     MOVE_FS
     TMP_DIR
     SPOOL_DIR
     STORE_DIRECTORY (for a disk cache manager, DCM)

     For example, if the filesystem to be mounted on the directory specified by CACHE_DIR is on the /dev/cxvm/fscache device, you could specify the following cxfs_admin commands to restrict it to the CXFS potential metadata server nodes on which the DMF server can run (say server1 and server2) and the parallel data mover nodes (say pdm1 and pdm2):

     ```
     # cxfs_admin -c "create filesystem name=fscache mount_new_nodes=false \
     nodes=server1,server2,pdm1,pdm2" -i mycluster
     ```

For more information, see the `cxfs_admin` chapter's information about the mount command in the CXFS administration guide.

7. Ensure that DMF-managed user filesystems are CXFS filesystems and that they are mounted on the DMF server and all of the parallel data mover nodes. They may also be mounted on CXFS client-only nodes.
8. On the DMF server, use `ov_admin` to allow the parallel data mover node to be a DCP-enabled OpenVault client machine. Do the following:
   a. From the main menu in `ov_admin`, enter `23` to select `Manage OpenVault Client Machines`.
   b. Enter `1` to select `Activate an OpenVault Client Machine` and follow the prompts. Be sure to answer `yes` when asked if the machine will run DCPs.

   For more information about `ov_admin`, see the *OpenVault Operator's and Administrator's Guide*.
9. On the parallel data mover node, use `ov_admin` to configure DCPs for those tape drives that it should operate.
10. If not already done, give the parallel data mover permission to use the DMF application in OpenVault by setting up the application instances as described in step 4 of Procedure 7-4, page 235.
11. Verify the DMF configuration by using the `dmmaint` tool on the DMF server and clicking the **Inspect** button, or use the `dmcheck`(8) script on the DMF server.

    If there are errors, fix them by clicking the **Configure** button to edit the configuration file. Repeat these steps until there are no errors.
12. Start the DMF mover service on the parallel data mover node:

    ```
    pdmn# service dmf_mover start
    ```

After initial configuration, changes to `dmf.conf` will normally be propagated to parallel data mover nodes automatically while the DMF services are running. Certain changes, such as changing the `SERVER_NAME` or `SERVICES_PORT` of the DMF server, will require `dmf.conf` to manually be copied to the parallel data mover nodes followed by a restart of the DMF services on those nodes.

Determining the State of Parallel Data Mover Nodes

To determine the status of a parallel data mover node, use the following command as `root`:

```
# dmnode_admin -l
```

For example, showing the state for parallel data mover nodes `jar` and `zin`:

```
# dmnode_admin -l
Node Name  State     Enabled  Active Since          Dropouts
jar        Inactive  Yes      -                     0
zin        Active    Yes      2008-Nov-26,12:45:48  0
```

The node state can be one of the following:

`Active`	The node is connected to the `dmnode_service` on the DMF server and is eligible to run data mover processes.
`Inactive`	The node is not connected to the `dmnode_service`.
`Disabled`	The node is connected to the `dmnode_service` but has been disabled using `dmnode_admin`. See "Disabling Parallel Data Mover Nodes" on page 274.

The `Dropouts` field specifies the number of times that the node has transitioned from `Active` to `Inactive`. This count is reset when `dmnode_service` is restarted.

Note: If the `dmnode_service` is not running, the `dmnode_admin` command will not function. To restart `dmnode_service`, enter the following:

```
# service dmf start
```

Disabling Parallel Data Mover Nodes

To disable parallel data mover nodes in order to perform maintenance on the system or to diagnose a problem, enter the following:

```
# dmnode_admin -d nodename ...
```

The node will remain disabled across DMF restarts.

The disabled node is no longer eligible to start new data mover processes.

Existing data mover processes on the disabled node will be told to exit after the library server notices this change, which may take up to 2 minutes. The existing data mover processes may exit in the middle of recalling or migrating a file; this work will be reassigned to other data mover processes. Stopping data mover processes with the following command has the same result on existing processes:

```
# service dmf_mover stop
```

Reenabling Parallel Data Mover Nodes

To reenable parallel data mover nodes, making them eligible to run data mover processes, enter the following as `root`:

```
# dmnode_admin -e nodename ...
```

The node will remain enabled across DMF restarts.

To determine the current state of a node, see "Determining the State of Parallel Data Mover Nodes" on page 274

Note: DMF and DMF Manager must be running for the `dmnode_admin` command to function.

Message Logs

The dmfdaemon, dmlockmgr, dmfsmon, media-specific process (MSP), and library server (LS) message log files use the same general naming convention and message format. The message log file names are created using the extension *yyyymmdd*, which represents the year, month, and day of log file creation.

Each line in a message log file begins with the time the message was issued, an optional message level, the process ID number, and the name of the program that issued the message.

The optional message level is described below. The remainder of the line contains informative or diagnostic information. The following sections provide details about each of these log files:

- "Automated Space Management Log File" on page 283 for information about dmfsmon and autolog.*yyyymmdd*
- "Daemon Logs and Journals" on page 294 for information about dmfdaemon and dmdlog.*yyyymmdd*
- "dmlockmgr Communication and Log Files" on page 297 for information about dmlockmgr and dmlocklog.*yyyymmdd*
- "LS Logs" on page 308 and "FTP MSP Activity Log" on page 337 for information about dmatls, dmdskmsp, dmftpmsp, and msplog.*yyyymmdd*
- Chapter 14, "DMF Maintenance and Recovery" on page 347, for information about log file maintenance

Messages in the dmdlog, dmlocklog, moverlog, and msplog files contain a 2–character field immediately following the time field in each message that is issued. This feature helps to categorize the messages and can be used to extract error messages automatically from these logs. Because the only indication of DMF operational failure may be messages written to the DMF logs, recurring problems can go undetected if you do not check the logs daily.

Possible message types for autolog, dmdlog, moverlog, msplog, and dmlocklog are defined in Table 9-1. The table also lists the corresponding message levels in the configuration file.

Table 9-1 DMF Log File Message Types

Field	Message Type	Message Level
`-E`	Error	0
`-O`	Ordinary	0
`-I`	Informative	1
`-V`	Verbose	2
`-1`	Debug level 1	3
`-2`	Debug level 2	4
`-3`	Debug level 3	5
`-4`	Debug level 4	6

Automated Space Management

This chapter discusses the following:

The `dmfsmon` Daemon and `dmfsfree` Command

The `dmfsmon`(8) daemon monitors the free-space levels in filesystems configured with automated space management enabled (`auto`). When the free space in one of the filesystems falls below the free-space minimum, `dmfsmon` invokes `dmfsfree`(8). The `dmfsfree` command attempts to bring the free space and migrated space of a filesystem into compliance with configured values. `dmfsfree` may also be invoked directly by system administrators.

When the free space in one of the filesystems falls below its minimum, the `dmfsfree` command performs the following steps:

- Scans the filesystem for files that can be migrated and freed or ranges of files that can be freed. Each of these candidates is assigned a weight. This information is used to create a list, called a *candidate list*, that contains an entry for each file or range and is ordered by weight (largest to smallest).
- Selects enough candidates to bring the free space back up to the desired level. Files or ranges of files are selected in order from largest weight to smallest.
- Selects enough non-migrated files from the candidate list to achieve the *migration target*, which is the percentage of filesystem space you want to have as free space **and** space occupied by migrated but online files. Files are selected from the candidate list in order from largest weight to smallest.

The `dmfsmon` daemon should be running whenever DMF is active. You control automated space management by setting the filesystem and policy configuration

parameters in the DMF configuration file. The configuration parameters specify targets for migration and free space as well as one or more policies for weighting. Only filesystems configured as `MIGRATION_LEVEL auto` in the configuration file are included in the space-management process. "`policy` Object" on page 193, describes how to configure automated space management.

You can change the migration level of a filesystem by editing the configuration file.

Generating the Candidate List

The first step in the migration process occurs when `dmfsmon` determines it is time to invoke `dmfsfree`, which scans the filesystem and generates the candidate list. During candidate list generation, the inode of each online file in the specified filesystem is audited and a weight is computed for it.

A filesystem is associated with a weighting policy in the DMF configuration file. The applicable weighting policy determines a file's total weight, or, if a `ranges` clause is specified in the configuration file, the range's total weight. Total file or range weight is the sum of the `AGE_WEIGHT` and `SPACE_WEIGHT` parameters. Defaults are provided for these parameters, and you can configure either to make a change. You do not need to configure a weighting policy if the defaults are acceptable, but you should be aware that the default selects files based on age and not on size. If you want to configure a policy based on size that ignores file age, you should overwrite the default for `AGE_WEIGHT`.

The default weighting policy bases the weight of the file on the time that has passed since the file was last accessed or modified. Usually, the more recent a file's access, the more likely it is to be accessed again.

The candidate list is ordered by total file or range weight (largest to smallest). You can prevent a file from being automatically migrated by making sure that no ranges within the file have a positive weight value. You can configure the weighting parameters to have a negative value to ensure that certain files or ranges are never automatically freed.

Note: If you use negative weights to exclude files or ranges from migration, you must ensure that a filesystem does not fill with files or ranges that are never selected for automatic migration.

You can use the dmscanfs(8) command to print file information to standard output (stdout).

Selection of Migration Candidates

The dmfsfree(8) utility processes each ordered candidate list sequentially, seeking candidates to migrate and possibly free. The extent of the selection process is governed by values defined for the filesystem in the DMF configuration file as described in "policy Object" on page 193.

The most essential parameters are as follows:

- FREE_SPACE_MINIMUM specifies the minimum percentage of filesystem space that must be free. When this value is reached, dmfsmon will take action to migrate and free enough files or ranges to bring the filesystem into compliance. For example, setting this parameter to 10 indicates that when less than 10% of the filesystem space is free, dmfsmon will migrate and free files to achieve the percentage of free space specified by FREE_SPACE_TARGET. For the information on how this parameter is used when automated space management is not configured, see the dmf.conf(5) man page.
- FREE_SPACE_TARGET specifies the percentage of free filesystem space dmfsmon will try to achieve if free space falls below FREE_SPACE_MINIMUM. For example, if this parameter is set to 15 and FREE_SPACE_MINIMUM is set to 10, dmfsmon takes action when the filesystem is less than 10% free and migrates and frees files until 15% of the filesystem is available.
- MIGRATION_TARGET specifies the percentage of filesystem capacity that is maintained as a reserve of space that is free or occupied by dual-state files. DMF attempts to maintain this reserve in the event that the filesystem free space reaches or falls below FREE_SPACE_MINIMUM.

When dmfsmon detects that the free space on a filesystem has fallen below the level you have set as FREE_SPACE_MINIMUM, it invokes dmfsfree to select a sufficient number of candidates to meet the FREE_SPACE_TARGET. The dmfsfree utility ensures that these files are migrated and releases their disk blocks. It then selects additional candidates to meet the MIGRATION_TARGET and migrates them.

Figure 10-1 shows the relationship of automated space management migration targets to each other. Migration events occur when file activity causes free filesystem space to drop below FREE_SPACE_MINIMUM. dmfsmon generates a candidate list and begins to migrate files and free the disk blocks until the FREE_SPACE_TARGET is met, and

then it migrates regular files (creating dual-state files) until the MIGRATION_TARGET is met.

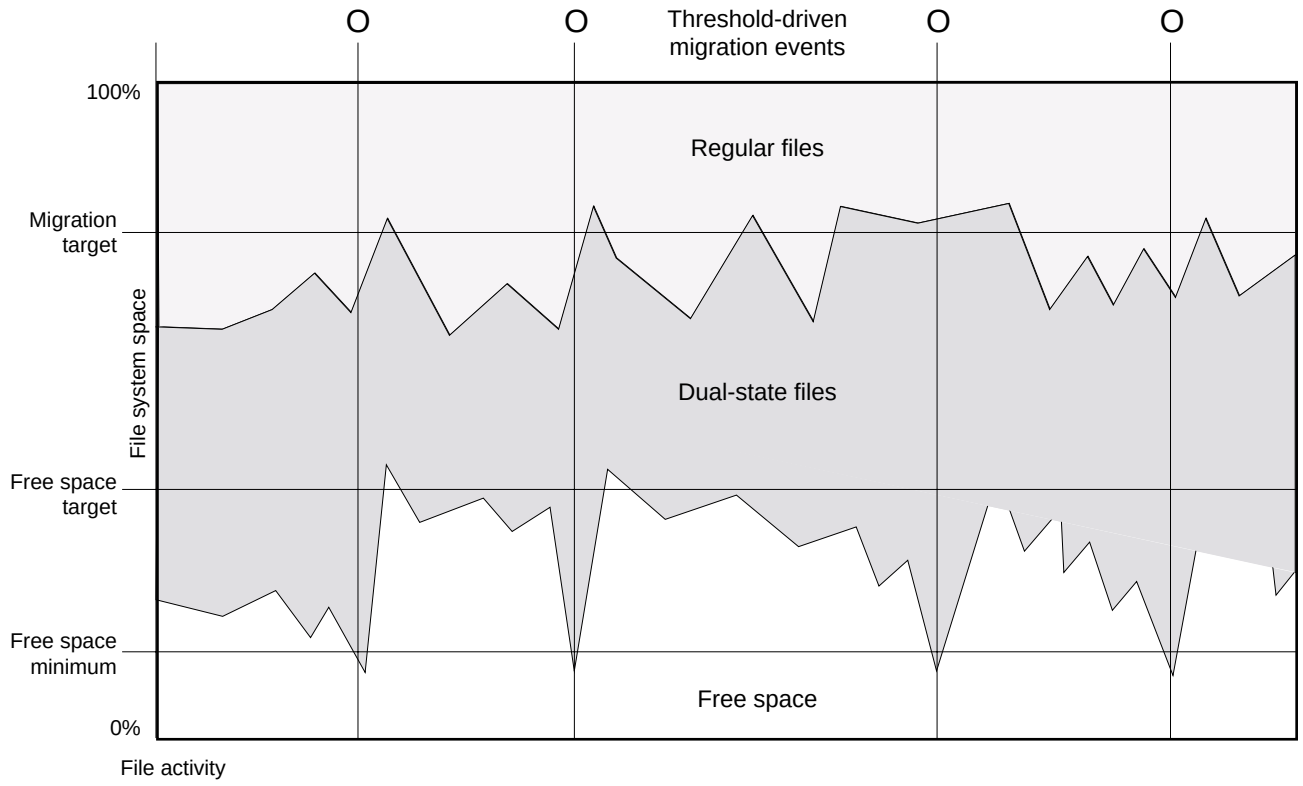

Figure 10-1 Relationship of Automated Space Management Targets

If dmfsmon does not find enough files to migrate (because all remaining files are exempt from migration), it uses another configuration parameter to decrement FREE_SPACE_MINIMUM.

FREE_SPACE_DECREMENT specifies the percentage of filesystem space by which dmfsmon will decrement FREE_SPACE_MINIMUM if it cannot find enough files to migrate to reach FREE_SPACE_MINIMUM. For example, suppose FREE_SPACE_MINIMUM is set to 10 and FREE_SPACE_DECREMENT is set to 2. If dmfsmon cannot find enough files to migrate to reach 10% free space, it will decrement FREE_SPACE_MINIMUM to 8 and try to find enough files to migrate so that

8% of the filesystem is free. If dmfsmon cannot achieve this percentage, it will decrement FREE_SPACE_MINIMUM to 6. dmfsmon will continue until it reaches a value for FREE_SPACE_MINIMUM that it can achieve, and it will try to maintain that new value. dmfsmon restores FREE_SPACE_MINIMUM to its configured value when it can be achieved. The default value for FREE_SPACE_DECREMENT is 2.

Note: DMF manages real-time partitions differently than files in a normal partition. The dmfsfree command can only migrate files in the non-real-time partition; it ignores files in the real-time partition. Any configuration parameters you set will apply only to the non-real-time partition. Files in the real-time partition can be manually migrated with the commands dmget(1), dmput(1), and dmmigrate(8). Files are retrieved automatically when they are read.

Space Management and the Disk Cache Manager

DMF prevents the DCM cache from filling by following the same general approach it takes with DMF-managed filesystems, with the following differences:

- The disk MSP (dmdskmsp) monitors the cache, instead of a separate monitoring program such as dmfsmon.
- The dmdskfree utility controls the movement of cache files to tape. This is analogous to dmfsfree.

Note: DCM uses parameters that are similar to those used for the disk MSP, although some names are different. See "policy Object" on page 193.

Automated Space Management Log File

All of the space-management commands record their activities in a common log file, autolog.*yyyymmdd* (where *yyyymmdd* is the year, month, and day of log file creation). The first space-management command to execute on a given day creates the log file for that day. This log file resides in the directory *SPOOL_DIR/daemon_name* (The *SPOOL_DIR* value is specified by the SPOOL_DIR configuration parameter; see "base Object" on page 152). The space-management commands create the

daemon_name subdirectory in *SPOOL_DIR* if it does not already exist. The full pathname of the common log file follows:

SPOOL_DIR/*daemon_name*/autolog.*yyyymmdd*

Each line in the autolog file begins with the time of message issue, followed by the name of the host where the message issuer ran, and the process number and program name of the message issuer. The remainder of the line contains informative or diagnostic information such as the following:

- Name of the filesystem being processed
- Number of files selected for migration and freeing
- Number of disk blocks that were migrated and freed
- Names of any other DMF commands executed
- Command's success or failure in meeting the migration and free-space targets

The following excerpt shows the format of an autolog file (line breaks shown here for readability)

```
23:39:35:702-V   zap  237082-dmfsmon /dmfusr1 - free_space=39.79, minimum=38
23:39:35:702-V   zap  237082-dmfsmon /dmfusr3 - free_space=15.48,minimum=15
23:40:55:723-I   zap  237082-dmfsmon Started 3409 for execution on /dmfusr3
23:40:56:782-I   zap    3409-dmfsfree /dmfusr3 - Number of blocks in the filesystem = 122232448
23:40:56:782-I   zap    3409-dmfsfree /dmfusr3 - Number of blocks in the free space target = 24446490 (20%)
23:40:56:782-I   zap    3409-dmfsfree /dmfusr3 - Number of blocks currently free = 18287168 (15.0%)
23:40:56:782-I   zap    3409-dmfsfree /dmfusr3 - Number of blocks to free = 6159322 (5.0%)
23:40:56:782-I   zap    3409-dmfsfree /dmfusr3 - Number of blocks in the migration target = 97785960 (80%)
23:40:56:782-I   zap    3409-dmfsfree /dmfusr3 - Number of blocks currently migrated = 74419040 (60.9%)
23:40:56:782-I   zap    3409-dmfsfree /dmfusr3 - Number of blocks to migrate = 5079752 (4.2%)
23:40:56:782-I   zap    3409-dmfsfree /dmfusr3 - Summary of files: online = 3760, offline = 6537, unmigrating
   = 30, partial = 0
23:40:56:782-I   zap    3409-dmfsfree /dmfusr3 - Number of candidates = 3629, rejected files = 0, rejected
   ranges = 0
23:41:31:150-I   zap    3409-dmfsfree /dmfusr3 - Migrated 5104824 blocks in 169 files
23:41:31:150-I   zap    3409-dmfsfree /dmfusr3 - Freed 6164480 blocks in 303 files
23:41:31:150-O   zap    3409-dmfsfree /dmfusr3 - Exiting: minimum reached - targets met by outstanding requests.
```

The DMF Daemon

The DMF daemon, dmfdaemon(8), is the core component of DMF. The daemon exchanges messages with commands, the kernel, the media-specific processes (MSPs), and the library servers (LSs).

When DMF is started, the daemon database is automatically initialized. To start the daemon manually, use the DMF startup script, as follows:

```
# service dmf start
```

Caution: For instructions about starting and stopping DMF and the mounting service in an HA environment, see the SGI HA documentation.

Typically, DMF should be initialized as part of the normal system startup procedure by using a direct call in a system startup script in the /etc/rc2.d directory.

The following sections provide additional information about the daemon database and daemon processing:

- "Daemon Processing" on page 285
- "Daemon Database and dmdadm" on page 287
- "Daemon Logs and Journals" on page 294

Daemon Processing

After initialization, dmfdaemon performs the following steps:

1. Isolates itself as a daemon process.
2. Checks for the existence of other dmfdaemon processes. If another dmfdaemon exists, the newer one terminates immediately.
3. Initializes the daemon log.
4. Opens the daemon database.
5. Initializes the daemon request socket.

6. Initiates the MSPs and LSs.
7. Enters its main request processing.

The daemon uses log files and journal files as described in "Daemon Logs and Journals" on page 294.

The main request processing section of the DMF daemon consists of the following sequence:

- The `select`(2) system call, which is used to wait for requests or for a default time-out interval
- A request dispatch switch to read and process requests detected by the `select` call
- A time processor, which checks activities (such as displaying statistics and running the administrator tasks) done on a time-interval basis

This processing sequence is repeated until a stop request is received from the `dmdstop`(8) command. When a normal termination is received, the MSPs and LSs are terminated, the daemon database is closed, and the logs are completed.

A typical request to the daemon starts with communication from the requester. The requester is either the kernel (over the DMF device interface) or a user-level request (from the command pipe). A user-level command can originate from the automated space-management commands or from an individual user.

After receipt, the command is dispatched to the appropriate command processor within the daemon. Usually, this processor must communicate with an MSP or LS before completing the specified request. The commands are queued within the daemon and are also queued to a specific group of daemon database entries. All entries referring to the same file share the same BFID. The command is dormant until the reply from the MSP/LS is received or the MSP/LS terminates. When command processing is completed, a final reply is sent to the issuing process, if it still exists.

A final reply usually indicates that the command has completed or an error has occurred. Often, error responses require that you analyze the daemon log to obtain a full explanation of the error. An error response issued immediately usually results from an invalid or incorrect request (for example, a request to migrate a file that has no data blocks). A delayed error response usually indicates a database, daemon, MSP, or LS problem.

Daemon Database and dmdadm

The daemon database resides in the directory *HOME_DIR*/*daemon_name* (*HOME_DIR* is specified by the HOME_DIR configuration parameter). This daemon database contains information about the offline copies of a given file, as well as some information about the original file. The daemon database also contains the bit-file identifier (BFID), which is assigned when the file is first migrated.

Other information maintained on a per-entry basis includes the following:

- File size (in bytes)
- MSP or volume group (VG) name and recall path
- Date and time information, including the following:
  - Time at which the record was created
  - Time at which the record was last updated
  - A check time for use by the administrator
  - A soft-delete time, indicating when the entry was soft-deleted
- Original device and inode number
- Base portion of the original file name, if known

The dmdadm(8) command provides maintenance services for the daemon database.

dmdadm executes directives from stdin or from the command line when you use the -c option. All directives start with a directive name followed by one or more parameters. Parameters may be positional or keyword-value pairs, depending on the command. White space separates the directive name, keywords, and values.

When you are inside the dmdadm interface (that is, when you see the adm *command_number* > prompt), the command has a 30–minute timeout associated with it. If you do not enter a response within 30 minutes of the prompt having been displayed, the dmdadm session terminates with a descriptive message. This behavior on all the database administrative commands limits the amount of time that an administrator can lock the daemon and MSP/LS databases from updates.

The rest of this section discusses the following:

- "dmdadm Directives" on page 288
- "dmdadm Field and Format Keywords" on page 290

• "dmdadm Text Field Order" on page 294

dmdadm Directives

The dmdadm directives are as follows:

Directive	Description
count	Displays the number of records that match the expression provided.
create	Creates a record.
delete	Deletes the specified records.
dump	Prints the specified records to standard out in ASCII; each field is separated by the pipe character (|).
help	Displays help.
list	Shows the fields of selected records. You may specify which fields are shown.
load	Applies records to the daemon database obtained from running the dump directive.
quit	Stops program execution after flushing any changed database records to disk. The abbreviation q and the string exit produce the same effect.
set	Specifies the fields to be shown in subsequent list directives.
update	Modifies the specified records.

The syntax for the dmdadm directives is as follows:

```
count selection [limit]
create bfid settings
delete selection [limit]
dump selection [limit]
help
list selection [limit] [format]
load filename
quit (or q or exit)
set format
update selection [limit] to settings...
```

where:

- The *selection* parameter specifies the records to be acted upon.
- The *limit* parameter restricts the records acted upon.
- The *bfid* parameter for the create directive specifies the bit-file-identifier (BFID) for the record being created.
- The *settings* parameter for the create and update directives specifies one or more fields and their values.
- The *format* parameter selects the way in which output is displayed. Any program or script that parses the output from this command should explicitly specify a format; otherwise the default is used, which may change from release to release.

The value for *selection* can be one of the following:

- A BFID or range of BFIDs
- The keyword `all`
- A period (`.`), which recalls the previous selection
- An expression involving any of the above, field value comparisons, `and`, `or`, or parentheses

A field value comparison may use the following to compare a field keyword to an appropriate value:

< (less than)
> (greater than)
= (equal to)
!= (not equal to)
<= (less than or equal to)
>= (greater than or equal to)

The syntax for *selection* is as follows:

selection ::= *or-expr*
or-expr ::= *and-expr* [or *or-expr*]
and-expr ::= *nested-expr* [and *or-expr*]
nested-expr ::= *comparison* | (*or-expr*)
comparison ::= *bfid-range* | *field-keyword op field-value*
op ::= < | > | = | != | >= | <=
bfid-range ::= *bfid* [- *bfid*] | [*bfid* - [*bfid*]] | *key-macro*

key-macro ::= `all`
field-keyword ::= *name or abbreviation of the record field*
field-value ::= *appropriate value for the field*
bfid ::= *character representation of the bfid*

Thus valid values for *selection* could be any of the following:

```
305c74b200000010-305c74b200000029
7fffffff000f4411-
-305c74b2000004c8
all
origsize>1m
. and origage<7d
```

dmdadm Field and Format Keywords

The *field* parameter keywords listed below can be used as part of a *selection* parameter to select records. They can also be used in a *settings* parameter, as part of a keyword-value pair, to specify new values for a field, or in a *format* parameter. When specifying new values for fields, some of the field keywords are valid only if you also specify the `-u` (unsafe) option.

Keyword	Description
`checkage` (`ca`)	The time at which the record was last checked; the same as `checktime`, except that it is specified as an *age string* (see below). Valid only in unsafe (`-u`) mode.
`checktime` (`ct`)	The time at which the record was last checked; an integer that reflects raw UNIX or Linux time. Valid only in unsafe (`-u`) mode.
`deleteage` (`da`)	The time at which the record was soft-deleted; the same as `deletetime`, except that it is specified as an age string. Valid only in unsafe (`-u`) mode.
`deletetime` (`dt`)	The time at which the record was soft-deleted; an integer that reflects raw UNIX or Linux time. Valid only in unsafe (`-u`) mode.
`mspname` (`mn`)	The name of the MSP or VG with which the file is associated; a string of up to 8 characters. Valid only in unsafe (`-u`) mode.

`mspkey` (`mk`)	The string that the MSP or VG can use to recall a record; a string of up to 50 characters. Valid only in unsafe (`-u`) mode.
`origage` (`oa`)	Time at which the record was created; the same as `origtime`, except that it is specified as an age string.
`origdevice` (`od`)	Original device number of the file; an integer.
`originode` (`oi`)	Original inode number of the file; an integer.
`origname` (`on`)	Base portion of the original file name; a string of up to 14 characters.
`origsize` (`os`)	Original size of the file; an integer.
`origtime` (`ot`)	Time at which the record was created; an integer that reflects raw UNIX or Linux time.
`origuid` (`ou`)	Original user ID of the record; an integer.
`updateage` (`ua`)	Time at which the record was last updated; the same as `updatetime`, except that it is specified as an age string.
`updatetime` (`ut`)	Time at which the record was last updated; an integer that reflects raw UNIX or Linux time.

The time field keywords (`checktime`, `deletetime`, `origtime`, and `updatetime`) have a value of either `now` or UNIX or Linux *raw time* (seconds since January 1, 1970). These keywords display their value as raw time. The value comparison `>` used with the date keywords means newer than the value given. For example, `>36000` is newer than 10AM on January 1, 1970, and `>852081200` is newer than 10AM on January 1, 1997.

The age field keywords (`checkage`, `deleteage`, `origage`, and `updateage`) let you express time as a string. They display their value as an integer followed by the following:

`w` (weeks)
`d` (days)
`h` (hours)
`m` (minutes)
`s` (seconds)

For example, `8w12d7h16m20s` means 8 weeks, 12 days, 7 hours, 16 minutes, and 20 seconds old.

The comparison > used with the age keywords means older than the value given (that is, `>5d` is older than 5 days).

A *limit* parameter in a directive restricts the records acted upon. It consists of one of the following keywords followed by white space and then a value:

Keyword	Description
`recordlimit` (`rl`)	Limits the number of records acted upon to the value that you specify; an integer.
`recordorder` (`ro`)	Specifies the order that records are scanned: • `bfid`, which specifies that the records are scanned in BFID order. • `data`, which specifies that the records are scanned in the order in which they are found in the daemon database data file. `data` is more efficient for large databases, although it is essentially unordered.

The *format* parameter selects a format to use for the display. If, for example, you want to display fields in a different order than the default or want to include fields that are not included in the default display, you specify them with the format parameter. The format parameter in a directive consists of one of the following:

- `format default`
- `format keyword`
- `format` *field-keywords*

The `format keyword` form is intended for parsing by a program or script and therefore suppresses the headings.

The *field-keywords* may be delimited by colons or white space; white space requires the use of quotation marks.

Note: `BFID` is always included as the first field and need not be specified.

For any field that takes a byte count, you may append the letter `k`, `m`, or `g` (in either uppercase or lowercase) to the integer to indicate that the value is to be multiplied by one thousand, one million, or one billion, respectively.

The following is sample output from the `dmdadm list` directive; `recordlimit 20` specifies that you want to see only the first 20 records.

```
adm 3>list all recordlimit 20
            BFID     ORIG        ORIG   ORIG MSP      MSP
                      UID        SIZE    AGE NAME     KEY
-------------------------------------------------------------------------
305c74b200000010    20934    69140480   537d silo1    88b49f
305c74b200000013    26444      279290   537d silo1    88b4a2
305c74b200000014    10634       67000   537d silo1    88b4a3
305c74b200000016    10634   284356608   537d silo1    88b4a5
305c74b200000018    10634     1986560   537d silo1    88b4a7
305c74b20000001b    26444      232681   537d silo1    88b4aa
305c74b20000001c    10015     7533688   537d silo1    88b4ab
305c74b200000022    8964     23194990   537d silo1    88b4b1
305c74b200000023    1294    133562368   537d silo1    88b4b2
305c74b200000024    10634       67000   537d silo1    88b4b3
305c74b200000025    10634   284356608   537d silo1    88b4b4
305c74b200000026    10634     1986560   537d silo1    88b4b5
305c74b200000027    1294      1114112   537d silo1    88b4b6
305c74b200000028    10634       25270   537d silo1    88b4b7
305c74b200000029    1294     65077248   537d silo1    88b4b8
305c74b20000002b    9244      2740120   537d silo1    88b4ba
305c74b200000064    9335         9272   537d silo1    88b4f3
305c74b200000065    9335        10154   537d silo1    88b4f4
305c74b200000066    9335         4624   537d silo1    88b4f5
305c74b200000067    9335        10155   537d silo1    88b4f6
adm 4>
```

The following example displays the number of records in the daemon database that are associated with user ID `11789` and that were updated during the last five days:

```
adm 3>count origuid=11789 and updateage<5d
72 records found.
```

dmdadm Text Field Order

The text field order for daemon records generated by the dmdump(8), dmdumpj(8), and the dump directive in dmdadm is listed below. This is the format expected by the load directives in dmdadm:

1. bfid
2. origdevice
3. originode
4. origsize
5. origtime
6. updatetime
7. checktime
8. deletetime
9. origuid
10. origname
11. mspname
12. mspkey

To isolate the mspname and mspkey from the daemon records soft-deleted fewer than three days ago, use the following command:

```
dmdadm -c "dump deleteage<3d and deletetime>0" | awk "-F|" '(print $11,$12}'
```

Daemon Logs and Journals

The DMF daemon uses log files to track various types of activity. Journal files are used to track daemon database transactions.

The ASCII log of daemon actions has the following format (*SPOOL_DIR* refers to the directory specified by the SPOOL_DIR configuration parameter):

SPOOL_DIR/*daemon_name*/dmdlog.*yyyymmdd*

The file naming convention is that *yyyy*, *mm*, and *dd* correspond to the date on which the log was created (representing year, month, and day, respectively). Logs are created automatically by the DMF daemon.

Note: Because the DMF daemon will continue to create log files and journal files without limit, you must remove obsolete files periodically by configuring the `run_remove_logs` and `run_remove_journals` tasks in the configuration file, as described in "`taskgroup` Object" on page 170.

The DMF daemon automatically creates journal files that track daemon database transactions. They have the following pathname format (*JOURNAL_DIR* refers to the directory defined by the `JOURNAL_DIR` configuration parameter):

JOURNAL_DIR`/`*daemon_name*`/dmd_db.`*yyyymmdd[.hhmmss]*

Existing journal files are closed and new ones created in two circumstances:

- When the first transaction after midnight occurs
- When the journal file reaches size defined by the `JOURNAL_SIZE` configuration parameter

When the first transaction after midnight occurs, the existing open journal file is closed, and the suffix `.235959` is appended to the current file name no matter what the time (or date) of closing. The closed file represents the last (or only) transaction log of the date *yyyymmdd*. A new journal file with the current date is then created.

When the journal file reaches `JOURNAL_SIZE`, the file is closed and the suffix *.hhmmss* is added to the name; *hh*, *mm*, and *ss* represent the hour, minute, and second of file closing. A new journal file with the same date but no time is then created.

For example, the following shows the contents of a *JOURNAL_DIR*`/`*daemon_name* directory on 15 June 1998:

```
dmd_db.19980604.235959  dmd_db.19980612.235959
dmd_db.19980605.235959  dmd_db.19980613.145514
dmd_db.19980608.235959  dmd_db.19980613.214233
dmd_db.19980609.235959  dmd_db.19980613.235959
dmd_db.19980610.235959  dmd_db.19980614.235959
dmd_db.19980611.094745  dmd_db.19980615
dmd_db.19980611.101937
dmd_db.19980611.110429
dmd_db.19980611.235959
```

For every date on which daemon database transactions occurred, there will exist a file with that date and the suffix `.235959`, with the exception of an existing open journal file. Some dates have additional files because the transaction log reached `JOURNAL_SIZE` at a specified time and the file was closed.

You can configure `daemon_tasks` parameters to remove old journal files (using the `run_remove_journals.sh` task and the `JOURNAL_RETENTION` parameter. For more information, see "`taskgroup` Object" on page 170.

Warning: If a daemon database becomes corrupt, recovery consists of applying journals to a backup copy of the database. Database recovery procedures are described in "Database Recovery" on page 360.

The DMF Lock Manager

The dmlockmgr(8) process must be executing at all times for any DMF process to safely access and update a DMF database. The dmlockmgr process and its clients (such as dmatls, dmfdaemon(8),dmvoladm(8), and dmcatadm(8)) communicate through files, semaphores, and message queues. There are times when abnormal process terminations will result in non-orderly exit processing that will leave files and/or interprocess communication (IPC) resources allocated. As a DMF administrator, periodically you will want to look for these resources to remove them.

Note: *HOME_DIR* and *SPOOL_DIR* refer to the values of the HOME_DIR and SPOOL_DIR parameter, respectively, in the DMF configuration file. See "base Object" on page 152..

The dmlockmgr files used by the database utilities are found in several different places. There are the following types of files:

- "dmlockmgr Communication and Log Files" on page 297
- "dmlockmgr Individual Transaction Log Files" on page 299

dmlockmgr Communication and Log Files

The dmlockmgr communication and activity log files are all found in a directory formed by *HOME_DIR*/RDM_LM. The *HOME_DIR*/RDM_LM and *HOME_DIR*/RDM_LM/ftok_files directories contain the token files used to form the keys that are used to create and access the IPC resources necessary for the dmlockmgr to communicate with its clients, its standard output file, and the transaction file.

The dmlockmgr token files have the form shown in Table 12-1 on page 298.

Table 12-1 `dmlockmgr` Token Files

File	Description
HOME_DIR/RDM_LM/dmlockmgr	Used by the `dmlockmgr` and its clients to access `dmlockmgr`'s semaphore and input message queue
HOME_DIR/RDM_LM/ftok_files/ft*nnnn*	Preallocated token files that are not currently in use. As processes attempt to connect to `dmlockmgr`, these files will be used and renamed as described below. *nnnn* is a four digit number from 0000 through 0099.
HOME_DIR/RDM_LM/ftok_files/ft*nnnn.xxxpid*	The renamed version of the preallocated token files. *nnnn* is a four digit number from 0000 through 0099. *xxx* is a three-character process identifier with the following meaning: • `atr` = `dmatread` • `ats` = `dmatsnf` • `cat` = `dmcatadm` • `ddb` = `dmdadm` • `dmd` = `dmfdaemon` • `dmv` = `dmmove` • `hde` = `dmhdelete` • `lfs` = `dmloadfs` • `lib` = `dmatls` • `sel` = `dmselect` • `vol` = `dmvoladm` *pid* is the numeric process ID of the process connected to `dmlockmgr`.

The IPC resources used by DMF are always released during normal process exit cleanup. If one of the `dmlockmgr` client processes dies without removing its message queue, `dmlockmgr` will remove that queue when it detects the death of the client. The token files themselves are periodically cleaned up by the `dmlockmgr` process.

Note: Normally, the `dmlockmgr` process is terminated as part of normal shutdown procedures. However if you wish to stop `dmlockmgr` manually, you must use the following command:

`/usr/sbin/dmclripc -u dmlockmgr -z` *HOME_DIR*`/RDM_LM`

This command will do all of the necessary IPC resource and token file maintenance.

If the `dmlockmgr` process aborts, all DMF processes must be stopped and restarted in order to relogin to a new `dmlockmgr` process. If the `dmfdaemon` or `dmatls` processes abort during a period when the `dmlockmgr` has died, when they restart they will attempt to restart the `dmlockmgr`. The new `dmlockmgr` process will detect existing DMF processes that were communicating with the now-dead copy of `dmlockmgr`, and it will send a termination message to those DMF processes.

The `dmlockmgr` maintains a log file that is named as follows, where *yyyy*, *mm*, and *dd* are the year, month, and day:

HOME_DIR`/RDM_LM/dmlocklog.`*yyyymmdd*

The log file is closed and a new one opened at the first log request of a new day. These files are not typically large files, but a new file will be created each day. These log files are removed via the `run_remove_log.sh` daemon task command. For more information about `run_remove_log.sh`, see "`taskgroup` Object" on page 170.

`dmlockmgr` Individual Transaction Log Files

The individual transaction log files have the following form:

prefix`.log`

where *prefix* is the same format as the token file name described in Table 12-1 on page 298 as `ft`*nnnn.xxxpid*. The prefix associates a log file directly with the token file of the same name.

Most of these log files will be created in the *HOME_DIR* under the daemon's and library servers' subdirectories. In almost all cases, the processes that create these log files will remove them when they exit. However, if a process terminates abnormally, its log file may not be removed. Transaction log files can sometimes become quite large, on the order of 10's of Mbytes. Most of these orphaned log files will be removed by the daemon as part of its normal operation.

Several DMF commands allow accessing copies of database files in places other than the *HOME_DIR*. If an orphaned log is encountered in a location other than in the *HOME_DIR*, it may be removed after it is clear that it is no longer in use. In order to verify that it is no longer in use, search the *HOME_DIR*/RDM_LM/ftok_files directory for a file with the same name as the prefix of the log file. If no such ftok_files file exists, it is safe to remove the log file.

The transaction activity file, *HOME_DIR*/RDM_LM/vista.taf, is the transaction log file that contains information about active transactions in the system. It is used to facilitate automatic database transaction processing.

Caution: Do **not** delete the *HOME_DIR*/RDM_LM/vista.taf file.

Media-Specific Processes and Library Servers

Media-specific processes (MSPs) and library servers (LSs) migrate files from one media to another:

- The file transfer protocol (FTP) MSP allows the DMF daemon to manage data by moving it to a remote machine.
- The disk MSP migrates data to a directory that is accessible on the current systems.
- The disk cache manager (DCM) MSP migrates data to a cache disk.
- The tape LS copies files from a disk to a tape or from a tape to a disk. The LS can manage multiple active copies of a migrated file. The LS contains of one or more volume groups (VGs). When a file is migrated from disk to tape, the selection policy can specify that it be copied to more than one VG. Each VG can manage at most one copy of a migrated file. Each VG has an associated pool of tapes. Data from more than one VG is never mixed on a tape.

This chapter discusses the following:

LS Operations

The LS consists of the following programs:

- `dmatls`
- `dmatwc`
- `dmatrc`

The DMF daemon executes `dmatls` as a child process. In turn, `dmatls` executes `dmatwc` (the write child) to write data to tape and `dmatrc` (the read child) to read data from tape.

The `dmatls` program maintains the following records in the LS database:

- Catalog (CAT) records, which contain information about the files that the LS maintains
- Volume (VOL) records, which contain information about the media that the LS uses

The database files are not text files and cannot be updated by standard utility programs. Detailed information about the database files and their associated utilities is provided in "CAT Records" on page 306, and "VOL Records" on page 306.

The LS provides a mechanism for copying active data from volumes that contain largely obsolete data to volumes that contain mostly active data. This process is referred to as *volume merging*. Data on LS volumes becomes obsolete when users delete or modify their files. Volume merging can be configured to occur automatically (see "LS Tasks" on page 240). It can also be triggered by marking LS volumes as sparse with the `dmvoladm`(8) command.

The LS provides two utilities that read LS volumes directly:

- `dmatread`(8), which copies all or part of a migrated file to disk
- `dmatsnf`(8), which audits and verifies LS volumes

This section discusses the following:

LS Directories

Each instance of the LS needs three types of directories, one for each of the following:

- Database files for CAT and VOL records
- Database journal files
- Log files

Sites define the location of these directories by editing the base object configuration file parameters `HOME_DIR`, `JOURNAL_DIR`, and `SPOOL_DIR`, whose values are referred to as *HOME_DIR*, *JOURNAL_DIR*, and *SPOOL_DIR* in this document. A given instance of the LS creates a subdirectory named after itself in each of these three directories.

For example, if an instance of the LS is called `cart1`, its database files reside in directory *HOME_DIR*`/cart1`. If another instance of the LS is called `cart2`, its database files reside in *HOME_DIR*`/cart2`. If an instance of the LS is called `cart3`, its database files reside in *HOME_DIR*`/cart3`.

Similarly, LS `cart1` stores its journal files in directory *JOURNAL_DIR*`/cart1` and its log files and other working files in *SPOOL_DIR*`/cart1`.

Media Concepts

The LS takes full advantage of the capabilities of modern tape devices, including data compression and fast media positioning. To accommodate these capabilities and to

provide recovery from surface or other media defects, `dmatls` uses a number of structural concepts built on top of traditional tape structure.

The components are as follows:

- The *block* is the basic structural component of most tape technologies. It is the physical unit of I/O to and from the media. The optimal block size varies with the device type. For example, the default block size for an STK T10000A tape drive is 524288 bytes.
- A *chunk* is as much or as little of a user file as fits on the remainder of the tape (see Figure 13-1 on page 305). Thus, every migrated file has at least one, and sometimes many, chunks. Such a concept is necessary because the capacity of a volume is unknown until written, both because of natural variation in the medium itself and because the effect of data compression varies with the data contents.
- A *zone* is a logical block containing many physical blocks ending with a tape mark. A zone has a target size that is configurable by media type. The default zone size is 50000000 bytes.

  The VG writes chunks into the zone until one of three conditions occurs:

  – The zone size is exceeded

  – The VG exhausts chunks to write

  – The end of tape is encountered

  Thus, the actual zone size can vary from well below the target size to the entire tape volume. A zone never spans physical volumes.

  The zone plays several roles:

  – The zone size is the amount of data that triggers `dmatls` to start a process to write files to tape.

  – The LS maintains the beginning of each zone in its database. This allows the LS to use fast hardware positioning functions to return to the beginning, so that it can restore the chunks in that zone.

  Because getting the tape position and writing a tape mark can be very costly, the concept of a zone and the target size provides a way to control the trade offs between write performance, safety, and recall speed.

Figure 13-1 illustrates the way files are distributed over chunks, zones, and volumes, depending upon the file size. The tape with volume serial number (VSN) `VOL001` has

two zones and contains six files and part of a seventh. The tapes with VSNs `VOL002` and `VOL003` contain the rest of file `g`. Notice that on `VOL001` file `g` is associated with chunk 7, while on the other two tapes it is associated with chunk 1. File `g` has three VSNs associated with it, and each tape associates the file with a chunk and zone unique to that tape.

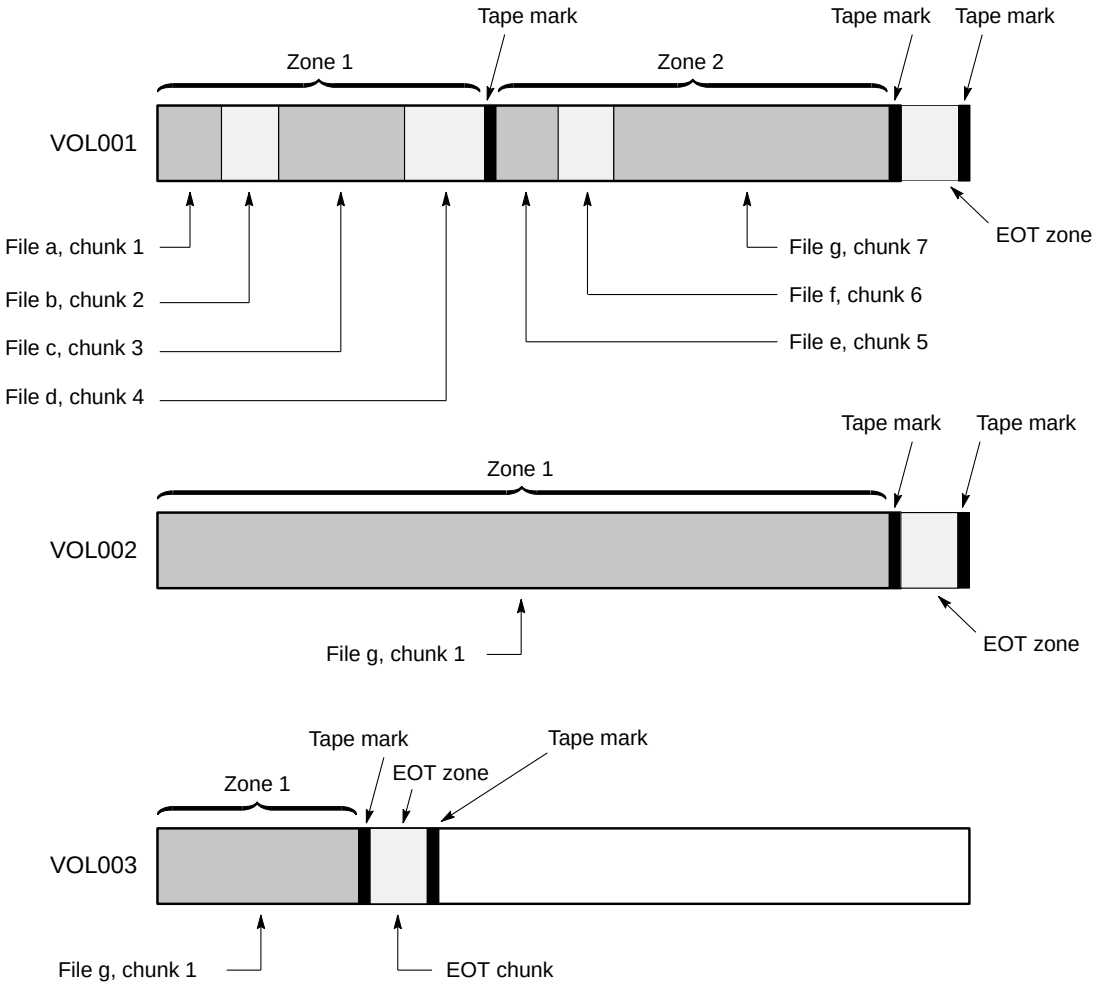

Figure 13-1 Media Concepts

CAT Records

Catalog (CAT) records store the location of each file chunk in terms of its volume, zone, and chunk number. The key for these records is the file's bit-file identifier (BFID).

Note: You do not explicitly create CAT records; they are created when files migrate.

There are the following files:

CAT Files	Description
`tpcrdm.dat`	Contains the catalog data records
`tpcrdm.key1.keys`, `tpcrdm.key2.keys`	Contains the indexes to the catalog data

The `libsrv_db.dbd` LS database definition file in the same directory describes the CAT record files and their record structure.

All files are non-ASCII and cannot be maintained by standard utility programs. The `dmcatadm` command provides facilities to create, query, and modify CAT records (see "`dmcatadm` Command" on page 313).

Note: The ability to create or modify CAT records with `dmcatadm` is provided primarily for testing or error recovery purposes. In the normal course of operations, you would never use this capability.

VOL Records

Volume (VOL) records in the LS database contain information about each volume that exists in the pool of tapes to be used by `dmatls`. These records are indexed by the volume serial number (VSN) of each volume and contain information such as the following:

- Volume's type
- Estimated capacity
- Label type
- A number of flags indicating the state of the volume

- VG or allocation group

Note: Unlike CAT records, you must create VOL records before using `dmatls` for the first time.

There are the following files:

VOL Files	Description
`tpvrdm.dat`	Contains the volume data records
`tpvrdm.vsn.keys`	Contains the indexes to the volume data records

The `libsrv_db.dbd` LS database definition file in the same directory describes the VOL record files and their record structure.

The files contain binary data and require special maintenance utilities. The `dmvoladm` command, described in more detail in "`dmvoladm` Command" on page 322, provides facilities to create, query, and modify VOL records. Additional database maintenance utilities are described in "Database Recovery" on page 360.

Note: If you have more than one instance of a VG, you must ensure that the volume sets for each are mutually exclusive.

LS Journals

Each instance of `dmatls` protects its database by recording every transaction in a journal file. Journal file pathnames have the following format:

JOURNAL_DIR`/`*ls_name*`/libsrv_db.`*yyyymmdd[.hhmmss]*

The LS creates journal files automatically.

Existing journal files are closed and new ones created in two circumstances:

- When the first transaction after midnight occurs
- When the journal file reaches the size defined by the `JOURNAL_SIZE` configuration parameter

When the first transaction after midnight occurs, the existing open journal file is closed and the suffix `.235959` is appended to the current file name no matter what

the time (or date) of closing. The closed file represents the last (or only) transaction log of the date *yyyymmdd*. A new journal file with the current date is then created.

When the journal file reaches JOURNAL_SIZE, the file is closed and the suffix *.hhmmss* is added to the name; *hh*, *mm*, and *ss* represent the hour, minute, and second of file closing. A new journal file with the same date but no time is then created.

For example, the following shows the contents of a *JOURNAL_DIR/ls_name* directory on 15 June 2004:

```
libsrv_db.20040527.235959  libsrv_db.20040606.235959
libsrv_db.20040528.235959  libsrv_db.20040607.235959
libsrv_db.20040529.235959  libsrv_db.20040608.235959
libsrv_db.20040530.235959  libsrv_db.20040609.235959
libsrv_db.20040531.235959  libsrv_db.20040610.235959
libsrv_db.20040601.235959  libsrv_db.20040611.235959
libsrv_db.20040602.235959  libsrv_db.20040612.235959
libsrv_db.20040603.235959  libsrv_db.20040613.235959
libsrv_db.20040604.235959  libsrv_db.20040614.235959
libsrv_db.20040605.235959  libsrv_db.20040615
```

For every date on which LS database transactions occurred, there will exist a file with that date and the suffix .235959, with the exception of an existing open journal file. Some dates may have additional files because the transaction log reached JOURNAL_SIZE at a specified time and the file was closed.

You can configure daemon_tasks parameters to remove old journal files (using the run_remove_journals.sh task and the JOURNAL_RETENTION parameter. For more information, see "taskgroup Object" on page 170.

If an LS database becomes corrupt, recovery consists of applying the journal files to a backup copy of the database.

LS Logs

All DMF MSPs and LSs maintain log files named msplog.*yyyymmdd* in the MSP/LS spool directory which, by default, is *SPOOL_DIR/mspname*. *SPOOL_DIR* is configured in the base object of the configuration file; *mspname* is the name of the MSP/LS in the daemon object of the configuration file; *yyyymmdd* is the current year, month, and day.

These log files are distinct from the logs maintained by the DMF daemon; however, some of the messages that occur in the daemon log are responses that the MSP/LS generates. The content of the log is controlled by the MESSAGE_LEVEL configuration

parameter. For a description of the levels of logging available, see the `dmf.conf`(5) man page.

The `msplog.`*yyyymmdd* file is the primary log for the LS and contains most of the messages. This file is written by `dmatls`. In addition, `dmatrc` and `dmatwc` create a `moverlog.`*yyyymmdd* log file each day in the subdirectory `moverlogs/`*hostname*.

This section describes informational statistics provided by the tape log files. These messages appear in the *SPOOL_DIR*`/`*msp_name*`/msplog.`*yyyymmdd* files. Timing information provided (such as MB transferred per second) should not be used as an accurate benchmark of actual data transfer rates. This information is provided for monitoring DMF and should only be used in comparison to similar data provided by DMF. Text in all uppercase references a parameter defined in the DMF configuration file. You can reference the comments in the sample configuration file or in the `dmf.conf`(5) man page for a more detailed definition of these parameters.

Note: Because the LS will continue to create log files and journal files without limit, you must remove obsolete files periodically by configuring the `run_remove_logs.sh` and `run_remove_journals.sh` tasks in the configuration file, as described in "`taskgroup` Object" on page 170.

Example 13-1 LS Statistics Messages

The following is an example of LS statistics messages taken from an `msplog.`*yyyymmdd* file. These messages are automatically and periodically issued by the LS.

```
08:46:00:404-I      zap   237076-dmatls vg1.stats: children=2/2/0/2, btp=672617104/527956913/0, wc=1/2, cwc=?
08:46:00:404-I      zap   237076-dmatls vg2.stats: children=0/0/0/2, btp=0/0/0, wc=0/2, cwc=?
08:46:00:404-I      zap   237076-dmatls vg1.stats: data put=92957.718 mb, data recalled=24964.680 mb
08:46:00:404-I      zap   237076-dmatls vg2.stats: data put=1239.537 mb, data recalled=1120.492 mb
08:46:00:404-I      zap   237076-dmatls vg1.stats:     Put_File -      0   8900      0    282
08:46:00:404-I      zap   237076-dmatls vg1.stats:     Get_File -      0   1809      0      0
08:46:00:404-I      zap   237076-dmatls vg1.stats:  Delete_File -      0 107618      0      0
08:46:00:404-I      zap   237076-dmatls vg1.stats:   Cancel_Req -      0    282      0      0
08:46:00:404-I      zap   237076-dmatls vg1.stats:     Flushall -      0      5      0      0
08:46:00:404-I      zap   237076-dmatls vg1.stats:        Merge -     44      0      0      0
08:46:00:404-I      zap   237076-dmatls vg2.stats:     Put_File -      0   1850      0    211
08:46:00:404-I      zap   237076-dmatls vg2.stats:     Get_File -      0     68      0      0
08:46:00:404-I      zap   237076-dmatls vg2.stats:  Delete_File -      0      4      0      0
08:46:00:404-I      zap   237076-dmatls vg2.stats:   Cancel_Req -      0    211      0      0
08:46:00:404-I      zap   237076-dmatls vg2.stats:     Flushall -      0      1      1      0
08:46:00:404-I      zap   237076-dmatls vg1.stats: mc=2, ms=2000000000, mu=679346176, sm=0
```

The information provided by these entries is defined as follows:

- `children=2/2/0/2` represents the total child processes (`2`), the active child processes (`2`), the clean processes running (`0`), and the current maximum number of children the VG may have (`2`). Clean children are used when a `dmatrc` or `dmatwc` process dies without cleaning up.
- `btp=672617104/527956913/0` represents the bytes queued for putting (`672617104`), the threshold at which to start the next put child (`527956913`), and the bytes assigned to socket I/O (`0`)
- `wc=1/2` represents the active write child processes (`1`) and the configured value of `MAX_PUT_CHILDREN` (`2`)
- `cwc=?` represents the host name and process ID of the current write child (that is, the write child that is accepting data to write). `?` represents none.

The next set of lines gives the total amount of data put (such as `92957.718` MB) and recalled (such as `24964.680` MB).

The next set of six lines provide statistics for each type of VG request. Statistics information is provided only for requests that have been issued since the LS was started. These lines have the following format:

request_name active successful errors canceled

active represents the number of requests not yet completed; *successful* represents the number of successfully completed requests; *error* represents the number of requests that completed with errors; *canceled* represents the number of canceled requests.

The last set of lines provide the following information:

- `mc` is the configured value for `MERGE_CUTOFF`, the cutoff to stop scheduling tapes for merging (such as `2`)
- `ms` is the configured value for `CACHE_SPACE`, the merge cache space available (such as `2000000000` bytes)
- `mu` is the merge cache space used (such as `679346176` bytes)
- `sm` is the number of socket merge children (`0`)

The LS write child (`dmatwc`) and read child (`dmatrc`) also produce statistics messages in the `moverlog` file. These messages contain timing statistics whose format changes from release to release, and they are not documented in this manual.

Volume Merging

When users delete or modify their migrated files, the copy on tape becomes obsolete. Over time, some volumes will become entirely empty and can be reused. However, most volumes experience a gradual increase in the ratio of obsolete data to active data; such volumes are said to be *sparsely populated* or *sparse*. To reclaim the unused space on these volumes, DMF provides a *volume merge* facility, which copies the active data from several sparse volumes to a new volume, thus freeing the sparse volumes for reuse. Volume merging can be configured to occur automatically by using the `run_merge_tapes.sh` or `run_merge_mgr.sh` tasks (see "LS Tasks" on page 240).

Volume merging can also be done manually. `dmatls` performs merge operations whenever sparse volumes and the necessary resources exist at the same time. Use the `dmvoladm select` directive to mark VG volumes as sparse. (The `select` directive is described in "`dmvoladm` Command" on page 322.) Because the merge processing occurs simultaneously with other DMF activities, it is easiest to configure DMF to

automatically perform merges at night or during other periods of relatively low activity.

The `dmatls` utility can perform volume-to-volume merging. Volume-to-volume merging is accomplished by moving data across a socket connection between the LS tape read-child and the LS tape write-child. The benefit of using a socket to transfer data between volumes is that you do not have to reserve disk space. The drawback to using a socket for data transfer is the cost of linking the process that performs the read with the process that performs the write.

In busy environments that have heavy contention for tape drives, the close coupling between the socket's tape reader and tape writer can be costly, especially when short files are being transferred. For large files, the overhead and possible delays in waiting for both tapes to be mounted is small compared to the benefit of rapid transfer and zero impact on free disk space. For this reason, you can move small files through a disk cache and big files through a socket. This process is mediated by the following configuration parameters:

Parameter	Description
`CACHE_SPACE`	Specifies the amount of disk space that will be used to temporarily store chunks during a merge operation.
`CACHE_DIR`	Specifies the directory into which the LS will store chunks while merging them from sparse tapes. If `CACHE_DIR` is not specified, `TMP_DIR` is used.
`MAX_CACHE_FILE`	Specifies the largest chunk that will be stored temporarily on disk during a merge operation.
`MERGE_CUTOFF`	Specifies the number of child processes after which the VG will stop scheduling tapes for merging. This number is the sum of the active and queued children generated from gets, puts, and merges.

Using a small amount of disk space to hold small chunks can have a significant impact on the total time required to perform merges. The default configuration options are set to move 100% of merge data across sockets.

Note: It is important to avoid volume merging on more than one VG simultaneously if they share a tape device. If you initiate a merge process on more than one VG on the same device at the same time (either by entering the same time in the DMF configuration file or by triggering the process manually), both processes will compete for tape transports. When a limited number of tape transports are available, a deadlock can occur. If you chose not to configure DMF to perform merges automatically by configuring the `run_merge_tape.sh` or `run_merge_mgr.sh` tasks, ensure that your `cron` jobs that automatically initiate volume merging refrain from initiating a second merge process until after all previously initiated merges are complete. You can accomplish this by using the `dmvoladm` command within the `cron` job to check for tapes that have the `hsparse` flag, as shown in the following example for an LS with two VGs:

```
tapes=$(dmvoladm -m ls -c "count hsparse")
if [[ -z "$tapes" ]]; then
        # start merge on vg2
        dmvoladm -m ls -c "select hfull and threshold<=30 and vg=vg2"
fi
```

`dmcatadm` Command

The `dmcatadm`(8) command provides maintenance services for CAT records.

When you are inside the `dmcatadm` interface (that is, when you see the `adm` *command_number* `>` prompt), the command has a 30–minute timeout associated with it. If you do not enter a response within 30 minutes of the prompt having been displayed, the `dmcatadm`session terminates with a descriptive message. This behavior on all the database administrative commands limits the amount of time that an administrator can lock the daemon database and the LS database from updates.

Note: Most of these facilities, especially the ability to create and modify CAT records in the LS database, are intended primarily for testing or error recovery purposes.

`dmcatadm` Directives

The `dmcatadm` command executes directives from `stdin` or from the command line when you use the `-c` option. All directives start with a directive name followed by one or more parameters. Parameters may be positional or keyword-value pairs,

depending on the command. White space separates the directive name, keywords, and values.

The `dmcatadm` directives are as follows:

Directive	Description
`count`	Displays the number of records that match the expression provided.
`create`	Creates a CAT record.
`delete`	Deletes the specified CAT records.
`dump`	Prints the specified CAT records to standard out in ASCII; each field is separated by the pipe character (`|`).
`help`	Displays help.
`list`	Shows the fields of selected CAT records. You may specify which fields are shown.
`load`	Applies records to the LS database obtained from running the `dump` directive.
`quit`	Stops program execution after flushing any changed database records to disk. The abbreviation `q` and the string `exit` produce the same effect.
`set`	Specifies the fields to be displayed in subsequent `list` directives.
`update`	Modifies the specified CAT records.
`verify`	Verifies the LS database against the daemon database.

The first parameter of most directives specifies the records to manipulate, and the remaining parameters are keyword-value pairs.

The syntax for the `dmcatadm` directives is summarized as follows:

`count` *selection* [*limit*]
`create` *bfid settings* `...`
`delete` *selection* [*limit*]
`dump` *selection* [*limit*]
`help`
`list` *selection* [*limit*] [*format*]
`load` *filename*
`quit (or q or exit)`
`set` [*format*]
`update` *selection* [*limit*] `to` *settings*`...`
`verify` *selection* [*entries*] [*vgnames*] [*limit*]

The parameters are as follows:

- The *selection* parameter specifies the records to be acted upon. The value for *selection* can be one of the following:
  - A *bfid* or range of *bfids* in the form *bfid* [-] [*bfid*]. *bfid*- specifies all records starting with *bfid*, and -*bfid* specifies all records up to *bfid*.
  - The keyword `all`
  - A period (`.`), which recalls the previous selection
  - An expression involving any of the above, field value comparisons, `and`, `or`, or parentheses.

  A field value comparison may use the following to compare a field keyword to an appropriate value:

  < (less than),
  > (greater than)
  = (equal to)
  != (not equal to)
  <= (less than or equal to)
  >= (greater than or equal to)

  The syntax for *selection* is as follows:

  selection ::= *or-expr*
  or-expr ::= *and-expr* [or *or-expr*]
  and-expr ::= *nested-expr* [and *or-expr*]
  nested-expr ::= *comparison* | (*or-expr*)
  comparison ::= *key-range* | *field-keyword op field-value*
  op ::= < | > | = | != | <= | >=
  bfid-range ::= *bfid* [- *bfid*] | [*bfid* - [*bfid*]] | *key-macro*
  key-macro ::= all
  field-keyword ::= *name or abbreviation of the record field*
  field-value ::= *appropriate value for the field*
  key ::= *character representation of the record bfid*

  Thus valid *selection*s could be any of the following:

```
305c74b200000010-305c74b200000029
7fffffff000f4411-
-305c74b2000004c8
all
```

```
chunkoffset>0
chunknumber>0 and writeage<5d
. and writeage>4d
vsn=S07638
```

- The *limit* parameter restricts the records acted upon.
- The *bfid* parameter for the `create` directive specifies the bit-file-identifier (BFID) for the record being created. The value for *bfid* may be a bit-file identifier (BFID) designator in the form of a hexadecimal number.
- The *settings* parameter for the `create` and `update` directives specify one or more fields and their values.
- The *format* parameter selects the way in which output is displayed. Any program or script that parses the output from this command should explicitly specify a format; otherwise the default is used, which may change from release to release.
- The *entries* parameter specifies a file of daemon database entries.
- The *vgnames* parameter specifies the names of the VGs associated with the records.

`dmcatadm` Keywords

You can use the *field* keywords listed below as part of a *selection* parameter to select records, in a *format* parameter, or in a *settings* parameter to specify new values for a field; in that case, you must specify a keyword-value pair. A keyword-value pair consists of a keyword followed by white space and then a value. When specifying new values for fields, some of the keywords are valid only if you also specify the `-u` (unsafe) option. The abbreviation for each of the keywords is given in parenthesis following its name.

Keyword	Description
`cflags` (`cf`)	For future use.
`chunkdata` (`cd`)	Specifies the actual number of bytes written to tape by the VG for the chunk. In the case of sparse files, this field will be smaller than `chunklength`. This is valid only in unsafe (`-u`) mode.
`chunklength` (`cl`)	The size of the chunk in bytes; an integer. This is valid only in unsafe (`-u`) mode.

`chunknumber` (`cn`)	The ordinal of the chunk on its volume. For example, 1 if the chunk is the first chunk on the volume, 2 if it is the second, and so on. Not valid as part of a *settings* parameter in an `update` directive.
`chunkoffset` (`co`)	The byte offset within the file where the chunk begins; an integer. For example, the first chunk of a file has `chunkoffset` 0. If that first chunk is 1,000,000 bytes long, the second chunk would have `chunkoffset` 1000000. This is valid only in unsafe (`-u`) mode.
`chunkpos` (`cp`)	The block offset within the zone where the chunk begins — a hexadecimal integer. For example, the first chunk in a zone has `chunkpos` 1. A value of 0 means unknown. Valid only in unsafe (`-u`) mode.
`filesize` (`fs`)	The original file size in bytes, an integer. This is valid only in unsafe (`-u`) mode.
`readage` (`ra`)	The date and time when the chunk was last read; the same as `readdate`, except specified as *age*.
`readcount` (`rc`)	The number of times the chunk has been recalled to disk; an integer.
`readdate` (`rd`)	The date and time when the chunk was last read, an integer that reflects raw UNIX or Linux time.
`volgrp` (`vg`)	The VG name. This keyword is valid for LSs only. This keyword is not valid as part of a *settings* parameter.
`vsn` (`v`)	The volume serial numbers; a list of one or more 6-character alphanumeric volume serial numbers separated by colons (`:`). This keyword is not valid as part of a *settings* parameter in an `update` directive.
`writeage` (`wa`)	The date and time when the chunk was written; the same as `writedate`, except specified as *age*. This is valid only in unsafe (`-u`) mode.
`writedate`(`wd`)	The date and time when the chunk was written, an integer that reflects raw UNIX or Linux time. This is valid only in unsafe (`-u`) mode.

`zoneblockid` (`zb`)	Allows just the block ID portion of the `zonepos` to be displayed, returned, or changed. This is valid only in unsafe (`-u`) mode.
`zonenumber` (`zn`)	Allows just the zone number portion of the `zonepos` to be displayed, returned, or changed. This is valid only in unsafe (`-u`) mode.
`zonepos` (`zp`)	The physical address of the zone on the volume, expressed in the form *integer/hexadecimal-integer*, designating a zone number and block ID. A value of zero is used for *hexadecimal-integer* if no block ID is known. *integer* is the same as `zonenumber`, and *hexadecimal-integer* is the same as `zoneblockid`. This is valid only in unsafe (`-u`) mode.

The date field keywords (`readdate` and `writedate`) have a value of either `now` or raw UNIX or Linux time (seconds since January 1, 1970). These keywords display their value as raw UNIX or Linux time. The value comparison `>` used with the date keywords means newer than the value given. For example, `>36000` is newer than 10AM on January 1, 1970, and `>852081200` is newer than 10AM on January 1, 1997.

The age field keywords (`readage` and `writeage`) let you express time as *age* in a string in a form. They display their value as an integer followed by the following:

`w` (weeks)
`d` (days)
`h` (hours)
`m` (minutes)
`s` (seconds)

For example, `8w12d7h16m20s` means 8 weeks, 12 days, 7 hours, 16 minutes, and 20 seconds old.

The comparison `>` used with the age keywords means older than the value given (that is, `>5d` is older than 5 days).

The *limit* parameter in a directive limits the records acted upon. It consists of one of the following keywords followed by white space and then a value:

Keyword	**Description**
`recordlimit` (`rl`)	Limits the number of records acted upon to the value that you specify; an integer.

`recordorder` (`ro`)	Specifies the order that records are scanned; may be `key`, `vsn`, or `data`. `key` specifies that records are scanned in ascending order of the chunk key. `vsn` specifies that records are scanned in ascending order of the chunk VSN. `data` specifies that records are scanned in the order in which they are stored in the LS database, which is fastest but essentially unordered.

The following keywords specify files of daemon database entries:

Keyword	Description
`entries` (`e`)	Specifies a file of daemon database entries. This keyword applies to the `verify` directive and consists of the word `entries` (or its abbreviation `e`) followed by a string.
`vgnames` (`vn`)	Specifies the names of the VGs associated with the record. This keyword applies to the `verify` directive and consists of the word `vgnames` (or its abbreviation `vn`) followed by a quoted, space-separated list of names.

The *format* parameter in a directive consists of the word `format` followed by white space and then either the word `default`, the word `keyword`, or a list of field keywords.

The keyword form, intended for parsing by a program or script, suppresses the headings.

If a list of field keywords is used in the *format* parameter, they may be delimited by colons or spaces, but spaces will require the use of quoting.

Note: The BFID is always included as the first field and need not be specified.

For any field that takes a byte count, you may append the letter `k`, `m`, or `g` (in either uppercase or lowercase) to the integer to indicate that the value is to be multiplied by one thousand, one million, or one billion, respectively.

For information about the role of the `dmcatadm` (8)command in database recovery, see "Database Recovery" on page 360.

Example 13-2 `dmcatadm list` Directive

The following is sample output from the `dmcatadm list` directive. The file with key `3273d5420001e244` has two chunks because it spans two physical tape volumes; the first chunk contains bytes 0 through 24821759, and the second chunk bytes 24821760 (the `CHUNK OFFSET`) to the end of the file.

```
adm 3>list 3273d5420001e242- recordlimit 10
                          WRITE     CHUNK       CHUNK   CHUNK
                    KEY     AGE    OFFSET      LENGTH     NUM       VSN
-----------------------------------------------------------------------
       3273d5420001e242     61d         0    77863935      13    S12940
       3273d5420001e244     61d         0    24821760     168    S12936
       3273d5420001e244     61d  24821760    23543808       1    S12945
       3273d5420001e245     61d         0    51019776       2    S12945
       3273d5420001e246     61d         0    45629440      59    S12938
       3273d5420001e247     61d         0    35586048      60    S12938
       3273d5420001e248     61d         0     9568256       3    S12944
       3273d5420001e249     61d         0    14221312       4    S12944
       3273d5420001e24a     61d         0      458752       5    S12944
       3273d5420001e24b     61d         0    14155776       6    S12944
```

The following is sample output from the `dmcatadm list` directive for an LS. The file with key `3b4b28f2000000000000ae80` has 2 chunks because it was migrated to two different VGs within this LS. The output from the `dmvoladm list` directive that follows shows that `VSN 000700` is assigned to the VG named `vg8a15`, and `VSN 00727` is assigned to the VG named `vg8a05`.

```
# dmcatadm -m ls1
adm 1>list 3b4b28f2000000000000ae80- recordlimit 4
                          WRITE       CHUNK       CHUNK CHUNK
                     KEY    AGE      OFFSET      LENGTH   NUM VSN
-----------------------------------------------------------------
3b4b28f2000000000000ae80     1d           0     2305938   120 000700
3b4b28f2000000000000ae80     4d           0     2305938    32 000727
3b4b28f2000000000000ae82     1d           0      234277   247 003171
3b4b28f2000000000000ae82     1d           0      234277   186 003176

adm 2> quit

# dmvoladm -m ls1
adm 1>list vsn=000700
                                          DATA    EOT   EOT             WR/FR
VSN      VOLGRP LB      DATA LEFT        WRITTEN  CHUNK  ZONE   HFLAGS      AGE
-------------------------------------------------------------------------------
000700   vg8a15 al     150.280473     233.786093    123     9  ------u--     1d
adm 2>list vsn=000727
                                          DATA    EOT   EOT             WR/FR
VSN      VOLGRP LB      DATA LEFT        WRITTEN  CHUNK  ZONE   HFLAGS      AGE
-------------------------------------------------------------------------------
000727   vg8a05 al     159.107337     200.443980    102     6  ---------     1d
```

dmcatadm Text Field Order

The text field order for chunk records generated by the dmdump(8), dmdumpj(8), and the dump directive in dmcatadm is listed below. This is the format expected by the load directives in dmcatadm:

1. C (indicates the chunk record type)
2. bfid (hexadecimal digits)
3. filesize
4. writedata
5. readdate
6. readcount

7. chunkoffset
8. chunklength
9. chunkdata
10. chunknumber
11. flags (in octal)
12. zoneposition (zonenumber / zoneblockid) (in hexadecimal)
13. vsn
14. chunkpos (in hexadecimal)

dmvoladm Command

The dmvoladm(8) command provides maintenance services for VOL records. In addition to the creation and modification of volume records, dmvoladm has an important role in the recovery of VOL records from an LS database checkpoint and is the mechanism that triggers volume merge activity.

When you are inside the dmvoladm interface (that is, when you see the adm *command_number* > prompt), the command has a 30–minute timeout associated with it. If you do not enter a response within 30 minutes of the prompt having been displayed, the dmvoladm session terminates with a descriptive message. This behavior on all the database administrative commands limits the amount of time that an administrator can lock the daemon database and the LS database from updates.

dmvoladm Directives

The dmvoladm command executes directives from stdin or from the command line when you use the -c option. The syntax is the same as for dmcatadm: a directive name followed by parameters or paired keywords and values, all separated by white space.

Directive	Description
count	Displays the number of records that match the expression provided.
create	Creates a VOL record.
delete	Deletes the specified VOL records.

`dump`	Prints the specified VOL records to standard output in ASCII. Each field is separated by the pipe character (|).
`help`	Displays help.
`list`	Shows the fields of selected VOL records. You may specify which fields are shown.
`load`	Applies VOL records to the LS database obtained from running the `dump` directive.
`quit`	Stops program execution after flushing any changed records to disk. The abbreviation `q` and the string `exit` produce the same effect.
`repair`	Causes `dmvoladm` to adjust the usage information for specified volumes based on CAT records in the LS database. This directive is valid only in unsafe (`-u`) mode.
`select`	Marks selected volumes as being sparse. Equivalent to `update` *expression* `to hsparse on`.
`set`	Specifies the fields to be shown in subsequent `list` directives.
`update`	Modifies the specified VOL records.
`verify`	Verifies the LS database against the daemon database.

The syntax for the `dmvoladm` directives is summarized as follows:

```
count [limit]
create vsnlist volgrpspec [settings]
delete selection [limit]
dump selection [limit]
help
list selection [limit] [format]
load filename
quit (or q, or exit)
repair selection
select selection [limit]
set format
update selection [limit] to settings
verify selection
```

The *volgrpspec* parameter consists of the keyword `volgrp` (or `vg`), followed by a value for that keyword.

The value for *vsnlist* may be a single 6-character volume serial number (VSN) or a range of VSNs separated by the hyphen (-) character. A VSN string is case insensitive and may consist entirely of letters, entirely of digits, or a series of letters followed by digits. In a range of VSNs, the first must be lexically less than the second.

The value for *selection* may be one of the following:

- A *vsnlist* or range of VSNs in the form *vsn*[-*vsn*]. *vsn*- specifies all records starting with *vsn*, and -*vsn* specifies all records up to *vsn*.
- A period (.), which recalls the previous selection
- The name of one of the flags in the keyword list that follows in this section.
- One of the words `all`, `used`, `empty`, or `partial` or any of the *hold flags (hflags)*, whose meanings are as follows:

Flag	Description
`all`	Specifies all volumes in the LS database
`empty`	Specifies all volumes in which data left is `0`
`partial`	Specifies used volumes in which `hfull` is `off`
`used`	Specifies all volumes in which data written is not `0`

- An expression involving *vsnlists*, field-value comparisons, `and`, `or`, or parentheses.

  A field value comparison may use the following to compare a field keyword to an appropriate value:

  `<` (less than)
  `>` (greater than)
  `=` (equal)
  `!=` (not equal)
  `<=` (less than or equal to)
  `>=` (greater than or equal to)

The syntax for *selection* is as follows:

```
selection     ::=   or-expr
or-expr       ::=   and-expr [ or or-expr ]
and-expr      ::=   nested-expr [ and or-expr ]
nested-expr   ::=   comparison | ( or-expr )
comparison    ::=   vsnlist | field-keyword op field-value
op            ::=   < | > | = | != | >= | <=
```

vsn-range ::= *vsn* [- *vsn*] | [*vsn* - [*vsn*]] | *key-macro*
key-macro ::= all | empty | used | partial | *flag(s)*
field-keyword ::= *name or abbreviation of the record field*
field-value ::= *appropriate value for the field*
vsnlist ::= *character representation of the volume serial number*

Thus valid *selections* could be any of the following:

```
tape01-tape02
tape50-
-vsn900
all
hoa or hro
used and hfull=off
datawritten>0 and hfull=off
. and eotchunk>3000 and (eotchunk<3500 or hfree=on)
hfull and threshold<30
```

dmvoladm Field Keywords

You can use the *field* keywords listed below as part of a *selection* parameter to select records, in a *format* parameter, or in a *settings* parameter to specify new values for a field; in that case, a keyword-value pair must be specified. A keyword-value pair consists of a keyword followed by white space and then a value. When specifying new values for fields, some of the keywords are valid only if you also specify the -u (unsafe) option:

Keyword	Description
blocksize (bs)	Specifies the data block size in bytes when the tape was first written; an integer. This keyword is used only when mounting tapes with existing valid data. When an empty tape is first written, the VG uses the default value for the tape type, unless it is overridden by a value in the BLOCK_SIZE parameter for the drive group in the DMF configuration file. This is valid only in unsafe (-u) mode.
chunksleft (cl)	Specifies the number of active chunks on the volume; an integer. This is valid only in unsafe (-u) mode.
dataleft (dl)	Specifies the number of bytes of active data on the volume. You specify this number as an integer, but for

	readability purposes it is displayed in megabytes (MB). This is valid only in unsafe (`-u`) mode.
`datawritten` (`dw`)	Specifies the maximum number of bytes ever written to the volume. You specify this number as an integer, but for readability purposes it is displayed in MB. This is valid only in unsafe (`-u`) mode.
`eotblockid` (`eb`)	Specifies the blockid of the chunk containing the end-of-tape marker; a hexadecimal integer. This is valid only in unsafe (`-u`) mode.
`eotchunk` (`ec`)	Specifies the number of the chunk containing the end-of-tape marker; an integer. This is valid only in unsafe (`-u`) mode.
`eotpos` (`ep`)	Specifies the absolute position of the end-of-tape marker zone in the form *integer/hexadecimal-integer*, designating a zone number and block ID. A value of zero is used for *hexadecimal-integer* if no block ID is known. *integer* the same as `eotzone`, and *hexadecimal-integer* is the same as `eotblockid`. This is valid only in unsafe (`-u`) mode.
`eotzone` (`ez`)	Specifies the number of the zone containing the end-of-tape marker; an integer. This is valid only in unsafe (`-u`) mode.
`hflags` (`hf`)	Specifies the flags associated with the record. See the description of *flags* keywords. Not valid as part of a *settings* parameter.
`label` (`lb`)	Specifies the label type: `al` for ANSI standard labels; `sl` for IBM standard labels; or `nl` for nonlabeled volumes. The default is `al`.
`tapesize` (`ts`)	Specifies the estimated capacity in bytes; an integer. The default is 215 MB.
`threshold` (`th`)	Specifies the ratio of `dataleft` to `datawritten` as a percentage. This field is valid only as part of a *selection* parameter.
`upage` (`ua`)	Specifies the date and time of the last update to the volume's database record. The same as for `update`, except that it is expressed as *age*. This is not valid as part of a *settings* parameter.

`update` (`ud`)	Specifies the date and time of the last update to the volume's database record, expressed as an integer that reflects raw UNIX or Linux time. This is not valid as part of a *settings* parameter.
`version` (`v`)	Specifies the DMF tape format version, an integer. This is valid only in unsafe (`-u`) mode.
`volgrp` (`vg`)	Specifies the VG or allocation group.
`wfage` (`wa`)	Specifies the date and time that the volume was written to or freed for reuse. The same as for `wfdate`, except that it is expressed as *age*. This is valid only in unsafe (`-u`) mode.
`wfdate` (`wd`)	Specifies the date and time that the volume was written to or freed for reuse, expressed as an integer that reflects raw UNIX or Linux time. This is valid only in unsafe (`-u`) mode.

The date field keywords (`update` and `wfdate`) have a value of either `now` or UNIX or Linux *raw time* (seconds since January 1, 1970). These keywords display their value as raw time. The value comparison `>` used with the date keywords means newer than the value given. For example, `>36000` is newer than 10AM on January 1, 1970, and `>852081200` is newer than 10AM on January 1, 1997.

The age field keywords (`upage` and `wfage`) let you express time as *age* as a string.

The age keywords display their value as an integer followed by the following:

`w` (weeks)
`d` (days)
`h` (hours)
`m` (minutes)
`s` (seconds)

For example, `8w12d7h16m20s` means 8 weeks, 12 days, 7 hours, 16 minutes, and 20 seconds old.

The comparison `>` used with the age keywords means older than the value given (that is, `>5d` is older than 5 days).

The *limit* parameter in a directive limits the records acted upon. It consists of one of the following keywords followed by white space and then a value. The abbreviation for the keyword is given in parentheses following its name, if one exists:

Keyword	Description
`datalimit` (no abbreviation)	Specifies a value in bytes. The directive stops when the sum of `dataleft` of the volumes processed so far exceeds this value.
`recordlimit` (`rl`)	Specifies a number of records; an integer. The directive stops when the number of volumes processed equals this value.
`recordorder` (`ro`)	Specifies the order that records are scanned; may be either `data` or `vsn`. `vsn` specifies that the records are scanned in ascending order of the chunk VSN. `data` specifies that the records are scanned in the order in which they are found in the LS database, which is fastest but essentially unordered.

The *format* parameter in a directive consists of the word `format` followed by white space and then either the word `default`, the word `keyword`, or a list of field and or flag keywords.

The `keyword` form, intended for parsing by a program or script, suppresses the headings.

If a list of field or flag keywords is used in the format expression, they may be delimited by colons or spaces, but spaces will require the use of quoting. The VSN is always included as the first field and need not be specified.

The *flag* keywords listed below can be used to change the settings of the *hold flags (hflags)*. They can also be used as part of selection or format parameters. :

Keyword	Description
`hbadmnt` (`hb`)	Indicates that the LS could not mount the tape. It is displayed as `--------b`.
`herr` (`he`)	Indicates an LS database inconsistency for this volume. It is displayed as `e--------`.
`hflags` (no abbreviation)	(Not valid as part of a *settings* parameter.) Shows the complete set of hold flags as a 9–character string. Each flag has a specific position and alphabetic value. If the flag is off, a dash (`-`) is displayed in its position; if the flag is on, the alphabetic character is displayed in that position.

`hfree` (no abbreviation)	Indicates that the volume has no active data and is available for reuse after `HFREE_TIME` has expired, displayed as `-f-------`. See the `dmf.conf`(5) man page for information about the `HFREE_TIME` configuration parameter. This is valid only in unsafe (`-u`) mode.
`hfull` (`hu`)	Indicates that the volume cannot hold any more data; displayed as `------u--`.
`hlock` (`hl`)	Indicates that the tape cannot be used for either input or output. This is a transient condition; the flag will be cleared by the LS after `REINSTATE_VOLUME_DELAY` has expired and at LS startup. Displayed as `----l----`.
`hoa` (`ho`)	Indicates that the volume is not to be used for either input or output, displayed as `--o------`. This value is only set or cleared by the site administrator.
`hro` (`hr`)	Indicates that the volume is read-only, displayed as `---r-----`; this inhibits the LS from using the volume for output. This value is only set or cleared by the site administrator.
`hsite1` (`h1`)	Reserved for site use; ignored by DMF. Not normally displayed; see the `dmvoladm`(8) man page for details. `hsite2`, `hsite3`, and `hsite4` are also available.
`hsparse` (`hs`)	Indicates that the volume is considered sparse and thus a candidate for a volume merge operation, displayed as `-------s-`.
`hvfy` (`hv`)	Indicates that this tape should be tested and/or replaced when next empty; until that time, it is read-only. Displayed as `----v----`. This value is set by DMF but only cleared by the site administrator.

For any field that takes a byte count, you may append the letter `k`, `m`, or `g` (in either uppercase or lowercase) to the integer to indicate that the value is to be multiplied by one thousand, one million, or one billion, respectively.

For information about the role of the `dmvoladm` command in LS database recovery, see "Database Recovery" on page 360. For details about `dmvoladm` syntax, see the man page.

Example 13-3 `dmvoladm list` Directives

The following example illustrates the default format for the `list` directive when using an LS. The column marked `HFLAGS` uses a format similar to the `ls -l` command in that each letter has an assigned position and its presence indicates that the flag is “on”. The positions spell out the string `eforvlus`, representing `herr`, `hfree`, `hoa`, `hro`, `hvfy`, `hlock`, `hfull`, and `hsparse`.

```
adm 1> list 000683-000703
                                          DATA    EOT   EOT            WR/FR
VSN      VOLGRP LB      DATA LEFT        WRITTEN  CHUNK  ZONE   HFLAGS      AGE
-------------------------------------------------------------------------------
000683   vg8a01 al       0.000000       0.000000      1     1  ---------     3d
000700   vg8a00 al     267.539255     287.610294    124     7  ------u--     2d
000701   vg8a00 al     288.342795     308.147798    136     8  ------u--     2d
000702   vg8a00 al     255.718902     288.302830    120     7  ------u--     2d
000703      ag8 al       0.000000       0.000000      1     1  ---------     3d
```

The following example illustrates using the `list` command to show only volumes meeting some criterion (in this case, those having their `hfull` flag set):

```
adm 1>list hfull
                                        DATA    EOT   EOT             WR/FR
VSN      VOLGRP LB      DATA LEFT       WRITTEN  CHUNK  ZONE   HFLAGS      AGE
-------------------------------------------------------------------------------
000701   vg8a00 al     288.342795     308.147798    136     8  ------u--     2d
000702   vg8a00 al     255.718902     288.302830    120     7  ------u--     2d
000704   vg8a00 al     252.294122     292.271410    119     7  ------u--     2d
000705   vg8a00 al     250.207666     304.603059    143     7  ------u--     2d
000706   vg8a00 al     265.213875     289.200534    144     7  ------u--     2d
000707   vg8a00 al     278.744448     310.408119    140     7  ------u--     2d
000708   vg8a00 al     260.827748     295.956588    136     7  ------u--     2d
000709   vg8a00 al     253.481897     283.615678    138     8  ------u--     2d
000710   vg8a00 al     265.100985     291.243235    141     7  ------u--     2d
000711   vg8a00 al     276.288446     305.782035    144     7  ------u--     2d
000712   vg8a00 al     250.415786     275.606243    138     7  ------u--     2d
000716   vg8a00 al     287.964765     304.321543    144     7  ------u--     2d
000717   vg8a00 al     280.795058     287.084534    144     7  ------u--     2d
000718   vg8a00 al       0.000415     300.852018    180    27  ------u--     3d
003127   vg9a01 al     417.383784     461.535047    209    10  ------u--     2d
003128   vg9a01 al     427.773679     460.716741    229    11  ------u--     2d
```

The following example shows one way you can customize the list format to show only the fields that you want to see. The other way is to use the `set format` command with the same keyword list.

```
adm 21>list S03232-S03254 format "eotchunk eotzone eotpos"
          EOT   EOT
VSN     CHUNK  ZONE    EOTPOS
-------------------------------
S03232     10     2 2/4294967295
S03233      2     2 2/4294967295
S03234    598     2 2/4294967295
S03235     18     2 2/4294967295
S03236     38     2 2/4294967295
S03237     92     2 2/4294967295
S03238      1     1 1/4294967295
S03239      1     1 1/4294967295
S03240      1     1 1/4294967295
```

```
S03241    325     2 2/4294967295
S03242     81     2 2/4294967295
S03243     26     2 2/4294967295
S03244      1     1 1/4294967295
S03245     26     2 2/4294967295
S03246      5     2 2/4294967295
S03247    186     2 2/4294967295
S03248     17     2 2/4294967295
S03249    526     2 2/4294967295
S03250      1     1 1/4294967295
S03251    533     2 2/4294967295
S03252    157    17 17/2147483648
S03253    636     2 2/4294967295
S03254     38     2 2/4294967295
```

The following example gives a convenient way to show the several flag bits in a way different from their usual representation.

```
adm 23>list 003232-003254 format "hfree hfull hlock hoa hro"
       hfree hfull hlock hoa hro
VSN
--------------------------------
003232  off    on    off off off
003233  off   off    off off off
003234  off   off    off off off
003235  off   off    off off off
003236  off    on    off off off
003237  off    on    off off off
003238  off    on    off off off
003239  off    on    off off off
003240  off   off    off off off
003241  off    on    off off off
003242  off    on    off off off
003243  off   off    off off off
003244  off   off    off off off
003245  off    on    off off off
003246  off   off    off off off
003247  off    on    off off off
003248  off    on    off off  on
003249   on   off    off off  on
003250   on   off    off off  on
003251   on   off    off off  on
```

```
003252   on   off    off off  on
003253  off    on    off off  on
003254  off    on    off off  on
```

The following example shows how to display only those tapes assigned to the VG named vg9a00.

```
adm 3>list vg=vg9a00
                                          DATA    EOT   EOT             WR/FR
VSN      VOLGRP LB      DATA LEFT        WRITTEN  CHUNK  ZONE   HFLAGS      AGE
-------------------------------------------------------------------------------
003210   vg9a00 al       1.048576       1.048576      3     2  ---------    11d
003282   vg9a00 al      11.534336      11.534336     13     2  ---------     7d
```

dmvoladm Text Field Order

The text field order for volume records generated by the dmdump(8), dmdumpj(8), and the dump directive in dmvoladm is listed below. This is the format expected by the load directives in dmvoladm:

1. V (indicates the volume record type)
2. vsn
3. volgrp
4. lbtype
5. capacity
6. blocksize
7. hflags (in octal)
8. version
9. datawritten
10. eotchunk
11. eotposition (eotzone/eotblockid) (in hexadecimal)
12. dataleft
13. chunksleft

14. `wfdate`
15. `update`
16. `id` (in octal). This field indicates the type of process that last updated the record.

`dmatread` Command

Use the `dmatread`(8) command to copy all or part of the data from a migrated file back to disk. You might want to do this if, for example, a user accidentally deleted a file and did not discover that the deletion had occurred until after the database entries had been removed by the hard delete procedure. Using backup copies of the databases from before the hard delete was performed, `dmatread` can restore the data to disk, assuming that the tape volume has not been reused in the meantime.

Example 13-4 Restoring Hard-deleted Files Using `dmatread`

To copy migrated files back to disk, perform the following steps:

1. Determine the BFID of the file you want to restore. You can use backup copies of `dmdlog` or your `dbrec.dat` files, or a restored dump copy of the deleted file's inode (and the `dmattr` command).
2. Using backup copies of the LS database, execute a `dmatread`(8) command similar to the following:

   ```
   dmatread -p /a/dmbackup -B 342984C50000000000084155
   ```

   `342984C50000000000084155` is the BFID of the file to be restored, and `/a/dmbackup` is the directory containing the backup copies of the LS database. Your file will be restored to the current directory as `B342984C50000000000084155`

DMF does not know the original name of the file; you must manually move the restored data to the appropriate file.

If you have access to chunk and VSN information for the file to be restored, you can use the `dmatread -c` and `-v` options and avoid using backup copies of the LS database. In this case, `dmatread` will issue messages indicating that the chunk is not found in the current LS database, but it will continue with the request and restore the file as described in this example.

`dmatsnf` Command

Use the `dmatsnf`(8) command to verify the readability of the LS volumes or to audit their contents. You may also generate text database records that can be applied to the LS database using the `load` directive in `dmcatadm` and `dmvoladm`, respectively; you can use the text records to add the contents of a few volumes to the LS database (this is impractical for large numbers of volumes).

`dmatsnf` can be used to verify one or more tape volumes against the LS database. It also can be used to generate journal entries, which can be added to the LS database by using the `load` directive in `dmvoladm` and `dmcatadm`.

`dmaudit verifymsp` Command

Use the `verifymsp` option of the `dmaudit`(8) command to check the consistency of the daemon database and LS database after an MSP, LS, DMF daemon, or system failure. This command captures the database files and compares the contents of the daemon database with each LS database. Any problems are reported to standard output, but no attempt is made to repair them.

You can also perform this function directly using `dmatvfy`(8) after taking a snapshot.

FTP MSP

The FTP MSP allows the DMF daemon to manage data by moving it to a remote machine. Data is moved to and from the remote machine with the protocol described in RFC 959 (FTP). The remote machine must understand this specific protocol.

Note: It is desirable that the remote machine run an operating system based on UNIX, so that the MSP can create subdirectories to organize the offline data. However, this is not a requirement.

The FTP MSP does not need a private database to operate; all information necessary to retrieve offline files is kept in the daemon database, DMF configuration file, and login information file. The login information file contains configuration information, such as passwords, that must be kept private. As a safeguard, the MSP will not operate if the login information file is readable by anyone other than the system administrator.

This section discusses the following:

FTP MSP Processing of Requests

The FTP MSP is always waiting for requests to arrive from the DMF daemon, but, to improve efficiency, it holds `PUT` and `DELETE` requests briefly and groups similar requests together into a single FTP session. No `PUT` request will be held longer than 60 seconds. No `DELETE` request will be held longer than 5 seconds. `GET` requests are not held. The MSP will stop holding requests if it has a large amount of work to do (more than 1024 individual files or 8 MB of data). The FTP MSP also limits the number of FTP sessions that can be active at once and the rate at which new sessions can be initiated.

After a request has been held for the appropriate amount of time, it enters a ready state. Processing usually begins immediately, but may be delayed if resources are not available.

The following limits affect the maximum number of requests that can be processed:

- An administrator-controlled limit on the maximum number of concurrent FTP sessions per MSP (`CHILD_MAXIMUM`).
- An administrator-controlled limit on the number of child processes that are guaranteed to be available for processing delete requests (`GUARANTEED_DELETES`).
- An administrator-controlled limit on the number of child processes that are guaranteed to be available for processing `dmget`(1) requests (`GUARANTEED_GETS`).
- A system-imposed limit of 85 FTP sessions in any 60-second period. This limit is seldom a concern because of the MSP's ability to transfer many files in one session. Because requests are grouped into batches only when resources are immediately available, `GET` requests (which are not normally held) are batched when resources are in short supply.

Requests are processed by forking off a child process. The parent process immediately resumes waiting for requests to arrive from the DMF daemon. The child process attempts to initiate an FTP session on the remote FTP server. If the remote machine

has multiple Internet Protocol (IP) addresses, all of them are tried before giving up. If the child process cannot connect, it waits 5 minutes and tries again until it succeeds.

Once a connection is established, the child process provides any required user name, password, account, and default directory information to the remote FTP server. `PUT`, `GET`, or `DELETE` operations are then performed as requested by the DMF daemon. `PUT`, `GET`, or `DELETE` operations are not intermixed within a batch. If an individual request does not complete successfully, it does not necessarily cause other requests in the same batch to fail. Binary transfer mode is used for all data transfer.

The stored files are not verbatim copies of the user files. They are stored using the same format used to write tapes, and you can use MSP utilities such as `dmatread` and `dmatsnf` to access the data in them.

FTP MSP Activity Log

All DMF MSPs maintain log files named `msplog.`*yyyymmdd* in the MSP spool directory which, by default, is *SPOOL_DIR/mspname*. *SPOOL_DIR* is configured in the `base` object of the configuration file; *mspname* is the name of the MSP in the `daemon` object of the configuration file; *yyyymmdd* is the current year, month, and day.

The activity log shows the arrival of new requests, the successful completion of requests, failed requests, creation and deletion of child processes, and all FTP transactions. Sensitive information (passwords and account information) does not appear in the activity log. In addition, the MSP lists the contents of its internal queues in its activity log if it is given an `INTERRUPT` signal.

Note: Because the FTP MSP will continue to create log files files without limit, you must remove obsolete files periodically by configuring the `run_remove_logs` task in the configuration file, as described in "`taskgroup` Object" on page 170.

FTP MSP Messages

The MSP also recognizes and handles the following messages issued from the DMF daemon:

Message	Description
`CANCEL`	Issued when a previously requested action is no longer necessary, for example, when a file being migrated with

	a PUT request is removed. The MSP is able to cancel a request if it is being held or if it is waiting for resources. A request that has begun processing cannot be canceled and will run to normal completion.
FINISH	Issued during normal shutdown. When the MSP receives a FINISH message, it finishes all requested operations as quickly as it can and then exits.
FLUSHALL	Issued in response to the dmdidle(8) command. When the MSP receives a FLUSHALL message, it finishes all requested operations as quickly as it can.

Caution: If the remote filesystem must be restored to a previous state, inconsistencies may arise: remote files that reappear after being deleted are never removed, and remote files that disappear unexpectedly result in data loss. There is presently no way to detect these inconsistencies. You should avoid situations that require the remote filesystem to be restored to a previous state.

Disk MSP

The disk MSP (dmdskmsp) migrates data into a directory that is accessed on the current system. It uses POSIX file interfaces to open, read, write, and close files. The directory may be NFS-mounted, unless the disk MSP is configured as a disk cache manager (see "Disk Cache Manager (DCM) MSP" on page 340). The data is read and written with root (UID 0) privileges. By default, dmdskmsp stores the data in DMF-blocked format, which allows the MSP to do the following:

- Keep metadata with a file
- Keep sparse files sparse when they are recalled
- Verify that a file is intact on recall

The disk MSP does not need a private database to operate; all information necessary to retrieve offline files is kept in the daemon database and DMF configuration file.

The disk MSP may also be used as an import MSP. In this case, it only permits recalls and copies the data unchanged for a recall.

This section discusses the following:

Disk MSP Processing of Requests

The disk MSP is always waiting for requests to arrive from the DMF daemon, but, to improve efficiency, it holds `PUT` and `DELETE` requests briefly and groups similar requests together into a single session. No `PUT` request will be held longer than 60 seconds. No `DELETE` request will be held longer than 5 seconds. `GET` requests are not held. The MSP will stop holding requests if it has a large amount of work to do (more than 1024 individual files or 8 MB of data).

After a request has been held for the appropriate amount of time, it enters a ready state. Processing usually begins immediately, but may be delayed if resources are not available.

The following limits affect the maximum number of requests that can be processed:

- An administrator-controlled limit on the maximum number of concurrent operations per MSP (`CHILD_MAXIMUM`).
- An administrator-controlled limit on the number of child processes that are guaranteed to be available for processing delete requests (`GUARANTEED_DELETES`).
- An administrator-controlled limit on the number of child processes that are guaranteed to be available for processing `dmget`(1) requests (`GUARANTEED_GETS`).

Requests are processed by forking off a child process. The parent process immediately resumes waiting for requests to arrive from the DMF daemon.

`PUT`, `GET`, or `DELETE` operations are performed as requested by the DMF daemon. `PUT`, `GET`, or `DELETE` operations are not intermixed within a batch. If an individual request does not complete successfully, it does not necessarily cause other requests in the same batch to fail. Binary transfer mode is used for all data transfer.

The stored files are not verbatim copies of the user files. They are stored using the same format used to write tapes, and you can use MSP utilities such as `dmatread` and `dmatsnf` to access the data in them.

Disk MSP Activity Log

All DMF MSPs maintain log files named `msplog.`*yyyymmdd* in the MSP spool directory which, by default, is *SPOOL_DIR*`/`*mspname*. *SPOOL_DIR* is configured in the `base` object of the configuration file; *mspname* is the name of the MSP in the `daemon` object of the configuration file; *yyyymmdd* is the current year, month, and day).

The log file shows the arrival of new requests, the successful completion of requests, failed requests, and creation and deletion of child processes. In addition, the MSP lists the contents of its internal queues in its activity log if it is given an `INTERRUPT` signal.

Note: Because the disk MSP will continue to create log files without limit, you must remove obsolete files periodically by configuring the `run_remove_logs` task in the configuration file, as described in "`taskgroup` Object" on page 170.

Disk Cache Manager (DCM) MSP

The *Disk cache manager (DCM) MSP* is the disk MSP configured for *n*–tier capability using a dedicated filesystem as a cache. The DCM provides fast access for files whose activity levels remain high while also providing migration to tape for those files requiring less frequent access.

To allow the disk store that is managed by the disk MSP to function as a dynamically managed cache (as opposed to a static store), the DCM creates and maintains a filesystem attribute on each file that is created in the MSP `STORE_DIRECTORY`. This attribute is used by the `dmdskfree` process to evaluate files for downward migration and for possible removal from the disk cache. For this reason, the DCM `STORE_DIRECTORY` must be a local XFS or CXFS filesystem mount point with DMAPI enabled.

The DCM supports *dual-resident state*, in which files reside in the cache and also in a lower VG. This provides the access speed of a disk file, but allows that cache file to be quickly released without the need to first write it to tape. This is directly analogous to the concept of a dual-state file in the standard DMF-managed filesystem.

Automated movement in the opposite direction (from tape back to the cache) is not available. Any recalls of files that no longer have copies held in the cache will come directly from tape; they are not recalled via the cache and they can only be restored to the cache by an explicit `dmmove`(8) command.

dmdskvfy Command

The dmdskvfy command verifies that copies of migrated files in DCM and disk MSPs are consistent with the daemon database entries that refer to them.

Moving Migrated Data between MSPs and VGs

DMF provides a mechanism to move copies of offline or dual-state files from one MSP or VG to another. The dmmove(8) command takes a list of such files and moves them to a specified set of MSPs or VGs. The list of MSPs or VGs specified to the dmmove command indicates which MSPs or VGs are to contain migrated copies of a file after the move process is completed. All other migrated copies are hard-deleted unless the dmmove -d option is used to select which copies are to be hard-deleted.

If a file's migrated state is offline, dmmove recalls the file to disk and then remigrates it to the specified MSPs or VGs. (The one exception to this is that if a disk cache manager disk MSP copy exists, the file will be moved directly from that file copy.) When the migration process is complete, the online copy is removed. The file is recalled to a scratch filesystem that is specified by the MOVE_FS configuration parameter. If the file is dual-state, dmmove does not need to recall the file first, but instead uses the existing online copy.

The dmselect(8) command can be used to determine which files you want to move. dmselect selects files based on age, size, ownership, and MSP criteria. The output from the dmselect command can be used with the dmmove command. The dmmove command also accepts a list of pathnames as input.

See the man pages for dmselect and dmmove for all the possible options and further information.

LS Error Analysis and Avoidance

The drive group component of the LS monitors tape use, analyzes failures, and uses this information to avoid future errors.

The drive group component can react to some failures without looking for any patterns of behavior. Among these are the following:

- Mounting service failure. If the mounting service is TMF, by default, DMF makes one attempt to restart it. If this attempt does not succeed, DMF notifies the

administrator by e-mail and waits for the administrator's intervention. When TMF is back again, DMF resets the auto-restart flag so that if TMF fails again, it will once again make one attempt to restart it.

If OpenVault is the mounting service, by default, no attempt is made to restart it. Instead, an e-mail is sent to the administrator.

A site can set the number of automatic restart attempts by using the drive group's `MAX_MS_RESTARTS` configuration parameter, but caution and thorough testing are advised. There are many possible failure modes for a mounting service, and automated restarts might not always be appropriate.

- Tape volume is not in the tape library. Obviously, this problem will not be fixed by trying again. To prevent further access, the volume is locked by setting the `HLOCK` flag, as described below, and the user requests that triggered the access attempt are retried on another tape, if possible; otherwise, they are aborted. The administrator is notified by e-mail.

- For TMF only, a tape mount was cancelled by an operator or administrator. Although the user requests are retried or aborted, the volume is not disabled. If the volume were disabled, it would be inaccessible for a period of time (default 24 hours) unless `dmvoladm` were used to preempt this delay. All operators do not necessarily have access to the `dmvoladm` command.

  Because the reason for the cancellation is unknown to DMF, repeated requests for the same volume are quite possible, and the operator might have to cancel each one.

The drive group handles other types of failure by examining the recent history of the tape volume and the tape drive that was used. The drive group maintains records of past tape I/O errors, and uses these to control the way it reacts to future errors.

For example, if a tape has been unusable several times in a row, even though different tape drives were used, the drive group concludes that the problem most likely involves the tape volume rather than the drive. Therefore, it suspends use of that tape, forcing DMF to migrate to a different tape in that VG or to recall the file from another tape held by a different VG. This suspension is usually done by setting the `HLOCK` flag in the tape's entry in the VOL record of the LS database. This makes the tape inaccessible to the VG for both reading and writing until it is automatically cleared after `REINSTATE_VOLUME_DELAY` minutes.

If a variety of volumes fail on a specific drive but are usable on other drives, a drive problem is likely, and the tape drive can be automatically configured down if permitted by the administrator's setting of `DRIVES_TO_DOWN` to a value higher than

its default of zero. When a drive is configured down in this way, it is configured up again after REINSTATE_DRIVE_DELAY minutes.

The analyses of drive and volume errors are performed independently of each other; it is possible for one additional error to result in both the drive and the volume being disabled.

There are several reasons for reinstating drives and volumes after a delay. The most important is that the analyses of previous failures might lead to a faulty conclusion in some situations, such as when DMF is under a very light load, or when multiple failures occur concurrently. A wrong diagnosis might impact DMF's performance, and should not be accepted indefinitely. Disabling a suspected drive or volume for a while is usually enough to break any repetitive cycles of failure. If such patterns re-establish themselves when the reinstatement occurs, the drive group will again analyze the behavior, possibly reaching a different conclusion, and again try to prevent it.

There are some variations from these general reactions. For example, if a tape volume with existing data on it is diagnosed as faulty when appending new data, instead of setting the HLOCK flag, the drive group sets HVFY, which results in the tape being used in a read-only mode until eventually emptied by merges or hard deletion of its files. At that time, the administrator may choose to test it and possibly replace or delete it. If it is to be returned to service, the HVFY flag should be cleared by using dmvoladm. Full details of these procedures are included in the email sent to the administrator at the time of the error.

If it is considered desirable to return a volume or drive to service earlier than defined in the DMF configuration, the appropriate command (dmvoladm, tmconfig, or ov_drive) can be safely used.

LS Drive Scheduling

When multiple VGs are requesting the use of more tape drives than exist in the drive group, the resource scheduler is used to decide which VGs should wait and which should be assigned the use of the drives.

The resource scheduler is unaware of non-volume-group activity on the drives in its drive group. Such activity includes XFS dumps any direct tape use by the system's users; it does not prevent the LS from working properly, though it might be less than optimal.

LS Status Monitoring

You can observe the performance of the LS in two ways:

- Monitor its log file with a tool like `tail -f`, which allows an experienced administrator to follow the flow of events as they happen
- Use the resource watcher component, when enabled by use of the `WATCHER` parameter in the `libraryserver` configuration stanza

The resource watcher is intended to give the administrator a view of the status of an LS and some of its components. It maintains a set of text files on disk that are rewritten as events happen. These files can be found in the `SPOOL_DIR/lsname/_rwname` directory, where `SPOOL_DIR` is defined in the DMF configuration file, as are the names of the LS and resource watcher; for example, `lsname` and `rwname`. The easiest way to find the precise path is to look in the LS log file for messages like the following:

```
rwname.config_changed:  URL of home page is file:/dmf/spool/lsname/_rwname/lsname.html
```

This message is issued at DMF startup or whenever the configuration file is altered or its modification time changes; for example, by using the `touch`(1) command.

The `SPOOL_DIR/lsname/_rwname` directory contains files with names ending in `.html`, which are automatically refreshing HTML files. You can access these files by using a browser running on the same machine. The following example shows an LS page that contains links to drive group pages, and they in turn have links to VG pages, if the VGs are active at the time:

```
netscape file:/dmf/spool/lsname/_rwname/lsname.html
```

If running the browser on the DMF machine is inconvenient, you can include the directory in your HTTP server configuration to allow those same pages to be accessed via the web.

This directory also contains files whose names end in `.txt`, designed to be parsed with programs like `awk`. The data format is described by comments within those files and can be compared with the equivalent HTML files.

If the format of the text ever changes, the version number will change. If the changes are incompatible with previous usage, the number before the decimal point is altered. If they are compatible, the number after the decimal point is altered.

An example of compatibility is adding extra fields to the end of existing lines or adding new lines. Programs using these files should check the version number to ensure compatibility. Also, it might be useful to check the following:

- DMF version shown by `dmversion`(1)
- Linux kernel version shown by `uname`(1)
- Linux distribution version shown by `head /etc/*release`

DMF Maintenance and Recovery

This chapter contains the following:

Retaining Old DMF Daemon Log Files

The daemon generates the *SPOOL_DIR*`/`*daemon_name*`/dmdlog.`*yyyymmdd* log file, which contains a record of DMF activity and can be useful for problem solving for several months after creation. All MSPs and LSs generate a *SPOOL_DIR*`/`*msp_or_ls_name*`/msplog.`*yyyymmdd* log file, which also contains useful information about its activity. The LS also generates *SPOOL_DIR*`/`*ls_name*`/moverlogs/`*hostname*`/moverlog.`*yyyymmdd* log files, which also contain useful information about its activity. These log files should be retained for a period of some months. Log files more than a year old are probably not very useful.

Do not use DMF to manage the *SPOOL_DIR* filesystem.

The `dmfsmon`(8) automated space management daemon generates a log file in *SPOOL_DIR*`/`*daemon_name*`/autolog.`*yyyymmdd*, which is useful for analyzing problems related to space management.

To manage the log files, configure the `run_remove_logs.sh` task, which automatically deletes old log files according to a policy you set. See "`taskgroup` Object" on page 170, for more information.

Retaining Old DMF Daemon Journal Files

The daemon and the LS generate journal files that are needed to recover databases in the event of filesystem damage or loss. You also configure DMF to generate backup copies of those databases on a periodic basis. You need only retain those journal files that contain records created since the oldest database backup that you keep. In theory, you should need only one database backup copy, but most sites probably feel safer with more than one generation of database backups.

For example, if you configure DMF to generate daily database backups and retain the three most recent backup copies, then at the end of 18 July there would be backups from the 18th, 17th, and 16th. Only the journal files for those dates need be kept for recovery purposes.

To manage the journal files and the backups, configure the `run_remove_journals.sh` and `run_copy_databases.sh` tasks. These tasks automatically delete old journal files and generate backups of the databases according to a policy you set. See "`taskgroup` Object" on page 170, for more information.

Soft- and Hard-Deletes

When a file is first migrated, a bit-file identifier (BFID) is placed in the inode; this is the key into the daemon database. When a migrated file is removed, its BFID is no longer needed in the daemon database.

Initially, it would seem that you could delete daemon database entries when their files are modified or removed. However, if you actually delete the daemon database entries and then the associated filesystem is damaged, the files will be irretrievable after you restore the filesystem.

For example, assume that migrated files were located in the `/x` filesystem, and you configured DMF to generate a full backup of `/x` on Sunday as part of your site's weekly administrative procedures (the `run_full_dump.sh` task). Next, suppose that you removed the migrated files in `/x` on Monday morning and removed the corresponding daemon database entries. If a disk hardware failure occurs on Monday afternoon, you must restore the `/x` filesystem to as recent a state as possible. If you restore the filesystem to its state as of Sunday, the migrated files are also returned to their state as of Sunday. As migrated files, they contain the old BFID from Sunday in their inodes, and, because you removed their BFIDs from the daemon database, you cannot recall these files.

Because of the nature of the filesystem, a daemon database entry is not removed when a migrated file is modified or removed. Instead, a deleted date and time field is set in the database. This field indicates when you were finished with the database entry, except for recovery purposes; it does not prohibit the daemon from using the database entry to recall a file. When the `/x` filesystem is restored in the preceding example, the migrated files have BFIDs in their inodes that point to valid database entries. If the files are later modified or removed again, the delete field is updated with this later date and time.

The term *soft-deleted* refers to a database entry that has the delete date and time set. The term *hard-deleted* refers to a file that is removed completely from the daemon database and the MSPs/LSs. You should hard-delete the older soft-deleted entries periodically; otherwise, the daemon database continues to grow in size without limit as old, unnecessary entries accumulate. Configure the `run_hard_deletes.sh` task to perform hard-deletes automatically. See "`taskgroup` Object" on page 170, for more information.

If you look at all of the tapes before and after a hard-delete operation, you will see that the amount of space used on some (or all) of the tapes has been reduced.

Note: Because hard-deletions normally use the same expiry times as backups, the `run_hard_deletes.sh` is normally run from the same task group.

Backups and DMF

This section discusses the interrelationships between DMF and backup products:

Caution: The fact that DMF maintains copies of data on another medium does not mean that it is a backup system. The copies made by DMF may become inaccessible if there is a failure and proper backups have not been made.

In addition, although using RAID may protect you against the failure of one disk spindle, data can still be endangered by software problems, human error, or hardware failure.

Therefore, **backups are essential**.

DMF-managed User Filesystems

Many backup and recovery software packages make backup copies of files by opening and reading them using the standard UNIX or Linux system calls. In a user filesystem managed by DMF, this causes files that are offline to be recalled back to disk before they can be backed up. If you have a DMF-managed filesystem in which a high percentage of the files are offline, you may see a large amount of tape or other activity caused by the backup package when it initially does its backups. You should take this behavior into account when deciding whether or not to use such backup packages with filesystems managed by DMF.

This section discusses the following:

Using SGI `xfsdump` and `xfsrestore` with Migrated Files

Note: `xfsrestore` may attempt to read, write, or delete files that are under DMF management. If this occurs while DMF is not running, the `xfsrestore` process may block indefinitely waiting for a DMF event to be completed. If you use `xfsrestore` to create or modify files in a filesystem that already contains files managed by DMF, you are more likely to encounter this issue than if you use `xfsrestore` to populate an empty filesystem. To avoid this problem, use `xfsrestore` while DMF is running.

The xfsdump(8) and xfsrestore(8) commands back up filesystems. These utilities are designed to perform the backup function quickly and with minimal system overhead. They operate with DMF in two ways:

- When xfsdump encounters an offline file, it does not cause the associated data to be recalled. This distinguishes the utility from tar(1) and cpio(1), both of which cause the file to be recalled when they reference an offline file.
- The dmmigrate(8) command lets you implement a 100% migration policy that does not interfere with customary management of space thresholds.

  The xfsdump command supports the -a option specifically for DMF. If you specify the -a option, xfsdump will dump DMF dual-state (DUL) files as if they were offline (OFL) files. That is, when xfsdump detects a file that is backed up by DMF, it retains only the inode for that file because DMF already has a copy of the data itself. This dramatically reduces the amount of tape space needed to back up a filesystem and it also reduces the time taken to complete the dump, thereby minimizing the chances of it being inaccurate due to activity elsewhere in the system. An added advantage of using -a is that files that are actively being recalled will still be backed up correctly by xfsdump because it does not need to copy the file's data bytes to tape.

  You can also use dmmigrate to force data copies held only in a disk cache manager (DCM) cache to be copied to tapes in the underlying volume groups (VGs). This removes the need to back up the cache filesystem. However, if you do wish to back up the cache instead of flushing it to tape, you can use any backup utility. As the cache is not a DMF-managed filesystem, you are not restricted to using xfsdump.

Most installations periodically do a full (level 0) dump of filesystems. Incremental dumps (levels 1 through 9) are done between full dumps; these may happen once per day or several times per day. You can continue this practice after DMF is enabled. When a file is migrated (or recalled), the inode change time is updated. The inode change time ensures that the file gets dumped at the time of the next incremental dump.

To automatically manage dump tapes, DMF includes configurable administrative scripts called run_full_dump.sh and run_partial_dump.sh, which employ xfsdump. Both of these tasks are simple wrappers around a script called do_xfsdump.sh, which performs the following actions:

- *(optional)* Migrates all eligible files to dual-state
- *(optional)* Copies all eligible DCM files on a DCM system to dual-residency state

- Performs a database snapshot using dmsnap
- Backs up the directory containing that snapshot
- Backs up other filesystems
- After a successful full backup, frees up old backup tapes for future reuse

DMF also supports a matching wrapper around xfsrestore named dmxfsrestore to be used when restoring files that were dumped by these backup scripts. See the dmxfsrestore(8) man page for more information on running the command.

You can configure tasks in the dump_tasks object to automatically do full and incremental dumps of the DMF-managed filesystems. See "taskgroup Object" on page 170, for more information.

A typical dump_tasks stanza might look like the following:

```
define  dump_tasks
        TYPE                    taskgroup
        RUN_TASK                $ADMINDIR/run_full_dump.sh on sunday at 03:00
        RUN_TASK                $ADMINDIR/run_partial_dump.sh \
                                   on monday tuesday wednesday \
                                   thursday friday saturday at 03:00
        RUN_TASK                $ADMINDIR/run_hard_deletes.sh at 23:00
        DUMP_TAPES              HOME_DIR/tapes
        DUMP_RETENTION          4w
        DUMP_DEVICE             dg1
        DUMP_MIGRATE_FIRST      yes
        DUMP_FLUSH_DCM_FIRST    yes        # Only if you run a DCM
        DUMP_INVENTORY_COPY     /save/dump_inventory
enddef
```

Note: When an MSP, LS, daemon, or configuration file object (such as dump_tasks) obtains a path such as HOME_DIR from the configuration file, the actual path used is the value of HOME_DIR plus the MSP/LS/daemon/object name appended as a subdirectory. In the above example, if the value of HOME_DIR was set to /dmf/home in the configuration file, then the actual path for DUMP_TAPES would be resolved to /dmf/home/dump_tasks/tapes.

For more information about parameters, see "Starting and Stopping the DMF Environment" on page 91.

Sites using OpenVault can add new backup tapes by using `dmov_makecarts` and/or `dmov_loadtapes` by providing the name of the task group as a parameter. Sites using TMF do not need any special steps to add new tapes, as TMF does not record details of which tapes are available to it.

Recycling old backup tapes is performed automatically after the successful completion of a full dump. In certain situations, such as running out of dump tapes, this pruning must be done manually by running `dmxfsprune`.

Ensuring Accuracy with `xfsdump`

The `xfsdump` program is written such that it assumes dumps will only be taken within filesystems that are not actively changing. `xfsdump` cannot detect that a file has changed while it is being dumped, so if a user should modify a file while it is being read by `xfsdump`, it is possible for the backup copy of the file to be inaccurate.

To ensure that all file backup copies are accurate, perform the following steps when using `xfsdump` to dump files within a DMF filesystem:

1. Make sure that there is no user activity within the filesystem.
2. Ensure that DMF is not actively migrating files within the filesystem.
3. Run `xfsdump`, preferably with the `-a` option.

Dumping and Restoring Files without the DMF Scripts

If you choose to dump and restore DMF filesystems without using the provided DMF scripts, there are several items that you must remember:

- The DMF scripts use `xfsdump` with the `-a` option to dump only data not backed up by DMF. You may also wish to consider using the `-a` option on `xfsdump` when dumping DMF filesystems manually.
- **Do not use the `-A` option** on either `xfsdump` or `xfsrestore`. The `-A` option avoids dumping or restoring extended attribute information. DMF information is stored within files as extended attributes, so if you do use `-A`, migrated files restored from those dump tapes will not be recallable by DMF.
- When restoring migrated files using `xfsrestore`, you must specify the `-D` option in order to guarantee that all DMF-related information is correctly restored.
- If you use the Tape Management Facility (TMF) to mount tapes for use by `xfsdump`, be aware that `xfsdump` will not detect the fact that the device is a tape,

and will behave as if the dump is instead being written to a regular disk file. This means that `xfsdump` will not be able to append new dumps to the end of an existing tape. It also means that if `xfsdump` encounters end-of-tape, it will abort the backup rather than prompting for additional volumes. You must ensure that you specify enough volumes using the `tmmnt -v` option before beginning the dump in order to guarantee that `xfsdump` will not encounter end-of-tape.

Filesystem Consistency with `xfsrestore`

When you restore files, you might be restoring some inodes containing BFIDs that were soft-deleted since the time the dump was taken. (For information about soft-deletes, see "Soft- and Hard-Deletes" on page 348.) `dmaudit`(8) will report this as an inconsistency between the filesystem and the daemon database, indicating that the database entry should not be soft-deleted.

Another form of inconsistency occurs if you happen to duplicate offline or dual-state files by restoring all or part of an existing directory into another directory. In this case, `dmaudit` will report as an inconsistency that two files share the same BFID. If one of the files is subsequently deleted causing the database entry to be soft-deleted, the `dmaudit`-reported inconsistency will change to the type described in the previous paragraph.

While these `dmaudit`-reported inconsistencies may seem serious, there is no risk of losing user data. The `dmhdelete`(8) program responsible for removing unused database entries always first scans all DMF-managed filesystems to make sure that there are no remaining files which reference the database entries it is about to remove. It is able to detect either of these inconsistencies and will not remove the database entries if inconsistencies are found.

Be aware that inconsistencies between a filesystem and the daemon database can occur as a result of restoring migrated files. It is good practice to run `dmaudit` after every restore to correct those inconsistencies.

Using DMF-aware Third-Party Backup Packages

Some third-party backup packages can use a DMF library to perform backups in a DMF-aware manner. When the DMF-aware feature is enabled, these packages will not cause offline (`OFL`) files to be recalled during a backup. Dual-state (`DUL`) files will be dumped as if they were offline, which will reduce the time and space needed for a backup.

To use a DMF-aware third-party backup package to back up DMF filesystems, do the following:

1. Configure the backup package to include the DMF filesystems in the backups.
2. Enable the DMF-aware feature on those filesystems.

For more information about third-party backup packages, see Appendix D, "Third-Party Backup Package Configuration" on page 463.

DMF provides a script called `do_predump.sh` that is meant to be run just prior to a backup of the DMF filesystems using a third-party backup package. The `do_predump.sh` script does the following:

- *(Optional)* Migrates all eligible files to dual-state
- *(Optional on a DCM system)* Copies all eligible DCM files to dual-residency state
- *(Optional)* Performs a snapshot of the databases by using `dmsnap`

To use `do_predump.sh`, do the following:

1. Configure the backup package to run `do_predump.sh` as the pre-backup command. For details, see the application-specific information in Appendix D, "Third-Party Backup Package Configuration" on page 463.
2. Define a task group in the `dmf.conf` file that is referred to by the `dmdaemon` object. In the supplied configurations, this task group is called `dump_tasks`.

The parameters `do_predump.sh` uses are as follows:

`DUMP_DATABASE_COPY`	Specifies a path to where a snapshot of the DMF databases will be placed. The backup package should be configured to backup this directory. If not specified, no snapshot will be taken.
`DUMP_FLUSH_DCM_FIRST`	If set to `YES`, specifies that `dmmigrate` is run to ensure that all non-dual-resident files in the DCM caches are migrated to tape. If `DUMP_MIGRATE_FIRST` is also enabled, that is processed first.
`DUMP_FILE_SYSTEMS`	If `DUMP_MIGRATE_FIRST` is enabled, specifies the DMF-managed filesystems on which to run the `dmmigrate` command. The default value for this parameter is all DMF-managed filesystems.

DUMP_MIGRATE_FIRST	If set to YES, specifies that dmmigrate is run to ensure that all migratable files in the DMF-managed filesystems are migrated, thus reducing the number of tapes needed for the dump and making it run much faster.

Because hard-deletions normally use the same expiry time as backups, run_hard_deletes.sh is normally run from the same task group. The DUMP_RETENTION parameter should match the retention policy of the backup package.

When using a third-party backup package, a typical dump_tasks stanza might look like the following:

```
define dump_tasks
        TYPE                    taskgroup
        RUN_TASK                $ADMINDIR/run_hard_deletes.sh at 23:00
        DUMP_RETENTION          4w      # match backup package's policy

        DUMP_MIGRATE_FIRST      yes
        DUMP_FLUSH_DCM_FIRST    yes     # only if running a DCM
        DUMP_DATABASE_COPY      /path/to/db_snapshot
enddef
```

Note: Backups and restores must be run from the DMF server.

Only root can perform backups and restores. Although some third-party backup packages normally allow unprivileged users to restore their own files, unprivileged users cannot restore their own files from a DMF filesystem because doing so requires root privilege to set the DMF attribute.

Files backed up from a DMF filesystem should only be restored to a DMF filesystem. Otherwise, files that are offline (or treated as such) will not be recallable.

Using XVM Snapshots and DMF

You can use the xfsdump facility to backup XVM snapshots of a DMF-managed user filesystem. Note the following:

- XVM snapshots of DMF-managed filesystems should not be added to the DMF configuration file.
- You should not attempt to migrate or recall files from an XVM snapshot of a DMF filesystem.

- You can only restore DMF offline, partial, and unmigrating files by using `xfsdump` and `xfsrestore`. You cannot use a previously taken snapshot of a DMF filesystem to directly copy any of these file types back into the live filesystem. You may copy migrating or dual-state files back into the live filesystem (to DMF, they will appear to be new files).

For more information about XVM snapshots, see the *XVM Volume Manager Administrator's Guide*.

Optimizing Backups of Filesystems

You can greatly reduce the amount of time it takes to back up filesystems by configuring DMF to migrate all files. Do the following:

- Set the `DUMP_MIGRATE_FIRST` parameter to `yes`, which specifies that the `dmmigrate` command is run before the dumps are done to ensure that all migratable files in the DMF-managed user filesystems are migrated.
- Execute one of the following scripts:
  - `run_full_dump` to perform a full backup of the filesystems
  - `run_partial_dump` to perform a partial backup of the filesystems

For more information, see "Starting and Stopping the DMF Environment" on page 91.

Migrating all files before performing a backup has the following benefits:

- The backup image will be smaller because it contains just the metadata information, not the file data itself
- The backup will complete more quickly because:
  - It is reading just the metadata
  - There is less time spent performing random disk seeks to back up the data of unmigrated files

For any files that you want to remain permanently on disk (that is, permanently dual-state), you can assign a negative priority weight to those files, which would leave the files on disk. The result is that when the filesystem is filled up, DMF will never free the blocks for these files. The files therefore are always dual-state, ready to be used. When the filesystem is backed up, the backup facility will recognize that they are dual-state and therefore back them up as offline. The net effect is that there is no file data in the backup at all for these files, just their inodes, while keeping the

files always available. In the case of millions of small files, this speed-up of the backup process can be dramatic. For example, for a filesystem with a large number of small files (files of up to 64 KB), you could assign the following AGE_WEIGHT value:

```
AGE_WEIGHT      -1      0       when space < 64k
```

Be aware of the following:

- For extremely small files (under a few hundred bytes), the disk space required for DMF database entries may exceed the size of the original file. For extremely large numbers of such files, this issue should be considered.
- The space value in a when clause, as used above, refers to the space the file occupies on disk, which for sparse files may actually be smaller than the size of the file as shown by ls -l. The space value will be rounded upward to a multiple of the disk blocksize defined by mkfs(8); the default is 4096 bytes. For example, attempting to discriminate between files above or below 1000 bytes based on their space value is futile because all non-empty files will have a space value that is a multiple of (typically) 4096 bytes.

If you use negative weights with AGE_WEIGHT or SPACE_WEIGHT, DMF automatic migration will never free the space for these files but a user can still do a dmput -r on them to manually free the space.

However, if you do not want files to migrate for any reason, then you must continue to use the SELECT_VG method despite the slower and larger backups.

Storage Used by an FTP MSP or a Standard Disk MSP

If you are depending on an FTP MSP or a standard disk MSP to provide copies of your offline files in order to safeguard your data, then they should also be backed up.

If you use them just to hold extra copies for convenience or to speed data access, they need not be backed up. But you should consider how you would handle their loss. You would probably need to remove references to lost copies from the DMF daemon database, using dmdadm, which can only be done when the daemon is not running.

Filesystems Used by a Disk Cache Manager (DCM)

A DCM MSP differs from a disk MSP in that it uses DMAPI to manage the files. It will not operate properly if the files are reloaded by a package that cannot also restore the DMAPI information associated with each file.

Note: For simplicity, this discussion assumes that the site wishes to keep two copies of migrated files at all times to guard against media problems. (Keeping only one copy is considered risky, and keeping more than two copies is frequently impractical.)

The DCM can have one of the following configurations:

- A DCM may be holding an extra copy of files in addition to the normal number of tape-based copies. That is, after the initial migration has completed, there will be two tape copies and a third in the cache. The DCM may easily remove this third copy from the cache after some period of time, just leaving two tape-based copies. With this configuration, there is normally no need to back up the cache filesystem.
- The initial migration could result in one cache copy and one on tape. Later on, when the cache has to be flushed, a second tape copy is written by the DCM before the cache-resident one is deleted. If the file is hard-deleted before the cache flushes, the second tape copy will never be made, thereby saving time and tape. The tradeoff is that cache-flushing is slower and the cache filesystem should be backed up; otherwise a tape media problem in conjunction with a disk failure would result in data loss. With this configuration, the cache filesystem should be backed up. Otherwise, the loss of the cache disk could leave you with just one copy of data on tape. This is considered to be risky.

For both configurations, any backups require the use of a DMF-aware backup package (as listed in Appendix D, "Third-Party Backup Package Configuration" on page 463) to back up the cache.

To use `do_xfsdump.sh` to backup any of these filesystems, include the pathname of its mountpoint in the `DUMP_FILE_SYSTEMS` parameter.

DMF's Private Filesystems

The following DMF private filesystems do not require a DMF-aware backup package:

HOME_DIR
JOURNAL_DIR
SPOOL_DIR
TMP_DIR
CACHE_DIR
MOVE_FS

Take care when backing up the databases in *HOME_DIR* if there is any DMF activity going on while the backup is underway, due to the risk of making the copy of the database while it is being updated. A safe technique is to take a snapshot of the databases with `dmsnap` and back up the snapshot. The `do_xfsdump.sh` script does this automatically.

The journal files in *JOURNAL_DIR* should also be backed up if you keep older snapshots of the databases that may have to be reloaded and brought up-to-date with `dmdbrecover`. Preferably, journals should be backed up when DMF activity (apart from recalls) is minimal. The `do_xfsdump` parameters `DUMP_MIGRATE_FIRST` and `DUMP_FLUSH_DCM_FIRST` help achieve this by processing any queued up migration requests immediately before starting the backup.

SPOOL_DIR contains log files that may be of use for problem diagnosis, as well as history files controlling things like tape error recovery and reporting scripts. The loss of these files will not endanger user data, although DMF may act a little differently for a while until it reestablishes them. Back up *SPOOL_DIR* if you can.

The *TMP_DIR*, *CACHE_DIR*, and *MOVE_FS* filesystems do not require backup.

To use `do_xfsdump.sh` to backup any of these filesystems, simply include the pathnames of their mountpoints in the `DUMP_FILE_SYSTEMS` parameter.

Using `dmfill`

The `dmfill`(8) command allows you to fill a restored filesystem to a specified capacity by recalling offline files. When you execute `xfsdump -a`, only inodes are dumped for all files that have been migrated (including dual-state files). Therefore, when the filesystem is restored, only the inodes are restored, not the data. You can use `dmfill` in conjunction with `xfsrestore` to restore a corrupted filesystem to a previously valid state. `dmfill` recalls migrated files in the reverse order of migration until the requested fill percentage is reached or until there are no more migrated files left to recall on this filesystem.

Database Recovery

The basic strategy for recovering a lost or damaged DMF database is to recreate it by applying journal records to a backup copy of the database. For this reason it is essential that database backup copies and journal files reside on a different physical device from the production databases; it is also highly desirable that these devices

have different controllers and channels. The following sections discuss the database recovery strategy in more detail:

- "Database Backups" on page 361
- "Database Recovery Procedures" on page 361

Database Backups

You can configure commands in the `run_copy_databases.sh` task (in the `dump_tasks` object) to automatically generate DMF database backups. See "`taskgroup` Object" on page 170, for more information.

You must back up the following files:

- The daemon database files and definition file in the *HOME_DIR/daemon_name* directory:

  ```
  dbrec.dat
  dbrec.keys
  pathseg.dat
  pathseg.keys
  dmd_db.dbd
  ```

Each LS database has the following files in the *HOME_DIR/ls_name* directory:

- CAT records:

  ```
  tpcrdm.dat
  tpcrdm.key1.keys
  tpcrdm.key2.keys
  ```

- VOL records:

  ```
  tpvrdm.dat
  tpvrdm.vsn.keys
  ```

- Database definition file: `libsrv_db.dbd`

Database Recovery Procedures

The DMF daemon and LS write journal file records for every database transaction. These files contain binary records that cannot be edited by normal methods and that

must be applied to an existing database with the dmdbrecover(8) command. The following procedure explains how to recover the daemon database.

Warning: If you are running on multiple LSs, always ensure that you have the correct journals restored in the correct directories. Recovering a database with incorrect journals can cause irrecoverable problems.

Procedure 14-1 Recovering the Databases

If you lose a database through disk spindle failure or through some form of external corruption, use the following procedure to recover it:

1. Stop DMF (in an non-HA environment): [1]

   

   Caution: For instructions about starting and stopping DMF and the mounting service in an HA environment, see the SGI HA documentation.

   ```
   # service dmf stop
   ```

2. If you have configured the run_copy_databases task, restore the files from the directory with the most recent copy of the databases that were in *HOME_DIR* to *HOME_DIR/daemon* or *HOME_DIR/LS_NAME*.

3. If you have **not** configured the run_copy_databases task, reload an old version of the daemon or LS database. Typically, these will be from the most recent dump tapes of your filesystem.

4. Ensure that the default *JOURNAL_DIR*/*daemon_name* (or *JOURNAL_DIR/ls_name*) directory contains all of the time-ordered journal files since the last update of the older database.

   For the daemon, the files are named dmd_db.*yyyymmdd[.hhmmss]*.

   For the LS, the journal files are named libsrv_db.*yyyymmdd[.hhmmss]*.

5. Use dmdbrecover to update the old database with the journal entries from journal files identified in step 4.

[1] For instructions about starting and stopping DMF and the mounting service in an HA environment, see the SGI HA documentation.

Example 14-1 Database Recovery Example

Suppose that the filesystem containing *HOME_DIR* was destroyed on February 1, 2004, and that your most recent backup copy of the daemon database and LS database is from January 28, 2004. To recover the databases, you would do the following:

1. Stop DMF (in an non-HA environment): [2]

   # **service dmf stop**

2. Ensure that *JOURNAL_DIR/daemon_name* (or *JOURNAL_DIR/ls_name*) contains the following journal files (one or more for each day):

   JOURNAL_DIR/daemon_name

   ```
   dmd_db.20040128.235959
   dmd_db.20040129.235959
   dmd_db.20040130.235959
   dmd_db.20040131.235959
   dmd_db.20040201
   ```

   JOURNAL_DIR/ls_name

   ```
   libsrv_db.20040128.235959
   libsrv_db.20040129.235959
   libsrv_db.20040130.235959
   libsrv_db.20040131.235959
   libsrv_db.20040201
   ```

3. Restore databases from January 28, to *HOME_DIR*/*daemon_name* and/or *HOME_DIR/ls_name*. The following files should be present:

   HOME_DIR/*daemon_name*

   ```
   dbrec.dat
   dbrec.keys
   pathseg.dat
   pathseg.keys
   ```

[2] For instructions about starting and stopping DMF and the mounting service in an HA environment, see the SGI HA documentation.

HOME_DIR/ls_name

```
tpcrdm.dat
tpcrdm.key1.keys
tpcrdm.key2.keys
tpvrdm.dat
tpcrdm.vsn.keys
```

4. Update the database files created in step 3 by using the following commands:

```
dmdbrecover -n daemon_name dmd_db
dmdbrecover -n ls_name libsrv_db
```

DMF Client SOAP Service

This chapter discusses the following:

Overview of DMF Client SOAP

DMF provides access to the following functions via the DMF client Simple Object Access Protocol (SOAP) web service:

- `dmattr`
- `dmget`
- `dmput`
- `dmtag`
- `dmversion`

Note: A limited set of options are available for these commands via DMF Client SOAP. For more information, click on the operation name in the SOAP interface and read the information under the **Documentation** heading displayed.

Figure 15-1 shows an example of the `ws_dmattr` operation.

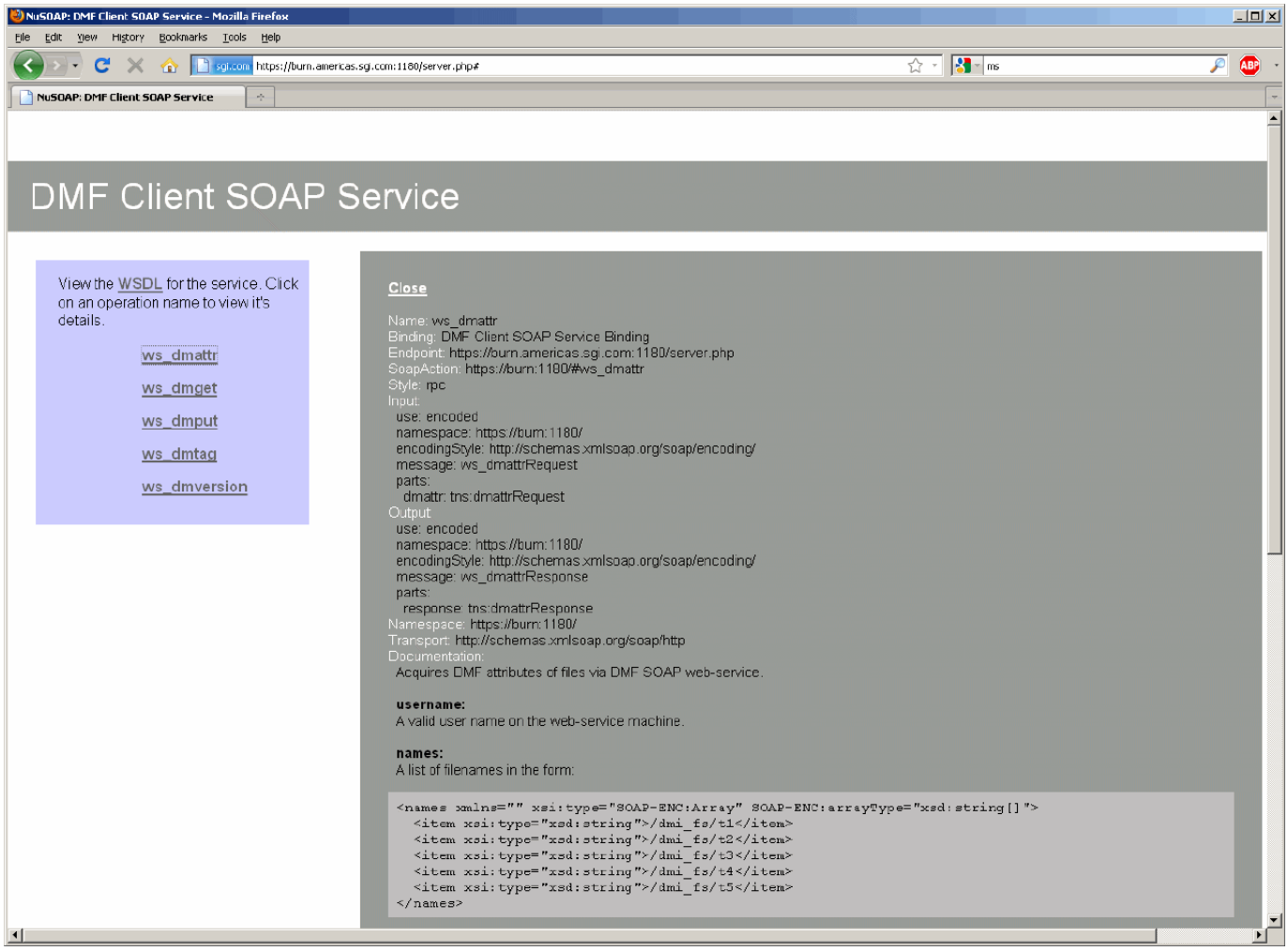

Figure 15-1 DMF Client SOAP Service

Accessing the DMF Client SOAP Service and WSDL

To access DMF Client SOAP, do the following:

1. Point your browser to the following secure address:

   `https://`*YOUR_DMF_SERVER*`:1180/server.php`

2. Accept the security certificate.

   Note: DMF Client SOAP generates its own SSL certificates, rather than having the SSL certificates signed by a commercial certificate authority. Therefore, the certificate warning is safe to ignore.

3. Enter the DMF Client SOAP service (`dmfsoap`) access password. The default password is `INSECURE`.

   To change the password to something site-specific (*NEWPASSWORD*), run the following command:

   # **`htpasswd2 -b -c /usr/share/dmfsoap/passwords/passwds dmfsoap`** ***NEWPASSWORD***

4. To access the web service definition language (WSDL) definition, click on **WSDL** in the interface. Use the brower's **Save As...** feature to save the WSDL to a file for consumption.

Starting and Stopping DMF Client SOAP

This section discusses the following:

Starting the `dmfsoap` Service

The `dmfsoap` service for DMF Client SOAP is off by default.

To start the service explicitly, execute the following on the DMF server:

```
dmfserver# service dmfsoap start
```

Preventing Automatic Start of dmfsoap After Reboot

To prevent automatic startup of the DMF environment, execute the following chkconfig(8) commands as root on the DMF server:

```
dmfserver# chkconfig dmfsoap off
```

Explicitly Stopping dmfsoap

To stop the DMF environment daemons explicitly, execute the following on the DMF server:

```
dmfserver# service dmfsoap stop
```

Security/Authentication

DMF SOAP leaves the security/authentication to the client and apache. By default, DMF SOAP will run over SSL to a password-protected virtual directory on the DMF server. You can customize or extend this functionality as desired.

DMF SOAP checks that the username supplied is a valid username on the system and executes the DMF commands as that user. However, no authentication beyond what is in the SOAP server and the client is done. The client has complete responsibility for user authentication.

Troubleshooting

This chapter contains the following:

Filesystem Errors

If the DMF administrative filesystems are not mounted when you try to apply configuration changes using DMF Manager or `dmmaint` or manually use `dmcheck`, you will see errors such as the following:

```
  ERROR:  Directory for JOURNAL_DIR (/dmf_journals/journals) does not exist.
  ERROR:  MOVE_FS "/dmf/move_fs" must be a filesystem root
  ERROR:  Filesystem "/dmi_fs" is not mounted.
  ERROR:  A DCM's STORE_DIRECTORY (/dmf/cache) must be a filesystem root.
  ERROR:  Filesystem "/" is not a DMAPI filesystem
  ERROR:  No such directory /dmf_journals/database_copies.
  ERROR:  OpenVault server is not up or client is misconfigured.
```

For example, following is the complete output from `dmcheck`:

```
# dmcheck

Checking DMF installation.
  Linux thud 2.6.16.60-0.21-default #1 SMP Tue May 6 12:41:02 UTC 2008
ia64 ia64 ia64 GNU/Linux - thud
  SuSE-release: SUSE Linux Enterprise Server 10 (ia64)
  SuSE-release: VERSION = 10
  SuSE-release: PATCHLEVEL = 2
  DMF version 4.3.0-ALPHA-20090717--dev2 installed.

Checking DMF config file
 Scanning for non-comment lines outside define/enddef pairs
 Scanning for DMF parameters without values
 Checking all objects for invalid names
 Checking base
  ERROR:  Directory for JOURNAL_DIR (/dmf_journals/journals) does not exist.
 Checking daemon
  ERROR:  MOVE_FS "/dmf/move_fs" must be a filesystem root
 Checking policy cache_policy
 Checking policy space_policy
 Checking policy chooser_policy
 Checking policy optional_chooser_policy
 Checking filesystem /dmi_fs
  ERROR:  Filesystem "/dmi_fs" is not mounted.
  WARNING:  Filesystem "/dmi_fs" inode size of 256 is inefficient for DMF.
```

```
 Checking filesystem /dmi_fs2
 Checking MSP msp
 Checking MSP cache (DCM-mode)
  ERROR:  A DCM's STORE_DIRECTORY (/dmf/cache) must be a filesystem root.
  ERROR:  Filesystem "/" is not a DMAPI filesystem
 Checking MSP cache1 (DCM-mode)
 Checking Library Server ov_lib
 Checking Resource Watcher rw
 Checking Drive Group ov_drv
 Checking Volume Group volume1
  WARNING:  Please consider setting ZONE_SIZE to improve write performance.
  See the dmf.conf(5) man page for more information.
 Checking Volume Group volume2
 Checking Resource Scheduler ov_drvrs
 Checking Services dmf_services
 Checking Task Group vgtasks
 Checking Task Group daemon_tasks
  ERROR:  No such directory /dmf_journals/database_copies.
 Checking Task Group dump_tasks
 Checking Task Group library_tasks
 Checking Task Group node_tasks
 Checking for unreferenced objects
  WARNING:  Unreferenced watcher rw.
 Cross-checking LSs and task groups for duplicate VSNs

Checking other daemons.
 Checking OpenVault
  ERROR:  OpenVault server is not up or client is misconfigured.
 Checking chkconfig

7 errors found.
3 warnings found.
```

To resolve these problems, you must make and mount the DMF administrative filesystems. See:

- "Overview of the Installation and Configuration Steps" on page 81
- "Configure DMF Administrative Filesystems and Directories Appropriately" on page 64

Unable to use the dmi Mount Option

By default, DMAPI is turned off on SLES 10 systems. If you try to mount with the dmi mount option, you will see errors such as the following:

```
kernel: XFS: unknown mount option [dmi]
```

See "DMAPI_PROBE Must Be Enabled for SLES 10 or SLES 11 Nodes When Using CXFS" on page 90.

EOT Error

A message of the following type means that there was no logical end-of-tape (EOT) mark written to tape:

```
05:47:26-E   382537-dmatwc end_tape: NOTE: An EOT was not written to VSN 057751 prior to close
```

When DMF appends data to a tape, it positions to the EOT chunk in the EOT zone. Without a valid EOT chunk in the EOT zone, DMF might not be able to append to the tape; this may eventually cause the HVFY flag to be set. Set the hsparse flag on the tape to merge all the data off of the volume.

Tape Drive Not Claimed by ts

If a tape drive is not claimed by ts, see the /var/log/messages file for an indication as to why ts did not attach to a device.

Drive Entry Does Not Correspond to an Existing Drive (OpenVault)

If OpenVault starts before an HBA has discovered the tape devices, the devices will be unusable by OpenVault. In this case, you would see a message similar to the following:

```
Drive lto1_3 DCP lto1_3@boom config file scsi: entry does not correspond to an existing drive
```

You must add the HBA driver to the /etc/sysconfig/kernel file and restart OpenVault. See "Add HBA Drivers to the initrd Image" on page 72.

Drive Does Not Exist (TMF)

If a drive is not visible to TMF, it may be because an HBA device was not properly discovered. In this case, there would be a message in `/var/spool/tmf/daemon.stderr` such as the following:

```
File /dev/ts/pci0002:00:01.1/fc/500104f000700269-500104f00070026a/lun0 does not exist
```

You must add the HBA driver to the `/etc/sysconfig/kernel` file and restart TMF. See "Add HBA Drivers to the `initrd` Image" on page 72.

DMF Manager Error Messages

This section describes problems you may encounter when monitoring DMF with DMF Manager:

- "DMF Statistics are Unavailable or DMF is Idle" on page 373
- "OpenVault Library Is Missing" on page 374

Also see "Filesystem Errors" on page 370.

DMF Statistics are Unavailable or DMF is Idle

```
This screen requires statistics from DMF that are unavailable;
check that DMF is running, including the "pmdadmf2"process.
Make sure the DMF "EXPORT_METRICS" configuration parameter is enabled.
```

This message indicates that DMF is idle. When this occurs, perform the following procedure:

1. Check the version of DMF by running the `dmversion` command.
2. Check that the `EXPORT_METRICS on` line has been added to `/etc/dmf/dmf.conf` after the `TYPE base` line.

   Run `dmcheck` to search the DMF configuration file for syntactic errors.
3. Check that DMF has been restarted after the change to `/etc/dmf/dmf.conf` was made in step 2.

4. Check that the data is being exported by DMF by running the following command:

   # **dmarenadump -v**

   If it is not, run the following commands as `root` to restart DMF, PCP, and DMF Manager:

   ```
   # cd /dmf/spool   # or equivalent at your site
   # rm base/arena
   # /etc/init.d/dmf restart
   # /etc/init.d/pcp stop
   # /etc/init.d/pcp start
   # /etc/init.d/dmfman restart        # if necessary
   ```

5. Check that the data is passing through PCP by running the following command:

   # **pminfo -f dmf2**

   If it is not, run the following commands as `root` to remove and reinstall the PCP performance metrics domain agents and restart DMF Manager:

   ```
   # cd /var/lib/pcp/pmdas/dmf2
   # ./Remove
   # ./Install
   # /etc/init.d/dmfman restart
   ```

OpenVault Library Is Missing

```
No OpenVault-controlled library found.
```

This indicates that OpenVault is not running. Run the following command to verify that the `ov_stat` command is available:

```
# ls -lL /usr/bin/ov_stat
-rws--x--x 1 root sys 322304 Jul 22 2005 /usr/bin/ov_stat
```

If the file permissions are not `-rws--x--x` as shown above, run the following command to change the permissions:

```
# chmod 4711 /usr/bin/ov_stat
```

Delay In Accessing Files in an SMB/CIFS Network Share

If there is a delay in accessing files in an SMB/CIFS network share, it may be because the files are in a fully or partially offline state. The Windows Explorer desktop can be enabled to display a small black clock on top of a migrated file's normal icon; the black clock symbol indicates that there may be a delay in accessing the contents of the file. (This feature is disabled by default.) For more information, see "Modify Settings If Providing File Access via Samba" on page 79.

Operations Timeout or Abort on Windows®

Operations such as `cp` can timeout on Windows systems or abort with the following message:

```
couldn't locate the origin file
```

This may occur if the `SessTimeout` parameter is set to a value that is inappropriate for a DMF environment. See "Modify Settings If Providing File Access via Samba" on page 79.

Windows Explorer Hangs

If the Windows Explorer hangs and the `no response ...` message appears in the Windows main title, it may be because the `SessTimeout` parameter is set to a value that is inappropriate for a DMF environment. See "Modify Settings If Providing File Access via Samba" on page 79.

Poor Migration Performance

If you encounter poor migration performance, you can try to tune DMF's direct I/O size by modifying the `DIRECT_IO_SIZE` parameter for the `filesystem` object in the DMF configuration file (`/etc/dmf/dmf.conf`).

You can also try switching to buffered I/O migration by setting the `MIN_DIRECT_SIZE` parameter to a very large value.

See "`filesystem` Object" on page 187.

Remote Connection Failures

If there are an insufficient number of `xinetd tcpmux` instances configured, you may see remote connection failures. If this condition occurs, you will see messages like the following in the `/var/log/xinetd.log` file:

```
10/3/2@13:41:09: FAIL: tcpmux service_limit from=128.162.246.75
```

To solve this problem, see "Set the `xinetd tcpmux instances` Parameter Appropriately" on page 72.

Using SGI Knowledgebase

If you encounter problems and have an SGI support contract, you can log on to Supportfolio and access the Knowledgebase tool to help find answers.

To log in to Supportfolio Online, see:

https://support.sgi.com/login

Then click on **Search the SGI Knowledgebase** and select the type of search you want to perform.

If you need further assistance, contact SGI Support.

Reporting Problems to SGI

As soon as you suspect a problem with DMF, run the following commands as `root` to gather relevant information about your DMF environment that will help you and SGI analyze the problem:

- Run the following command on the DMF server and every parallel data mover node in order to gather system configuration information:

  # **`/usr/sbin/system_info_gather -A -o`** ***`nodename`*****`.out`**

- Run the following command once on the DMF server to collect information for today and the specified number of additional days (*extra-days* must be a numerical value greater than or equal to `0`):

  # **`dmcollect`** ***`extra-days`***

Note: Take care to enter the correct number of previous days from which to gather information, so that logs containing the first signs of trouble are included in the collection.

See the `dmcollect`(8) man page for additional information.

When you contact SGI Support, you will be provided with information on how and where to upload the collected information files for SGI analysis.

Messages

This appendix discusses the following:

If you are uncertain about how to correct these errors, contact your customer service representative.

CAT Table and Daemon Database Comparisons

Error messages generated when comparing catalog (CAT) records in the LS database to the daemon database will start with the following phrase:

`Bfid` *bfid* `-`

The *bfid* is the bit file ID associated with the message.

The preceding phrase will be completed by one or more of the following phrases:

```
missing from cat db
missing from daemon db
for vsn volume_serial_number chunk chunk_number msg1 msg2
```

In the above, *msgn* can be one of the following:

```
filesize < 0
chunkoffset < 0
chunklength < 0
zonenumber < 0
chunknumber <0
filesize < chunklength + chunkoffset
zonenumber
missing or improper vsn
filesize != file size in daemon entry (size)
```

no chunk for bytes *msg1*, *msg2*

In the above, *msgn* gives the byte range as *nnn - nnn*

nnn bytes duplicated

VOL and CAT Record Comparisons

Error messages generated when comparing the volume (VOL) records to CAT records in the LS database will start with the following phrase:

Vsn *vsn*

The *vsn* is the volume serial number associated with the message.

The preceding phrase will be completed by one or more of the following phrases:

missing

eotpos < largest position in cat (3746)
eotchunk < largest chunk in cat (443)
eotzone < largest zone in cat (77)
chunksleft != number of cat chunks (256)
dataleft !=sum of cat chunk lengths (4.562104mb)

tapesize is bad
version is bad
blocksize is bad
zonesize is bad
eotchunk < chunksleft
dataleft > datawritten

volume is empty but *msg1*, *msg2*

In the above, *msgn* can be one of the following:

hfull is on
hsparse is on
datawritten != 0
eotpos != 1/0
eotchunk != 1

`volume is not empty but` *msg1*`,` *msg2*

In the above, *msgn* is one of the following:

```
hfree is on
version < 4 but msg1, msg2
```

In the above, *msgn* can be one of the following:

```
volume contains new chunks
hfull is off
eotpos !=2/0
```

dmcatadm Message Interpretation

The following lists the meaning of messages associated with the CAT records in the LS database:

nnn bytes duplicated in volume group name	Two or more chunks in the database, which belong to volume group (VG) name, contain data from the same region of the file.
for vsn DMF001 chunk 77 chunkoffset < 0	The chunkoffset value for chunk 77 on volume serial number (VSN) DMF001 is obviously bad because it is less than 0.
for vsn DMF001 chunk 77 chunklength < 0	The chunklength value for chunk 77 on VSN DMF001 is obviously bad because it is less than 0.
for vsn DMF001 chunk 77 chunknumber < 0	The chunknumber value for chunk 77 on VSN DMF001 is obviously bad because it is less than 0.
for vsn DMF001 chunk 77 filesize < 0	The filesize value for chunk 77 on DMF001 is obviously bad because it is less than 0.

`for vsn DMF001 chunk 77 filesize < chunklength + chunkoffset`	The value of `chunklength` plus `chunkoffset` should be less than or equal to the `filesize`. Therefore, one or more of these values is wrong.
`for vsn DMF001 chunk 77 missing or improper vsn`	The list of volume serial numbers for the chunk is improperly constructed. The list should contain one or more 6-character names separated by colons.
`for vsn DMF001 chunk 77 zonenumber < 0`	The `zonenumber` value for chunk 77 on DMF001 is obviously bad because it is less than 0.
`for vsn DMF001 chunk 77 zonenumber > chunknumber`	Either the `zonenumber` value or the `chunknumber` value for chunk 77 on DMF001 is wrong, because the `zonenumber` is larger than the `chunknumber` value. (Each zone contains at least two chunks, because the end-of-zone header on the tape counts as a chunk.)
`for vsn DMF001 chunk 77 filesize != file size in daemon entry (nnn)`	The `filesize` value in the chunk entry is different from the file size in the daemon record. If no daemon record was provided, this message indicates that more than one chunk exists for the BFID and that the `filesize` value is not the same for all the chunks.
`missing from cat db`	No corresponding CAT record was found for an existing daemon record.
`entry for volume group name missing from daemon db`	No corresponding daemon record was found for an existing CAT record.
`for volgrp name; no chunk for bytes nnn - nnn`	There is no chunk that contains the specified bytes of the file.

`dmvoladm` Message Interpretation

The following lists the meaning of messages associated with the VOL records in the LS database.

`blocksize is bad`	The `blocksize` field for the tape is less than or equal to 0.
`eotpos < largest position in cat (3746)`	The position for the EOT descriptor on the tape is less than the largest position of all the chunk entries for the tape.
`chunksleft != number of cat chunks (256)`	The number of chunks referencing the tape in the CAT table does not equal the number of chunks left recorded in the VOL entry for the tape.
`dataleft != sum of cat chunk lengths (4.562104mb)`	The sum of the chunks length for chunks referencing the tape in the CAT table does not equal the `dataleft` value recorded in the VOL entry for the tape.
`dataleft > datawritten`	The entry shows that more data remains on the tape than was written.
`eotchunk < chunksleft`	The entry shows that more chunks remain on the tape than were written.
`eotchunk < largest chunk in cat (443)`	The chunk number of the EOT descriptor on the tape is less than the largest chunk number of all the chunk entries for the tape.
`eotzone < largest zone in cat (77)`	The zone number of the EOT descriptor on the tape is less than the largest zone number of all the chunk entries for the tape.
`missing`	The volume was found in a chunk entry from the CAT table but is not in the VOL table.

`tapesize is bad`	The `tapesize` field for the tape is an impossible number.
`version is bad`	The `version` field for the tape is not 1 or 3 (for a tape still containing data written by an old MSP) or 4 (for a tape written by this MSP).
`volume is empty but hfull is on` `volume is empty but hsparse is on`	When a volume is empty, the `hfull`, and `hsparse` hold flags should be `off`.

`volume is empty but datawritten != 0`
`volume is empty but eotpos != 1/0`
`volume is empty but eotchunk != 1`
	When the `hfree hold` flag is cleared, the `datawritten` field is set to 0, the `eotpos` field is set to 1/0, and the `eotchunk` is set to 1. The entry is inconsistent and should be checked.

`volume is not empty but hfree is on`	When a volume contains data, the `hfree hold` flag must be `off`.
`volume is not empty and version is` *n* `but hfull is off`	Tapes containing data with a version value of less than 4 must have `hfull` set, because the LS cannot append to the tape.
`volume is not empty and version is` *n* `but eotpos != 2/0`	Tapes imported from the old MSP only have one zone of data, so `eotpos` must be 2/0.
`zonesize is too small`	The `zonesize` field for the tape is an impossible number.

DMF User Library `libdmfusr.so`

The subroutines that constitute the DMF user command application program interface (API) are available to user-written programs by linking to the DMF user library, `libdmfusr.so`. Sites can design and write their own custom DMF user commands, which eliminates the need to use wrapper scripts around the DMF user commands.

This appendix discusses the following:

Overview of the Distributed Command Feature and `libdmfusr.so`

The distributed command feature allows DMF commands to execute on a host other than the host on which the DMF daemon is running. (This feature was first made available with DMF 2.7.) A host that imports DMF-managed filesystems from the DMF daemon host machine can execute the DMF commands locally as defined in "DMF Clients" on page 10. The distributed command feature requires `tcpmux` (RFC 1078).

The DMF user commands communicate with a process named `dmusrcmd`, which is executed as `setuid root`. `dmusrcmd` performs validity checks and communicates with the DMF daemon. (In releases prior to DMF 2.7, user commands communicated directly with the DMF daemon and were installed as `setuid root` processes.)

In order for the DMF user commands to communicate in an efficient and consistent manner with the dmusrcmd process, they must access the DMF user library, which is installed in the following location according to platform operating system and architecture:

Platform	DMF User Library Location
irix-n32	/usr/lib32/libdmfusr.so[.*n*]
irix-64	/usr/lib64/libdmfusr.so[.*n*]
Linux ia64	/usr/lib/libdmfusr.so[.n]

Note: The old version of libdmfusr is located in /usr/lib/dmf/libdmfusr_v1 in order to prevent ldconfig(8) from updating the /usr/lib/libdmfusr.so symbolic link to point to the old library. Customers requiring the version 1 library can make use of it with the following steps:

```
# cd /usr/lib/dmf/libdmfusr_v1
# ln -s libdmfusr.so.1 libdmfusr.so
# export LD_LIBRARY_PATH=/usr/lib/dmf/libdmfusr_v1
```

Linux x86_64	/usr/lib64/libdmfusr.so[.*n*]
Solaris	/usr/lib/sparcv9/libdmfusr.so[.*n*]
Mac OS X	/usr/lib/libdmfusr.[*n*].dylib

Each of the DMF user commands is linked to the library for its protocol-based communications. (The DMF user library became a versioned shared-object library in DMF 3.1. See "libdmfusr.so Library Versioning" on page 388 for more information on accessing the correct version of libdmfusr.so.)

The underlying design of the API calls for the user command to make contact with a dmusrcmd process by creating an opaque context object via a call to the API. This context is then used as a parameter on each function API call (put, get, fullstat, or copy). The context is used by each API subroutine to perform the requested operation and to correctly return the results of the operation to the command.

In addition to the library, the libdmfusr.H, libdmfcom.H, and dmu_err.h header files are provided. These files are required for sites to effectively create their own commands. All header files are installed in /usr/include/dmf. The libdmf* header files contain all of the object and function prototype definitions required by

the API subroutine calls. The `dmu_err.h` file contains all of the API error code definitions. Along with each error code definition is a text string that is associated with each of the error codes. This text string is the same message that is generated automatically when the error occurs as part of the `DmuErrInfo_t` object (see "`DmuErrInfo_t`" on page 399). The text string is included in the file as informational only, and is not accessible by a program that includes `dmu_err.h`.

Each type of function request (`put`, `get`, `fullstat`, or `copy`) can be made via a synchronous or an asynchronous API subroutine call:

- Synchronous subroutine calls do not return to the caller until the request has completed, either successfully or unsuccessfully. These synchronous subroutines return an error object to the caller that can be processed to determine the success or failure of the call. If an application is making more than one call, these calls will usually perform less efficiently than their asynchronous counterparts because of the serial nature of their activity.
- Asynchronous subroutine calls return immediately to the caller. The return codes of these asynchronous subroutines indicate whether the request was successfully forwarded to `dmusrcmd` for processing. A successful return allows the calling program to continue its own processing in parallel with the processing being performed by `dmusrcmd` (or the DMF daemon) to complete the request. If the request was successfully forwarded, a request ID that is unique within the scope of the opaque context is returned to the caller. It is the responsibility of the caller to associate the request ID with the correct completion object (described in "`DmuCompletion_t`" on page 397) to determine the eventual result of the original request.

There are several API subroutine calls for processing asynchronous request completion objects. The user can choose to do any of the following:

- Be notified when all requests have completed without processing the return status of each request.
- Process the return status of each request in the order in which they complete.
- Wait synchronously on an individual asynchronous request's completion by specifying the request ID on which to wait. By using this method, each request return status can be processed in the order in which it was sent, known as *request ID order*.

The API includes well-defined protocols that it uses to communicate with the `dmusrcmd` process. Because these protocols make use of the `pthreads`(5)

mechanism, any user application program making use of the API via libdmfusr.so must also link to the libpthread.so shared object library via one of the following:

-lpthread compiler option using cc(1) or CC(1)
-lpthread loader option using ld(1) or rld(1)

In many cases, the API subroutines pass the address of an object back to the caller by setting a ** pointer accordingly. If errors occur and the subroutine is unable to complete its task, the address returned may be NULL. It is up to the caller to check the validity of an object's address before using it in order to avoid causing a SIGSEGV fault in the application program.

Considerations for IRIX®

The DMF user library for each IRIX platform (lib32 and lib64) was compiled using a MIPSpro™ compiler. Compiling user applications that call DMF user library API subroutines with compilers other than MIPSpro compilers may result in incompatibilities causing load-time or run-time errors.

libdmfusr.so Library Versioning

Caution: The old libdmfusr.so.1 version of the DMF library described below will be removed in a future release. Customers should recompile their applications to use the new library.

DMF 3.1 introduced a new version of the DMF user library. This new version is not compatible with the previous library nor with applications that were written and linked with the previous library. To allow the use of older applications after installing the current version of DMF and to facilitate upgrading older applications, the current version of DMF provides both the old version and the new version and introduces a linking mechanism.

When an application is created and linked with a shared object, the name of the actual library that the application is ultimately linked with is stored in the executable file and used at execution time to find a library of the same name for dynamic linking. In previous releases, the library was named libdmfusr.so. Therefore, all existing DMF commands and site-developed applications that use the library contain the filename libdmfusr.so in the executable for linking with the library at execution time.

A common practice when creating a new version of a library is to add the suffix *.n* to the library name, where *n* is an ever-increasing integer that refers to the current version number.

Prior to DMF 3.1, the library named `libdmfusr.so` was an actual library, rather than a link to a library. The current version of DMF provides the old library (renamed `libdmfusr.so.1`) and the new library (named `libdmfusr.so.2`). All current DMF user commands (such as `dmput`) were created and linked with `libdmfusr.so.2` and their executables contain the filename `libdmfusr.so.2` for linking with the library.

The `libdmfusr.so.1` library is identical to the `libdmfusr.so` library shipped prior to DMF 3.1. The current DMF installation process will install a link named `libdmfusr.so` that will point to `libdmfusr.so.2`. If needed, you can change the link to point to `libdmfusr.so.1` in order to satisfy linking for executables built with a pre-DMF 3.1 `libdmfusr.so`.

The locations of the libraries and the link have not changed from previous releases (see "Overview of the Distributed Command Feature and `libdmfusr.so`" on page 385).

The new `libdmfusr.so` link provides the following advantages:

- You can use the default setting, which does not require any knowledge about the latest version of the library. When developing new site applications using the library, the non-version-specific `ld` option `-ldmfusr` will result in the loader following the link and using the new version of the library, `libdmfusr.so.2`. The resulting applications will contain the name `libdmfusr.so.2` in their executable files for dynamic loading.
- You can reset the link to point to `libdmfusr.so.1`, which allows existing site-developed applications to continue to work with the older version of the library. This will not affect any of the DMF user commands because they contain the name of the new library and make no use of the link at execution time. When an older application executes, if filename `libdmfusr.so` is encountered by the loader and the link points to `libdmfusr.so.1`, the application will continue to work exactly as it did before the current DMF installation.

The two uses of the link as described above are mutually exclusive of each other. Take care when using the link to enable older applications to run with the old library while at the same time developing new applications using the new library. If the link points to `libdmfusr.so.1` and `-ldmfusr` is used to create a new application, the older version of the library will be found and the resulting executable will contain the filename `libdmfusr.so.1` for use at execution time. If older applications are required to run correctly while new applications are being developed, you must use

specific loader command options to ensure that the new applications are linked with the latest library. This can be done by including the specific library name, such as libdmfusr.so.2, on the ld or cc command instead of the generic library specification -ldmfusr.

libdmfusr.so.2 Data Types

The data types described in this section are defined in libdmfusr.H or libdmfcom.H. For the most up-to-date definitions of each of these types, see the appropriate file. The following information is provided as a general description and overall usage outline.

All of the data types defined in this section are C++ objects, and all have constructors and destructors. Many have copy constructors and some have operator override functions defined. Please refer to the appropriate .H header file to see what C++ functions are defined for each object in addition to the member functions described in this section.

DmuAllErrors_t

The DmuAllErrors_t object provides the caller with as much information regarding errors as is practical. The complex nature of the API and its communications allows for many types of errors and several locations (processes) in which they can occur. For example, a request might fail in the API, in the dmusrcmd process, or in the DMF daemon.

The public member fields and functions of this class are as follows:

entry	Specifies a read-only pointer allowing access to all DmuErrInfo_t entries in the DmuAllErrors_t internal array.
numErrors()	Returns the number of DmuErrInfo_t entries in the DmuAllErrors_t internal array.
resetErrors()	Clears the DmuAllErrors_t internal array.

Following is an example using a DmuAllErrors_t object.

Note: The following code is a guideline. It may refer to elements of a `DmuAllErrors_t` structure that are not defined in your installed version of `libdmfcom.H`.

```
report_errors(DmuAllErrors_t  *errs)
{
        int             i;

        if (!errs) {
                return;
        }
        for (i = 0; i < errs->numErrors(); i++) {
                fprintf(stdout, "group '%s' errcode '%d' who '%s' "
                        "severity '%s' position '%s' host '%s' message '%s'\n",
                        errs->entry[i].group ? errs->entry[i].group : "NULL",
                        errs->entry[i].errcode,
                        DmuLogGetErrWhoImage(errs->entry[i].errwho),
                        DmuLogGetSeverityImage(errs->entry[i].severity),
                        errs->entry[i].position ? errs->entry[i].position : "NULL",
                        errs->entry[i].host ? errs->entry[i].host : "NULL",
                        errs->entry[i].message ? errs->entry[i].message : "NULL");
        }
}
```

DmuAttr_t

The DmuAttr_t object defines the DMF attribute for a DMF-managed file.

The public member fields and functions of this class are as follows:

bfid — Specifies a DmuBfid_t object (defined in libdmfcom.H) that defines the file's bitfile-ID (bfid).

fsys — Specifies a DmuFileIoMethod_t object (defined in libdmfcom.H) that defines the file's filesystem type.

version — Specifies a DmuFileIoVersion_t object (defined in libdmfcom.H) that defines the filesystem version.

dmstate — Specifies a dmu_state_t object that defines the file state. Valid states are:

DMU_ST_REGULAR	Regular
DMU_ST_MIGRATING	Migrating
DMU_ST_DUALSTATE	Dual-state
DMU_ST_OFFLINE	Offline

	DMU_ST_UNMIGRATING	Unmigrating
	DMU_ST_NOMIGR	No migration allowed
dmflags	Specifies an integer defining a file's DMAPI flags. Currently unused.	
sitetag	Defines the file site tag value. See dmtag(1).	
regbuf	Specifies a DmuFullRegbuf_t object that defines the file full region information. See "DmuFullRegbuf_t" on page 402.	

DmuByteRange_t

The DmuByteRange_t object defines a range of bytes that are to be associated with a put or get request.

The public member fields and functions of this class are as follows:

start_off	Starting offset in bytes of the range in the file.
end_off	Ending offset in bytes of the range in the file.

Nonnegative values for start_off or end_off indicate an offset from the beginning of the file. The first byte in the file has offset 0. Negative values may be used to indicate an offset from the end of the file. The value -1 indicates the last byte in the file, -2 is the next-to-last byte, and so on. The range is inclusive, so if start_off has a value of 2 and end_off has a value of 2, it indicates a range of one byte.

DmuByteRanges_t

The DmuByteRanges_t object defines a set of DmuByteRange_t objects that are to be associated with a put or get request.

The public member fields and functions of this class are as follows:

rounding

Specifies the rounding method to be used to validate range addresses. Only DMU_RND_NONE is valid.

`entry`

Specifies a read-only pointer allowing access to all `DmuByteRange_t` entries in the `DmuByteRanges_t` internal array.

`numByteRanges()`

Returns the number of `DmuByteRange_t` objects contained in the entry array.

`resetByteRanges()`

Resets the number of `DmuByteRange_t` objects in the array to zero.

`setByteRange()`

Adds a new range. If the range being added overlaps or is adjacent to an existing range in the array, the items may be coalesced. It is expected that the starting offset not be closer to the end-of-file than the ending offset. For example, a starting offset of `5` and an ending offset of `4` is invalid, and the `setByteRange()` function may not add it to the array. The `setByteRange()` function cannot determine the validity of some ranges, however, and may add ranges that the `put` or `get` request will later ignore.

`fromByteRangesImage()`

Converts a string that represents a byte range and adds it to the `DmuByteRanges_t` object. Strings that represent byte ranges are described on the `dmput` man page.

Note: In a string representing a byte range, `-0` represents the last byte in the file, while in a `DmuByteRange_t` object, `-1` represents the last byte in the file.

For example, suppose `byteranges` is declared as the following:

```
DmuByteRanges_t byteranges;
```

Then each of the following statements will add the `DmuByteRange_t` object that covers the entire file:

```
byteranges.setByteRange(0,-1);
byteranges.fromByteRangesImage("0:-0" ,&errstr);
```

If the byte range overlaps or is adjacent to an existing range in the array, the items may be coalesced.

`clearByteRange`

Clears the specified byte range in the `DmuByteRanges_t` object. The `clearByteRange()` routine is restricted in how it handles negative offsets, both in the `DmuByteRange_t` members of the `DmuByteRanges_t` class and in its parameters. The following items give the details of these restrictions. In the following items, *start* and *end* are the parameters to the `clearByteRange()` routine, using the following format:

`clearByteRange(`*start*`,`*end*`)`

- If *start* and *end* exactly match a `DmuByteRange_t` entry, then that entry will be cleared. This includes negative numbers.
- If *start* is `0` and *end* is `-1`, all `DmuByteRange_t` entries will be cleared. `resetByteRanges()` is the preferred method for clearing all ranges.
- If *start* is positive and *end* is `-1`, then:
  - All `DmuByteRange_t` entrys that have a positive `start_off` value greater than or equal to *start* will be cleared
  - All `DmuByteRange_t` entrys that have a positive `start_off` value that is less than *start* and an `end_off` value of `-1` will be changed to have an `end_off` value of *start-1* (that is, *start* minus 1). For example, if `DmuByteRanges_t` has a single range, `3:-1`, then `clearByteRange(4,-1)` will leave a single range, `3:3`.
  - All `DmuByteRange_t` entrys that have a positive `start_off` value that is less than *start* and an `end_off` value that is greater than *start* will be changed to have an `end_off` value of *start-1*. For example, if `DmuByteRanges_t` has a single range `3:9`, then `clearByteRange(4,-1)` will leave a single range `3:3`.
- If *start* and *end* are both positive and a `DmuByteRange_t` entry has positive `start_off` and `end_off` values, then the range specified by *start* and *end* is cleared from the `DmuByteRange_t`.

- If *start*, *end*, and the start_off and end_off values of a DmuByteRange_t are all negative, the range specified is cleared from DmuByteRange_t.

You can create a valid DmuByteRanges_t object using the default constructor with or without the new operator, depending on the need. For example:

```
DmuByteRanges_t         ranges;

DmuByteRanges_t         *ranges = new DmuByteRanges_t;
```

The following example creates a DmuByteRanges_t named byteranges, adds a DmuByteRange_t to it, then prints the entry to stdout:

```
DmuByteRanges_t byteranges;
int             i;
byteranges.rounding = DMU_RND_NONE;
byteranges.setByteRange(0, 4095); /* specifies the first 4096 bytes in the file */
for (i = 0; i < byteranges.numByteRanges(); i++) {
        fprintf(stdout,"Starting offset %lld, ending offset %lld\n",
                byteranges.entry[i].start_off,
                byteranges.entry[i].end_off);
}
```

The output to stdout would be as follows:

```
starting offset 0, ending offset 4095
```

The following example creates a DmuByteRanges_t named b, adds a DmuByteRange_t to it, then clears a byte range:

```
DmuByteRanges_t b;
int i;
b.setByteRange(0,40960);
b.clearByteRange(4096,8191);
printf("Num byte ranges %d\n",b.numByteRanges());
for (i = 0; i < b.numByteRanges(); i++)
        printf("%lld %lld\n",b.entry[i].start_off, b.entry[i].end_off);
```

The output to stdout would be as follows:

```
Num byte ranges 2
0 4095
8192 40960
```

Note: The `toByteRangesImage()` member function is not yet supported.

DmuCompletion_t

The `DmuCompletion_t` object is returned by one of the API request completion subroutines (see "Request Completion Subroutines" on page 426) with the results of an asynchronous request.

The public member fields and functions of this class are as follows:

`request_id`	Associates the completion object with an asynchronous request that was previously issued. This value coincides with the request ID value that any of the asynchronous subroutines return to the user.
`request_type`	Specifies the type of the original request.
`reply_code`	Contains the overall success or failure status of the request. If this value is `DmuNoError`, the request was successful. If not, the `allerrors` field should be checked for the appropriate error information.
`ureq_data`	Specifies a pointer to user request-type specific data. For a `fullstat` user request, this will point to a `DmuFullstat_t` object. This field has no meaning for `put`, `get`, or `copy` user requests.
`fhandle`	Specifies the file handle of the file associated with the request.

DmuCopyRange_t

The `DmuCopyRange_t` object defines a range of bytes that are to be associated with a `copy` request.

The public member fields and functions of this class are as follows:

`src_offset`	Specifies the starting offset in bytes of the range in the source file to be copied.
`src_length`	Specifies the length in bytes of the range to be copied.

dst_offset	Specifies the starting offset in bytes in the destination file to which the copy is sent.

DmuCopyRanges_t

The DmuCopyRanges_t class defines an array of DmuCopyRange_t objects that are to be associated with a copy request.

The public member fields and functions of this class are as follows:

rounding	Specifies the rounding method to be used to validate range addresses. Only DMU_RND_NONE is supported.
entry	Specifies a read-only pointer allowing access to all the DmuCopyRange_t entries in the array.
numCopyRanges()	Returns the number of DmuCopyRange_t objects contained in the entry array. Only a single range is supported.
setCopyRange	Adds a new DmuCopyRange_t object to the array.
resetCopyRanges()	Resets the number of DmuCopyRange_t objects in the array to zero.

Example: Create a DmuCopyRanges_t, add a DmuCopyRange_t to it, then print the entry to stdout:

```
DmuCopyRanges_t copyranges;
int             i;

copyranges.rounding = DMU_RND_NONE;
copyranges.setCopyRange(0, 4096, 0);

for (i = 0; i < copyranges.numCopyRanges(); i++) {
        fprintf(stdout, "source offset %llu, length %llu, "
                "destination offset %llu\n",
                copyranges.entry[i].src_offset,
                copyranges.entry[i].src_length,
                copyranges.entry[i].dst_offset);
}
```

DmuErrHandler_f

The DmuErrHandler_f object defines a user-specified error handling subroutine. Many of the API subroutines may result in the receipt of error information from the dmusrcmd process or the DMF daemon in the processing of the request. As these errors are received, they are formatted into a DmuErrInfo_t object (see "DmuErrInfo_t" on page 399) and are generally returned to the caller either via a calling parameter or as part of a DmuCompletion_t object.

In addition, however, if the error occurs in the course of processing internal protocol messages, the DmuErrInfo_t object can also be passed into the DmuErrHandler_f that the caller defined when the opaque context was created.

As part of the DmuCreateContext() API subroutine call, the caller can specify a site-defined DmuErrHandler_f subroutine or the caller can use one of the following API-supplied subroutines:

DmuDefErrHandler	Outputs the severity of error and the message associated with the error to stderr.
DmuNullErrHandler	Does nothing with the error.

DmuErrInfo_t

The DmuErrInfo_t object contains the information about a single error occurrence.

The public member fields and functions of this class are as follows:

group	Defines the originator of the error: sgi_dmf (DMF routine) sgi_dmf_site (site-defined policy routine)
errcode	Specifies an integer value generated by the originating routine. This code may have many different meanings for a single value, depending on who the originator is.
errwho	Specifies an integer value that describes in more detail the originator of the error. Use the DmuLogGetErrWhoImage() subroutine to access a character string corresponding to this value .
severity	Specifies an integer value that describes the severity of the error. Use the DmuLogGetSeverityImage()

	subroutine to access a character string corresponding to this value.
position	Specifies a character pointer to a string that contains the position of where the error was generated. For example, this could be a pointer to a character string generated using the __FILE__ and __LINE__ cpp(1) macros. This field may be NULL.
host	Specifies a character pointer to a string that contains the hostname where the error originated.
message	Specifies a character pointer to a string that contains the body of the error message.

DmuError_t

The DmuError_t object is the type that most of the API subroutines pass as a return code. The definition DmuNoError is the general success return code.

DmuEvents_t

The DmuEvents_t object defines the various event mask settings that a file may contain.

Valid settings are defined as the logical OR of any of the following:

DMF_EVENT_READ	Generates a kernel event for each read request on the file.
DMF_EVENT_WRITE	Generates a kernel event for each write request on the file.
DMF_EVENT_TRUNCATE	Generates a kernel event for each truncate request on the file.
DMF_EVENT_DESTROY	Generates a kernel event will be generated for each destroy request on the file.

DmuFhandle_t

The DmuFhandle_t object contains the ASCII representation of the file fhandle as it is known on the host on which the file's filesystem is native.

The public member fields and functions of this class are as follows:

`hanp`	Specifies a character array containing the file handle.
`fromFhandleImage()`	Copies an ASCII file handle image string into the `hanp` field.
`toFhandleImage()`	Copies the `hanp` field into a `DmuStringImage_t` object.
`is_valid()`	Verifies the validity of the `hanp` field.

DmuFsysInfo_t

The `DmuFsysInfo_t` object contains the subset of DMF filesystem configuration information that may be relevant to a user command.

The public member functions of this class are as follows:

`is_configured()`	Returns `true` if the filesystem is defined in the DMF configuration file, either as a DMF-managed filesystem or an unmanaged filesystem.
`is_managed()`	Returns `true` if the filesystem is defined in the DMF configuration file and has a `MIGRATION_LEVEL` value other than `archive`. Files in DMF-managed filesystems can be used for all `libdmfusr.so` file request subroutines (such as `put` or `get`), with the exception that they cannot be the source file of an `archive` request (`DmuArchiveAsync`/`DmuArchiveSync`).
`is_unmanaged()`	Returns `true` if the filesystem is defined in the DMF configuration file and has a `MIGRATION_LEVEL` value of `archive`. Files in unmanaged filesystems can be used as the source of an `archive` request (`DmuArchiveAsync`/`DmuArchiveSync`) or the destination of a `copy` request (`DmuCopyAsync`/`DmuCopySync`). (However, see `min_archive_file_size()`.) Unmanaged filesystems do not support `put`, `get`, or `settag` requests, and cannot be used as the source of a `copy` request.

`min_archive_file_size()` Specifies the smallest file size that should be submitted in an `archive` request for this filesystem, or a `copy` request when this filesystem is the destination of the `copy` request. This only applies to filesystems for which `is_unmanaged()` is `true`.

DmuFullRegbuf_t

The `DmuFullRegbuf_t` object defines the DMF `fullregion` buffer information for a file Only a single region constituting of the whole file is supported.

The public member fields and functions of this class are as follows:

`arrcnt` Specifies the number of regions in the `regions` array.

`regcnt` Specifies the number of regions in the `regions` array that are valid. Only 0 and 1 are supported.

`regions` Specifies a `DmuFullRegion_t` array. See "`DmuRegion_t`" on page 403.

DmuFullstat_t

The `DmuFullstat_t` object is a user-accessible version of the internal DMF `fullstat` object. It contains all of the basic `stat`(2) information regarding the file, as well as all of the DMAPI-related fields.

The public member fields and functions of this class are as follows:

`inconsistent`	Indicates that the `DmuFullstat_t` object has inconsistencies in the fields.
`stat`	Specifies a `DmuStat_t` object that contains the fields representing those in the `stat`(5) structure. See the `stat`(2) system call.
`evmask`	Specifies a `DmuEvents_t` object that defines the event mask for the file. See "`DmuEvents_t`" on page 400
`regbuf`	Specifies a `DmuRegionbuf_t` object that defines the regions of the file. See "`DmuRegionbuf_t`" on page 403.
`attr`	Specifies a `DmuAttr_t` object that defines the DMF attribute of the file. See "`DmuAttr_t`" on page 392.
`host`	Specifies the hostname where the file is native.
`mntpt`	Specifies a `DmuOpaque_t` object (defined in `libdmfcom.H`) defining the mount point of the filesystem containing the file on `host`.
`relpath`	Specifies the relative path of the file in `mntpt` on `host`.
`is_valid()`	Returns `1` if the `DmuFullstat_t` is valid.

DmuRegion_t

The `DmuRegion_t` object defines a filesystem region.

The public member fields and functions of this class are as follows:

`rg_offset`	Defines the region starting offset in bytes. The start of the file is byte 0.
`rg_size`	Defines the region size in bytes.
`rg_flags`	Defines the region event flag bitmask. See "`DmuEvents_t`" on page 400.

DmuRegionbuf_t

The `DmuRegionbuf_t` object defines the filesystem region buffer information for a file. Only a single region constituting the whole file is supported.

The public member fields and functions of this class are as follows:

arrcnt	Specifies the number of regions in the regions array.
regcnt	Specifies the number of regions in the regions array that are valid. Only 0 and 1 are supported.
regions	Specifies the DmuRegion_t array. See the DmuRegion_t description.

DmuReplyOrder_t

The DmuReplyOrder_t object is used to select the order in which asynchronous replies are to be returned by the API reply processing subroutines.

Valid settings are defined as follows:

DmuAnyOrder	Returns replies in the order the replies are received.
DmuReqOrder	Returns replies in the order the requests were issued.

DmuReplyType_t

The DmuReplyType_t object is used to select the type of reply that an API can receive after sending a request. All requests will receive a final reply when the dmusrcmd process has completed processing the request whether it was successful or not.

Valid settings are defined as follows:

DmuIntermed	Specifies an intermediate reply. An informational message to alert the caller that the request is being processed and may not complete for some time. An example of this is the intermediate reply that is sent when a put request has been forwarded to an MSP or LS for processing and that the completion reply is deferred until that operation is complete.
DmuFinal	Specifies the final reply for the request.

This definition is used to specify the types of replies that some of the reply processing subroutines defined below are to consider.

DmuSeverity_t

The DmuSeverity_t object specifies the level of message reporting.

Valid settings are defined as follows:

DmuSevDebug4	Highest level of debug reporting.
DmuSevDebug3	Second-highest level of debug reporting.
DmuSevDebug2	Third-highest level of debug reporting.
DmuSevDebug1	Lowest level of debug reporting.
DmuSevVerbose	Verbose message reporting.
DmuSevInform	Informative message reporting.
DmuSevWarn	Warning message reporting.
DmuSevFatal	Error message reporting.

DmuVolGroup_t

The DmuVolGroup_t object defines a volume group (VG) name. As an entry in a DmuVolGroups_t array, it is used to specify one of the VGs to be used for a DMF put request. For more information about VGs, see "How DMF Works" on page 20.

The public member field and function of this class is as follows:

vgname	Specifies a character pointer to string containing the name of a valid VG.

DmuVolGroups_t

The DmuVolGroups_t object defines an array of DmuVolGroup_t objects. This object is used to specify the list of VGs to which a caller would like a file to be written in a DMF put request.

The public member fields and functions of this class are as follows:

setVolGroup()	Adds a DmuVolGroup_t object to the internal DmuVolGroup_t array.
clearVolGroup()	Removes a DmuVolGroup_t object from the internal DmuVolGroup_t array.

numVolGroups()	Returns the number of DmuVolGroup_t objects in the internal DmuVolGroup_t array.
resetVolGroups()	Clears the internal DmuVolGroup_t array.
toVolGroupsImage()	Converts a DmuVolGroups_t object to a DmuStringImage_t (defined in libdmfcom.H) in the following format: vgname1 vgname2 ... The delimiter between multiple vgname values may be a space, a tab, or a comma.
fromVolGroupsImage()	Converts a string image of the following format to a DmuVolGroups_t object: vgname1 vgname2 ... The delimiter between multiple vgname values may be a space, a tab, or a comma.

User-Accessible API Subroutines for libdmfusr.so.2

This section describes the following types of user-accessible API subroutines:

- "Context Manipulation Subroutines" on page 406
- "Filesystem Information Subroutine" on page 409
- "DMF File Request Subroutines" on page 410
- "Request Completion Subroutines" on page 426

Context Manipulation Subroutines

The DmuContext_t object manipulated by the DmuCreateContext(), DmuDestroyContext(), and DmuChangedDirectory() subroutines is designed to be completely opaque to the application. The context is used on all API subroutine calls so that the API can successfully manage user request and reply processing, but its internal contents are of no interest or use to the application.

You can use multiple DmuContext_t objects within the same process if desired.

DmuCreateContext() Subroutine

The DmuCreateContext() subroutine creates an opaque context for the API to use to correctly communicate with the dmusrcmd process. This subroutine should be the first API subroutine called by a DMF user command. Not only is the context created, but the communication channel to the dmusrcmd process is initialized.

Normally, a context would be used for multiple requests and only destroyed when no more requests are to be made. Creating and destroying a context for each request is likely to be inefficient if done frequently.

The prototype is as follows:

```
extern DmuError_t
DmuCreateContext(
        const   char            *prog_name,
                DmuCreateFlags_t create_flags,
                DmuSeverity_t   severity,
                DmuErrHandler_f err_handler,
                DmuContext_t    *dmuctxt,
                pid_t           *child_pid,
                DmuAllErrors_t  *errs)
```

The parameters are as follows:

prog_name
: Contains the name of the program. This field can be the full pathname of the program or some other representation.

create_flags
: Specifies a DmuCreateFlags_t object (defined in libdmfusr.H) that specifies create options. The only valid create_flags option is:

  CREATE_CHDIR
  : Allows change-directory requests via the DmuChangedDirectory() routine. See "DmuChangedDirectory() Subroutine" on page 408.

severity
: Specifies a DmuSeverity_t object that specifies the level of error reporting. See "DmuSeverity_t" on page 405.

err_handler	Specifies a user-defined error handling subroutine. The DmuErrHandler_f object is defined in libdmfusr.H. If the err_handler parameter is NULL, the default error handler DmuDefErrHandler is used. For more information, see "DmuErrHandler_f" on page 399.
dmuctxt	Specifies a DmuContext_t object (defined in libdmfusr.H) that is returned with the address of the newly created API to be used on all subsequent subroutine calls that require the program's API context.
child_pid	Specifies the process ID (PID) of the child that is forked and executed to create the dmusrcmd process. This value is returned to the caller so that the caller is free to handle the termination of child signals as desired.
errs	Specifies a pointer to a DmuAllErrors_t object. This value may be NULL. If it is not NULL, the subroutine will use it to return errors. See "DmuAllErrors_t" on page 390 .

If the DmuCreateContext call completes successfully, it returns DmuNoError.

DmuChangedDirectory() Subroutine

The DmuChangedDirectory subroutine changes the current directory of the context. This subroutine is useful to a process that will be making multiple API file requests using relative pathnames while the process might also be making chdir(3) subroutine calls.

When a process makes a chdir call, if the DmuChangedDirectory() subroutine is called before the next API file request that references a relative pathname is made, the file reference will be successfully made by the process.

The prototype is as follows:

```
extern DmuError_t
DmuChangedDirectory(
        const   DmuContext_t    dmuctxt,
        const   char            *new_directory,
        DmuAllErrors_t  *errs);
```

dmuctxt	Specifies a DmuContext_t object that was previously created by DmuCreateContext().

new_directory	Specifies a read-only character pointer to the string containing the directory path that was passed on the last chdir(3) subroutine call.
errs	Specifies a pointer to a DmuAllErrors_t object. This value may be NULL. If it is not NULL, the subroutine will use it to return errors. See "DmuAllErrors_t" on page 390.

DmuDestroyContext() Subroutine

The DmuDestroyContext() subroutine destroys the API context dmuctxt. The memory that had been allocated for its use is freed.

The prototype is as follows:

```
extern DmuError_t
DmuDestroyContext(
                DmuContext_t    dmuctxt,
                DmuAllErrors_t  *errs)
```

The parameters are as follows:

dmuctxt	Specifies a DmuContext_t object that was previously created by DmuCreateContext().
errs	Specifies a pointer to a DmuAllErrors_t object. This value may be NULL. If it is not NULL, the subroutine will use it to return errors. See "DmuAllErrors_t" on page 390.

Filesystem Information Subroutine

The DmuFilesysInfo() routine returns DMF configuration information about a filesystem. The dmarchive(1) command uses this routine to determine whether it can issue an archive or copy request to the DMF daemon when copying data between a source and target.

The DmuFilesysInfo() subroutine does not return until the request has either completed successfully or been aborted due to an error condition.

Upon success, a DmuFsysInfo_t object is transferred to the caller.

The prototype is as follows:

```
DmuError_t
DmuFilesysInfo(
        const   DmuContext_t    dmuctxt,
        const   char            *dmf_path,
        const   char            *fsys_path,
                DmuFsysInfo_t   *fsys_info,
                DmuAllErrors_t  *errs)
```

The parameters are as follows:

`dmuctxt`	Specifies a `DmuContext_t` object that was previously created by `DmuCreateContext()`.
`dmf_path`	Specifies a path on a DMF-managed filesystem, used only for the purposes of locating the DMF server.
`fsys_path`	Specifies a path on the filesystem for which you want configuration information. It does not need to be on a DMF-managed filesystem, nor does it need to be a mount point.
`fsys_info`	Specifies the pointer that will be returned with the `DmuFsysInfo_t` object.
`errs`	Specifies a pointer to a `DmuAllErrors_t` object. This value may be `NULL`. If it is not `NULL`, the subroutine will use it to return errors. See "`DmuAllErrors_t`" on page 390.

If the routine succeeds, it returns `DmuNoError`.

DMF File Request Subroutines

Each of the following subroutines makes a DMF file request:

- "`copy` File Requests" on page 412
- "`archive` File Requests" on page 414
- "`fullstat` Requests" on page 415
- "`put` File Requests" on page 418

- "`get` File Requests" on page 421
- "`settag` File Requests" on page 423

The context parameter that is included in each of these subroutines must have been already initialized via `DmuCreateContext`.

copy File Requests

The DmuCopyAsync() and DmuCopySync() subroutines perform copy requests in the manner of the dmcopy(1) command. The dmarchive(1) command also issues copy requests when copying from files that are in a migrated state in a DMF-managed filesystem.

The DmuCopyAsync() subroutine returns immediately after the copy request has been forwarded to the dmusrcmd process. If a reply is desired, the caller must process the reply to this request. See "Request Completion Subroutines" on page 426.

The DmuCopySync() subroutine does not return until the requested copy has either completed successfully or been aborted due to an error condition.

The prototypes are as follows:

```
extern DmuError_t
DmuCopyAsync(
        const DmuContext_t    dmuctxt,
        const char            *srcfile_path,
        const char            *dstfile_path,
              DmuCopyFlags_t  copy_flags,
        const DmuCopyRanges_t *copyranges,
              DmuPriority_t   priority,
              DmuReqid_t      *request_id,
              DmuAllErrors_t  *errs)

extern DmuError_t
DmuCopySync(
        const DmuContext_t    dmuctxt,
        const char            *srcfile_path,
        const char            *dstfile_path,
              DmuCopyFlags_t  copy_flags,
        const DmuCopyRanges_t *copyranges,
              DmuPriority_t   priority,
              DmuAllErrors_t  *errs)
```

The parameters are as follows:

dmuctxt	Specifies a DmuContext_t object that was previously created by DmuCreateContext().

`srcfile_path`	Specifies the pathname of the source (input) file for the `copy` operation. It must be an offline or dual-state DMF file.
`dstfile_path`	Specifies the pathname of the destination (output) file for the `copy` operation. This path must point to a file that exists or can be created on a filesystem visible from the DMF server and any parallel data mover nodes. See also "DMF Direct Archiving Requirements" on page 19.
`copy_flags`	Specifies the OR'd value of the following `copy` operation flags as defined in `libdmfcom.H`: • `COPY_NONE` – No flags specified • `COPY_PRESV_DFILE` – Do not truncate the destination file before the `copy` operation • `COPY_ADDR_ALIGN` – Allow an address in the destination file that is greater than the size of the file • `COPY_NOWAIT` – If the daemon is not available to process the request, do not wait (return immediately)
`copyranges`	Specifies a pointer to a `DmuCopyRanges_t` object, as defined in "`DmuCopyRanges_t`" on page 398 and in `libdmfcom.H`. This object can have only one `DmuCopyRange_t` as defined in "`DmuCopyRange_t`" on page 397 and in `libdmfcom.H`.
`priority`	Specifies a `DmuPriority_t` object (defined in `libdmfcom.H`) that defines the request priority. (Deferred implementation.)
`request_id`	Specifies a pointer to a `DmuReqid_t` object (defined in `libdmfcom.H`) parameter that will be returned with the unique request ID of the asynchronous request. This value can be used when processing `DmuCompletion_t` objects (see "Request Completion Subroutines" on page 426).
`errs`	Specifies a pointer to a `DmuAllErrors_t` object. This value may be `NULL`. If it is not `NULL`, the subroutine

will use it to return errors. See "DmuAllErrors_t" on page 390.

If the subroutine succeeds, it returns DmuNoError.

archive File Requests

The DmuArchiveAsync() and DmuArchiveSync() subroutines perform archive requests in the manner of the dmarchive(1) command, when dmarchive is operating in the mode of copying files from an unmanaged filesystem to a DMF-managed filesystem.

The DmuArchiveAsync() subroutine returns immediately after the archive request has been forwarded to the dmusrcmd process. If a reply is desired, the caller must process the reply to this request. See "Request Completion Subroutines" on page 426.

The DmuArchiveSync() subroutine does not return until the requested archive has either completed successfully or been aborted due to an error condition.

The prototypes are as follows:

```
extern DmuError_t
DmuArchiveAsync(
        const   DmuContext_t    dmuctxt,
        const   char            *src_path,
        const   char            *dst_path,
        const   DmuVolGroups_t  *volgroups,
                int             arch_flags,
                DmuPriority_t   priority,
                DmuReqid_t      *request_id,
                DmuAllErrors_t  *errs);

extern DmuError_t
DmuArchiveSync(
        const   DmuContext_t    dmuctxt,
        const   char            *src_path,
        const   char            *dst_path,
        const   DmuVolGroups_t  *volgroups,
                int             arch_flags,
                DmuPriority_t   priority,
                DmuAllErrors_t  *errs);
```

The parameters are as follows:

`dmuctxt`	Specifies a `DmuContext_t` object that was previously created by `DmuCreateContext()`.
`src_path`	Specifies the pathname of the source (input) file for the `archive` operation. It must be a file in a non-DMF managed filesystem. See "DMF Direct Archiving Requirements" on page 19.
`dst_path`	Specifies the pathname of the destination (output) file for the `archive` operation. This path must refer to a file on a DMF-managed filesystem which either does not currently exist or exists and is zero-length.
`arch_flags`	Specifies the OR'd value of the following `archive` operation flags as defined in `libdmfcom.H`: • `ARCH_NONE` – No flags specified • `ARCH_NOWAIT` – If the daemon is not available to process the request, do not wait (return immediately)
`priority`	Specifies a `DmuPriority_t` object (defined in `libdmfcom.H`) that defines the request priority. (Deferred implementation.)
`request_id`	Specifies a pointer to a `DmuReqid_t` object (defined in `libdmfcom.H`) parameter that will be returned with the unique request ID of the asynchronous request. This value can be used when processing `DmuCompletion_t` objects (see "Request Completion Subroutines" on page 426).
`errs`	Specifies a pointer to a `DmuAllErrors_t` object. This value may be NULL. If it is not NULL, the subroutine will use it to return errors. See "`DmuAllErrors_t`" on page 390.

If the subroutine succeeds, it returns `DmuNoError`.

`fullstat` Requests

The following subroutines send a `fullstat` request to the `dmusrcmd` process:

```
DmuFullstatByFhandleAsync()
DmuFullstatByFhandleSync()
```

DmuFullstatByPathAsync()
DmuFullstatByPathSync()

These subroutines have the following things in common:

- The 'Sync' versions of these subroutines do not return until the DmuFullstat_t has been received or the request has been aborted due to errors.
- The 'Async' versions of these subroutines return immediately after successfully forwarding the fullstat request to the dmusrcmd process. If a reply is desired, the caller must process the reply to this request. See "Request Completion Subroutines" on page 426. That is the only way to actually receive the DmuFullstat_t object for an 'Async' fullstat request, however. The DmuFullstatCompletion() subroutine has been supplied to extract the fullstat information from a fullstat completion object.
- The 'ByPath' versions of these subroutines allow the target file to be defined by its pathname.
- The 'ByFhandle' versions of these subroutines allow the target file to be defined by its filesystem handle, the fhandle. These subroutines are valid only when the command making the call is on the DMF server machine, and they are valid only when a user has sufficient (root) privileges.

These subroutines can return a successful completion (DmuNoError), but might not return valid DmuFullstat_t information. The subroutines are designed to return the normal stat type information regardless of whether a DMAPI fullstat could be successfully completed. Upon return from these subroutines, the caller can use the DmuFullstat_t is_valid() member function to verify the validity of the DMAPI information in the DmuFullstat_t block.

The ultimate result of this request is the transfer of a DmuFullstat_t object to the caller.

The prototypes are as follows:

```
extern DmuError_t
DmuFullstatByFhandleAsync(
                const    DmuContext_t   dmuctxt,
                const    DmuFhandle_t   *client_fhandle,
                         DmuReqid_t     *request_id,
                         DmuAllErrors_t *errs)

extern DmuError_t
DmuFullstatByFhandleSync(
                const    DmuContext_t   dmuctxt,
                const    DmuFhandle_t   *client_fhandle,
                         DmuFullstat_t  *dmufullstat,
                         DmuAllErrors_t *errs)

extern DmuError_t
DmuFullstatByPathAsync(
                const    DmuContext_t   dmuctxt,
                const    char           *path,
                         DmuReqid_t     *request_id,
                         DmuAllErrors_t *errs)

extern DmuError_t
DmuFullstatByPathSync(
                const    DmuContext_t   dmuctxt,
                const    char           *path,
                         DmuFullstat_t  *dmufullstat,
                         DmuFhandle_t   *fhandle,
                         DmuAllErrors_t *errs)
```

The parameters are as follows:

`dmuctxt`	Specifies a `DmuContext_t` object that was previously created by `DmuCreateContext()`.
`client_fhandle`	Specifies the DMF filesystem `fhandle` of the target file. Valid for use only by a privileged (`root`) user on the DMF server machine.
`path`	Specifies the relative or absolute pathname of the target file.

dmufullstat	Specifies the pointer that will be returned with the DmuFullstat_t object.
fhandle	Specifies the pointer that will be returned with the DmuFhandle_t value.
request_id	Specifies a pointer to a DmuReqid_t object (defined in libdmfcom.H) parameter that will be returned with the unique request ID of the asynchronous request. This value can be used when processing DmuCompletion_t objects (see "Request Completion Subroutines" on page 426).
errs	Specifies a pointer to a DmuAllErrors_t object. This value may be NULL. If it is not NULL, the subroutine will use it to return errors. See "DmuAllErrors_t" on page 390.

If the subroutine succeeds, it returns DmuNoError.

put File Requests

The following subroutines perform the put DMF request:

```
DmuPutByFhandleAsync()
DmuPutByFhandleSync()
DmuPutByPathAsync()
DmuPutByPathSync()
```

These subroutines have the following things in common:

- The 'Sync' versions do not return until the put request has either completed successfully, or been aborted due to errors.
- The 'Async' versions return immediately after successfully forwarding the put request to the dmusrcmd process. If a reply is desired, the caller must process the reply to this request. See "Request Completion Subroutines" on page 426.
- The 'ByPath' versions allow the target file to be defined by its pathname.
- The 'ByFhandle' versions allow the target file to be defined by its filesystem handle, the fhandle. These subroutines are valid only when the command making the call is on the DMF server machine, and they are valid only when a user has sufficient (root) privileges.

The prototypes are as follows:

```
extern DmuError_t
DmuPutByFhandleAsync(
               const   DmuContext_t    dmuctxt,
               const   DmuFhandle_t    *client_fhandle,
                       DmuMigFlags_t   mig_flags,
               const   DmuByteRanges_t *byteranges,
               const   DmuVolGroups_t  *volgroups,
                       DmuPriority_t   priority,
                       DmuReqid_t      *request_id,
                       DmuAllErrors_t  *errs)

extern DmuError_t
DmuPutByFhandleSync(
               const   DmuContext_t    dmuctxt,
               const   DmuFhandle_t    *client_fhandle,
                       DmuMigFlags_t   mig_flags,
               const   DmuByteRanges_t *byteranges,
               const   DmuVolGroups_t  *volgroups,
                       DmuPriority_t   priority,
                       DmuAllErrors_t  *errs)

extern DmuError_t
DmuPutByPathAsync(
               const   DmuContext_t    dmuctxt,
               const   char            *path,
                       DmuMigFlags_t   mig_flags,
               const   DmuByteRanges_t *byteranges,
               const   DmuVolGroups_t  *volgroups,
                       DmuPriority_t   priority,
                       DmuReqid_t      *request_id,
                       DmuAllErrors_t  *errs)

extern DmuError_t
DmuPutByPathSync(
               const   DmuContext_t    dmuctxt,
               const   char            *path,
                       DmuMigFlags_t   mig_flags,
               const   DmuByteRanges_t *byteranges,
               const   DmuVolGroups_t  *volgroups,
```

```
                    DmuPriority_t   priority,
                    DmuAllErrors_t  *errs)
```

The parameters are as follows:

dmuctxt	Specifies a DmuContext_t object that was previously created by DmuCreateContext().
client_fhandle	Specifies the DMF filesystem fhandle of the target file. Valid for use only by a privileged (root) user on the DMF server machine.
path	Specifies the relative or full pathname of the target file.
mig_flags	Specifies the following migration flags as defined in libdmfcom.H: • MIG_NONE – No flags specified • MIG_FREE – Free the space associated with the file. • MIG_NOWAIT – If the daemon is not available to process the request, do not wait (return immediately)
byteranges	Specifies a pointer to a DmuByteRanges_t object. See "DmuByteRanges_t" on page 393.
volgroups	Specifies a pointer to a DmuVolGroups_t object. See "DmuVolGroups_t" on page 405.
priority	Specifies a DmuPriority_t object (defined in libdmfcom.H) that defines the request priority. (Deferred implementation.)
request_id	Specifies a pointer to a DmuReqid_t object (defined in libdmfcom.H) parameter that will be returned with the unique request ID of the asynchronous request. This value can be used when processing DmuCompletion_t objects (see "Request Completion Subroutines" on page 426).
errs	Specifies a pointer to a DmuAllErrors_t object. This value may be NULL. If it is not NULL, the subroutine will use it to return errors. See "DmuAllErrors_t" on page 390.

If the subroutine succeeds, it returns DmuNoError.

get File Requests

The following subroutines perform the get DMF request:

```
DmuGetByFhandleAsync()
DmuGetByFhandleSync()
DmuGetByPathAsync()
DmuGetByPathSync()
```

These subroutines have the following things in common:

- The 'Sync' versions do not return until the get request has either completed successfully or has been aborted due to errors.
- The 'Async' versions return immediately after successfully forwarding the get request to the dmusrcmd process. If a reply is desired, the caller must process the reply to this request. See "Request Completion Subroutines" on page 426.
- The 'ByPath' versions of these calls allow the target file to be defined by its pathname.
- The 'ByFhandle' versions allow the target file to be defined by its filesystem handle, the fhandle. These subroutines are valid only when the command making the call is on the DMF server machine, and they are valid only when a user has sufficient (root) privileges.

The prototypes are as follows:

```
extern DmuError_t
DmuGetByFhandleAsync(
               const   DmuContext_t     dmuctxt,
               const   DmuFhandle_t     *client_fhandle,
                       DmuRecallFlags_t recall_flags,
               const   DmuByteRanges_t  *byteranges,
                       DmuPriority_t    priority,
                       DmuReqid_t       *request_id,
                       DmuAllErrors_t   *errs)

extern DmuError_t
DmuGetByFhandleSync(
               const   DmuContext_t     dmuctxt,
               const   DmuFhandle_t     *client_fhandle,
                       DmuRecallFlags_t recall_flags,
               const   DmuByteRanges_t  *byteranges,
```

```
                        DmuPriority_t    priority,
                        DmuAllErrors_t   *errs)

extern DmuError_t
DmuGetByPathAsync(
                const   DmuContext_t     dmuctxt,
                const   char             *path,
                        DmuRecallFlags_t recall_flags,
                const   DmuByteRanges_t  *byteranges,
                        DmuPriority_t    priority,
                        DmuReqid_t       *request_id,
                        DmuAllErrors_t   *errs)

extern DmuError_t
DmuGetByPathSync(
                const   DmuContext_t     dmuctxt,
                const   char             *path,
                        DmuRecallFlags_t recall_flags,
                const   DmuByteRanges_t  *byteranges,
                        DmuPriority_t    priority,
                        DmuAllErrors_t   *errs)
```

The parameters are as follows:

`dmuctxt`	Specifies a `DmuContext_t` object that was previously created by `DmuCreateContext()`.
`client_fhandle`	Specifies the DMF filesystem `fhandle` of the target file. Valid for use only by a privileged (`root`) user on the DMF server machine.
`path`	Specifies the relative or full pathname of the target file.

`recall_flags`	Specifies the following recall flags as defined in `libdmfcom.H`: • `RECALL_ATIME` - Update the access time of the file. This parameter is only valid with `DmuGetByPathAsync()` and `DmuGetByPathSync()`. • `RECALL_NONE` – No flags specified • `RECALL_NOWAIT` – If the daemon is not available to process the request, do not wait (return immediately).
`byteranges`	Specifies a pointer to a `DmuByteRanges_t` object. See "`DmuByteRanges_t`" on page 393.
`priority`	Specifies a `DmuPriority_t` object (defined in `libdmfcom.H`) that defines the request priority. (Deferred implementation.)
`request_id`	Specifies a pointer to a `DmuReqid_t` (defined in `libdmfcom.H`) parameter that will be returned with the unique request ID of the asynchronous request. This value can be used when processing `DmuCompletion_t` objects (see "Request Completion Subroutines" on page 426).
`errs`	Specifies a pointer to a `DmuAllErrors_t` object. This value may be `NULL`. If it is not `NULL`, the subroutine will use it to return errors. See "`DmuAllErrors_t`" on page 390.

If the subroutine succeeds, it returns `DmuNoError`.

`settag` File Requests

The `settag` request performs the same functional task as the `dmtag`(1)command. The following subroutines perform the `settag` DMF request:

```
DmuSettagByFhandleAsync()
DmuSettagByFhandleSync()
DmuSettagByPathAsync()
DmuSettagByPathSync()
```

These subroutines have the following things in common:

- The 'Sync' versions do not return until the settag request has either completed successfully or has been aborted due to errors.
- The 'Async' versions return immediately after successfully forwarding the settag request to the dmusrcmd process. If a reply is desired, the caller must process the reply to this request. See "Request Completion Subroutines" on page 426.
- The 'ByPath' versions allow the target file to be defined by its pathname.
- The 'ByFhandle' versions allow the target file to be defined by its filesystem handle, the fhandle. These subroutines are valid only when the command making the call is on the DMF server machine and when a user has sufficient (root) privileges.

The prototypes are as follows:

```
extern DmuError_t
DmuSettagByFhandleAsync(
               const   DmuContext_t     dmuctxt,
               const   DmuFhandle_t     *client_fhandle,
                       DmuSettagFlags_t settag_flags,
                       DmuSitetag_t     sitetag,
                       DmuPriority_t    priority,
                       DmuReqid_t       *request_id,
                       DmuAllErrors_t   *errs)

extern DmuError_t
DmuSettagByFhandleSync(
               const   DmuContext_t     dmuctxt,
               const   DmuFhandle_t     *client_fhandle,
                       DmuSettagFlags_t settag_flags,
                       DmuSitetag_t     sitetag,
                       DmuPriority_t    priority,
                       DmuAllErrors_t   *errs)

extern DmuError_t
DmuSettagByPathAsync(
               const   DmuContext_t     dmuctxt,
               const   char             *path,
                       DmuSettagFlags_t settag_flags,
                       DmuSitetag_t     sitetag,
```

```
                        DmuPriority_t    priority,
                        DmuReqid_t       *request_id,
                        DmuAllErrors_t   *errs)

extern DmuError_t
DmuSettagByPathSync(
               const    DmuContext_t     dmuctxt,
               const    char             *path,
                        DmuSettagFlags_t settag_flags,
                        DmuSitetag_t     sitetag,
                        DmuPriority_t    priority,
                        DmuAllErrors_t   *errs)
```

The parameters are as follows:

`dmuctxt` — Specifies a `DmuContext_t` object that was previously created by `DmuCreateContext()`.

`client_fhandle` — Specifies the DMF filesystem `fhandle` of the target file. Valid for use only by a privileged (`root`) user on the DMF server machine.

`path` — Specifies the relative or full pathname of the target file.

`settag_flags` — Specifies the following settag flags as defined in `libdmfcom.H`:

- `SETTAG_NONE` – No flags specified
- `SETTAG_NOWAIT` – If the daemon is not available to process the request, do not wait (return immediately)

`sitetag` — Defines the file site tag value. See `dmtag`(1).

`priority` — Specifies a `DmuPriority_t` object (defined in `libdmfcom.H`) that defines the request priority. (Deferred implementation.)

`request_id` — Specifies a pointer to a `DmuReqid_t` (defined in `libdmfcom.H`) parameter that will be returned with the unique request ID of the asynchronous request. This value can be used when processing `DmuCompletion_t` objects (see "Request Completion Subroutines" on page 426).

errs	Specifies a pointer to a DmuAllErrors_t object. This value may be NULL. If it is not NULL, the subroutine will use it to return errors. See "DmuAllErrors_t" on page 390.

If the subroutine succeeds, it returns DmuNoError.

Request Completion Subroutines

The request completion subroutines are provided so that the application can process the completion events of any asynchronous requests it might have issued. The caller can choose to process each request's completion object (DmuCompletion_t) or to be notified when each request has responded with either an intermediate or final (completion) reply.

The asynchronous requests described previously along with the following completion subroutines allow the user to achieve maximum parallelization of the processing of all requests.

DmuAwaitReplies() Subroutine

The DmuAwaitReplies() subroutine performs a synchronous wait until the number of outstanding request replies of the type specified is less than or equal to max_outstanding. This subroutine is called by a user who does not want to perform individual processing of each outstanding request, but wants to know when a reply (intermediate or final) has been received for each request that has been sent to this point.

The prototype is as follows:

```
extern DmuError_t
DmuAwaitReplies(
        const   DmuContext_t    dmuctxt,
                DmuReplyType_t  type,
                int             max_outstanding,
                DmuAllErrors_t  *errs)
```

The parameters are as follows:

dmuctxt	Specifies a DmuContext_t object that was previously created by DmuCreateContext().

`type`	Defines the type of reply to be received. The caller can wait for an intermediate or final reply for the outstanding requests. See the definition of `DmuReplyType_t` in "`DmuReplyType_t`" on page 404 or in `libdmfcom.H`.
`max_outstanding`	Specifies the number of outstanding requests allowed for which the `type` reply has not been received before the subroutine returns. If this parameter is `0`, all `type` replies will have been received when the subroutine returns.
`errs`	Specifies a pointer to a `DmuAllErrors_t` object. This value may be `NULL`. If it is not `NULL`, the subroutine will use it to return errors. See "`DmuAllErrors_t`" on page 390.

If no errors occurred getting the next reply, this subroutine returns `DmuNoError`.

`DmuFullstatCompletion()` Subroutine

The `DmuFullstatCompletion()` subroutine can be called when asynchronous `fullstat` replies are being processed by `DmuGetNextReply()` or `DmuGetThisReply()`. When the reply is received, the `DmuCompletion_t` object that is part of the reply can be used as an input parameter to this routine, which will then extract the `DmuFullstat_t` object and the `DmuFhandle_t` objects that are contained in the `DmuCompletion_t` object's `ureq_data` field.

The prototype is as follows:

```
extern DmuError_t
DmuFullstatCompletion(
                DmuCompletion_t  *comp;
                DmuFullstat_t    *dmufullstat,
                DmuFhandle_t     *fhandle,
                DmuAllErrors_t   *errs)
```

The parameters are as follows:

`comp`	Specifies the `DmuCompletion_t` object from an asynchronous `fullstat` request.
`dmufullstat`	Specifies a pointer to an existing `DmuFullstat_t` object. If `comp` references a successful `fullstat`

request, dmufullstat will be set to be equal to the DmuFullstat_t that was returned with the reply.

fhandle — Specifies a pointer to an existing DmuFhandle_t object. If comp references a successful fullstat request, fhandle will be set to be equal to the DmuFhandle_t that was returned with the reply.

errs — Specifies a pointer to a DmuAllErrors_t object. This value may be NULL. If it is not NULL, the subroutine will use it to return errors. See "DmuAllErrors_t" on page 390.

DmuGetNextReply() Subroutine

The DmuGetNextReply() subroutine returns the completion object of the next reply based on the order specified on the call.

The caller can specify DmuIntermed or DmuFinal for the type parameter. If DmuIntermed is specified and an intermediate reply is the next reply received and there are no completed replies available for processing, the comp parameter is not set (will be NULL) when the subroutine returns. An intermediate reply has no completion object associated with it; a return of this type is informational only.

This subroutine performs a synchronous wait until a request reply of the type specified on the call is received. At the time of the call, any reply that has already been received and is queued for processing is returned immediately.

The prototype is as follows:

```
extern DmuError_t
DmuGetNextReply(
        const   DmuContext_t    dmuctxt,
                DmuReplyOrder_t order,
                DmuReplyType_t  type,
                DmuCompletion_t *comp,
                DmuAllErrors_t  *errs)
```

The parameters are as follows:

dmuctxt — Specifies a DmuContext_t object that was previously created by DmuCreateContext().

order — Defines the order in which the request replies should be returned. The caller can process the replies in the order

the replies are received (DmuAnyOrder) or in the order the requests were issued (DmuReqOrder).

See the definition of DmuReplyOrder_t in "DmuReplyOrder_t" on page 404 or in libdmfcom.H.

type — Defines the type of reply to be received. The caller can wait for an intermediate or final reply for the outstanding requests. The receipt of an intermediate reply returns no data.

comp — Specifies a pointer to an existing DmuCompletion_t object. If a reply was available for processing according to the parameters on the calling subroutine, the DmuCompletion_t object pointed to by comp will be set with all of the appropriate values. See "DmuCompletion_t" on page 397.

If the reply_code field of the comp parameter is not DmuNoError, the comp->allerrors object will contain the error information needed to determine the cause of the error.

Note: The errs parameter on the subroutine call does not contain the error information for the failed request.

---

errs — Specifies a pointer to a DmuAllErrors_t object. This value may be NULL. If it is not NULL, the subroutine will use it to return errors. See "DmuAllErrors_t" on page 390.

Note: This object will return errors that occurred while waiting for or receiving this reply. It does not refer to the errors that might have occurred during the request processing that resulted in the reply Those errors are available in the comp object.

If no errors occurred getting the next reply, this subroutine returns DmuNoError. If there are no outstanding requests pending, a return code of DME_DMU_QUEUEEMPTY is returned. You can use a check for DME_DMU_QUEUEEMPTY to terminate a while

loop based on this subroutine. Any other error return code indicates an error, and the errs parameter can be processed for the error information.

DmuGetThisReply() Subroutine

The DmuGetThisReply() subroutine returns the completion object of the specified request. This subroutine performs a synchronous wait until a request reply specified on the call is received.

The prototype is as follows:

```
extern DmuError_t
DmuGetThisReply(
        const   DmuContext_t    dmuctxt,
                DmuReqid_t      request_id,
                DmuCompletion_t *comp,
                DmuAllErrors_t  *errs)
```

The parameters are as follows:

dmuctxt — Specifies a DmuContext_t object that was previously created by DmuCreateContext().

request_id — Specifies the unique request ID of the request for which the caller wants to wait.

comp — Specifies a pointer to an existing DmuCompletion_t object. If a reply was available for processing according to the parameters on the calling subroutine, the DmuCompletion_t object pointed to by comp will be set with all of the appropriate values. See "DmuCompletion_t" on page 397.

The reply_code field of the comp parameter is the ultimate status of the request. A successful comp has a reply_code of DmuNoError. If the reply_code of comp is not DmNoError, the comp->allerrors object will contain the error information needed to determine the cause of the error.

Note: The errs parameter on the subroutine call does not contain the error information for the failed request.

`errs` Specifies a pointer to a `DmuAllErrors_t` object. This value may be `NULL`. If it is not `NULL`, the subroutine will use it to return errors. See "`DmuAllErrors_t`" on page 390 .

Note: This object will return errors that occurred while waiting for or receiving this reply. It does not refer to the errors that might have occurred during the request processing that resulted in the reply Those errors are available in the `comp` object.

If no errors occurred getting the next reply, this subroutine returns `DmuNoError`. Any other error return code indicates an error, and the `errs` parameter can be processed for the error information.

Site-Defined Policy Subroutines and the `sitelib.so` Library

This appendix provides an overview of the site-defined policy feature and a summary of the policy subroutines sites may write:

Overview of Site-Defined Policy Subroutines

Site-defined policy subroutines are loaded dynamically by DMF to provide custom decision-making at key points in its processing. Several DMF processes, including `dmfdaemon`, can call subroutines within `sitelib.so`.

You do not need to use this feature, in which case DMF will function as documented in the manuals and man pages. But if you wish, you can implement one or more of these subroutines in order to override DMF's default behavior.

If you use the site-defined policy feature, you must communicate the policy changes to your user community; otherwise, they will not be able to predict how the user commands will work. The man page for any command with a site-defined policy will state something like the following:

> If your site is using the site-defined policy feature, the default behavior may be overridden. Please check with your administrator for any behavior differences due to site-defined policies.

You should also consider adding `ERROR`, `WARN`, and `INFO` messages into the reply stream for commands you customize so that you can routinely return messages to the user that explain what was changed in their request. Doing so will allows the users to understand why the behavior was different from what they expected.

The subroutines are written in C++ according to the subroutine prototypes in /usr/include/dmf/libdmfadm.H. They are placed in a shared-object library called /usr/lib/dmf/sitelib.so.

The parameters and return values of the subroutines and the name of the sitelib.so library are fixed and cannot be altered by the site. In general, the parameters provide all of the information DMF has that is relevant to the purpose of the subroutine, which is described in the comments preceding each subroutine.

The code within the subroutines performs whatever processing the site wishes. To assist in several common operations, such as extracting information from the DMF configuration file, optional helper subroutines are provided in /usr/include/dmf/libdmfadm.H.

Getting Started with Custom Subroutines

The /usr/share/doc/dmf-*/info/sample directory contains the following files to demonstrate generating the sitelib.so library:

- Basic example:
  - sample_sitelib.C contains source code of basic sample subroutines
  - sample_sitelib.mk is the associated makefile
- Example showing how to rotate migration requests across multiple media-specific processes (MSPs):
  - sample_sitelib2.C contains source code of sample subroutines to rotate migration requests
  - sample_sitelib2.mk is the associated makefile

Note: If you use these files as a base for implementing subroutines of your own, be sure to keep them in a different directory and/or rename them to avoid any conflict when DMF is upgraded and new sample files are installed. For example, you could rename the files sitelib.c and sitelib.mk.

For example, to use the basic subroutine example `sample_sitelib.C`, do the following:

1. Copy `sample_sitelib.C` and its associated makefile `sample_sitelib.mk` from `/usr/share/doc/dmf-*/info/sample` to a directory of your own with names of your own choice.

   For example, if you wanted to work in the `/tmp/testdmf` directory:

```
$ cp /usr/share/doc/dmf-*/info/sample/sample_sitelib.C /tmp/testdmf/sitelib.C
$ cp /usr/share/doc/dmf-*/info/sample/sample_sitelib.mk  /tmp/testdmf/sitelib.mk
```

2. In the makefile, specify the stem from which the library filename and source code filename will be derived by editing the value for the `SITELIB` parameter. For example, to use a stem of `sitelib` (that is, `sitelib.so` for the library and `sitelib.c` for the source code file):

   ```
   SITELIB=sitelib
   ```

   Note: Although you can set the `SITELIB` value to something other than `sitelib` for testing purposes, when you actually want to run with DMF, it must be `sitelib`.

3. Read the comments at the start of each subroutine and alter the supplied code to suit your requirements. As supplied, each subroutine is disabled. To enable one or more subroutines, modify the `SiteFncMap` variable at the bottom of the source file (in our example, `sitelib.C`).

   Note: The name of the `SiteFncMap` variable is fixed and cannot be altered. However, you can change the names of the site-defined subroutines such as `SiteCreateContext()`.

4. Build the `sitelib.so` library by using the `make`(1) command:

   ```
   $ make -f sitelib.mk
   ```

5. Print a list of the subroutines that have been enabled and visually verify that it is what you expect:

   ```
   # make -f sitelib.mk verbose
   ```

6. Install the library on a DMF server, which requires you to be the root user:

```
$ su
# make -f sitelib.mk install
```

Note: You do not need to install sitelib.so on a machine that functions only as a DMF client.

For subroutines that affect the operation of the DMF daemon, library server, or MSP, you must wait for a minute or so for the new sitelib.so library to be noticed. You will see a message in the relevant log file when this happens.

7. Test your new library by monitoring the relevant log file with tail -f while you present test cases to DMF. You may also find it useful to have a Resource Watcher configured and running or to use dmstat.

Considerations for Writing Custom Subroutines

As you write your own custom subroutines, be aware of the following:

- The sitelib.so file must be owned by root and must not be writable by anyone else, for security reasons. If these conditions are not met, DMF will ignore sitelib.so and use the default behavior.
- The sitelib.so library should not use the stdin, stdout, or stderr files as this could cause problems for DMF, possibly endangering data. For information about sending messages to users or to log files, see "DmaSendLogFmtMessage()" on page 459 and "DmaSendUserFmtMessage()" on page 460.
- If you overwrite the sitelib.so file while it is in use (for example by copying a new version of your file over the top of the old one), DMF processes may abort or run improperly. The DMF daemon may or may not be able to restart them properly.

  To update the file, you should do one of the following:

  – Use the mv(1) command to move the new file over the top of the old one, so that any existing DMF processes will continue to use the previous version of the file, which is now unlinked pending removal. The install target in the supplied makefile is also a safe way to update the file.

  - Delete the old file with rm(1) before installing the new one using cp, mv, or make install.
  - Shut down DMF while the update takes place.

  This warning also applies to changes to the DMF configuration file.

- Site-defined policy subroutines should not call subroutines in libdmfusr.so, such as DmuSettagByPathSync(). They are free to call member functions of classes defined in libdmfcom.H, such as DmuVolGroups_t::numVolGroups().
- At times, the site-defined subroutines may be called many times in rapid succession. They should therefore be as efficient as possible, avoiding any unnecessary processing, especially of system calls.

  For example, when dmfsfree is invoked to prevent a filesystem from filling, site-defined subroutines may be called one or more times for every file in the filesystem as dmfsfree prepares its list of candidates prior to migrating and/or freeing some of them. If the functions are slow, DMF may not be able to react to the situation in time to prevent the filesystem from filling.

sitelib.so Data Types

The data types described in this section are defined in libdmfadm.H. The information in this section is provided as a general description and overall usage outline. Other data types that are referenced in this file are defined in libdmfcom.H; see Appendix B, "DMF User Library libdmfusr.so" on page 385.

Note: For the most current definitions of these types, see the libdmfadm.H file.

DmaContext_t

The DmaContext_t object stores information for DMF in order to provide continuity from one subroutine call to the next. It is an opaque object that is created when a DMF process first loads sitelib.so and it exists until that process unloads it. This context is provided as a parameter for each of the site-defined policy subroutines.

Site-defined subroutines cannot directly access the information held in the context, but they can obtain information from it by using the following subroutines:

- "DmaGetContextFlags()" on page 456
- "DmaGetProgramIdentity()" on page 457
- "DmaGetUserIdentity()" on page 458

Site-defined subroutines can also store their own information in the context and retrieve it on subsequent calls by using the following subroutines:

- "DmaSetCookie()" on page 461
- "DmaGetCookie()" on page 456

DmaFrom_t

The DmaFrom_t object specifies the type of policy statement being evaluated.

There are the following possible values:

DmaFromAgeWeight	Indicates that an AGE_WEIGHT policy statement is being evaluated.
DmaFromSpaceWeight	Indicates that a SPACE_WEIGHT policy statement is being evaluated.
DmaFromVgSelect	Indicates that a SELECT_MSP or SELECT_VG policy statement is being evaluated.

DmaIdentity_t

The DmaIdentity_t object provides information, if known, about the program calling the site-defined subroutine and the user whose request generated the call.

The public member fields and functions of this class are as follows:

`realm_type`

Specifies the environment in which the type of data that is contained in the `realm_data` field is meaningful.

The following settings are defined:

- `DMF_REALM_UNIX`, which means that the `unix_1` member of `realm_data` contains valid information
- `DMF_REALM_UNKNOWN`, which means that `realm_data` is not reliable

`realm_data`

Specifies user identity information that is specific to the environment defined by `realm_type`. Only the `unix_1` member of the union is defined for the `realm_type` of `DMF_REALM_UNIX`.

If the UID and/or GID values are `0xffffffff`, the values are not reliable.

`logical_name`

Specifies a character string containing the program name of the process. This may be an absolute or relative pathname. If the value is `unknown`, the program name was unavailable.

`product_name_and_revision`

Specifies a character string containing the product name and revision (for example, `DMF_3.1.0.0`).

`locale_1`

Specifies a character string containing the locale value. See the `locale`(1) man page.

`host`

Specifies a character string containing the host on which the `DmaIdentity_t` originated.

`pid`

Specifies the process ID where the `DmaIdentity_t` originated.

instance_id

Specifies a further refinement of the PID field. Because a process may create more than one DmaIdentity_t, this value is incremented by one for each new DmaIdentity_t.

os_type

Specifies a character string containing a description of the operating system where the DmaIdentity_t originated.

os_version

Specifies a character string containing a description of the operating system version where the DmaIdentity_t originated.

cpu_type

Specifies a character string containing a description of the CPU type where the DmaIdentity_t originated.

Note: Any of the descriptive character strings may be set to unknown if the field's true value cannot be determined.

DmaLogLevel_t

The DmaLogLevel_t object specifies the level of a message. The administrator may select a log level in the DMF configuration file; messages with a less severe level than what is specified in the configuration file will not appear in the log.

DmaRealm_t

The DmaRealm_t object specifies the realm. Only the UNIX realm is supported.

DmaRecallType_t

The DmaRecallType_t object specifies the type of kernel recall being performed.

SiteFncMap_t

The SiteFncMap_t object specifies the site subroutine map. The various DMF processes that can call subroutines in sitelib.so look for a variable named SiteFncMap, of type SiteFncMap_t, in the sitelib.so library. It then uses the addresses provided in this variable to find the site-defined subroutines. If the variable is not found, DMF will not make any calls to subroutines in sitelib.so.

Site-Defined Policy Subroutines

DMF looks for the variable named SiteFncMap, of type SiteFncMap_t, in the sitelib.so library. It then uses the addresses provided in this variable to find site-defined subroutines listed in this section. You can provide any number of these subroutines in the sitelib.so library.

SiteArchiveFile()

The SiteArchiveFile() subroutine allows sites some control over the DMF archive requests. It is invoked when a dmarchive(1) command is issued to copy data directly to secondary storage or when one of the following libdmfusr.so subroutines is called:

```
DmuArchiveAsync()
DmuArchiveSync()
```

This subroutine is not called when automated space management migrates a file.

Caution: If SiteArchiveFile() is implemented, it takes precedence over any when clause being used to control MSP or volume group (VG) selection, whether or not SiteWhen() has been implemented.

If this subroutine returns a value other than DmuNoError, the archive request will be rejected. The subroutine may not issue log messages, but it can issue messages to the user.

The prototype is as follows:

```
typedef  DmuError_t (*SiteArchiveFile_f) (
        const   DmaContext_t    dmacontext,
        const   DmuFullstat_t   *fstat,
```

```
const   char            *src_path,
const   char            *dst_path,
const   DmuFhandle_t    *dst_fhandle,
const   int             flags,
const   DmuVolGroups_t  *policy_volgrps,

const   DmuPriority_t   user_priority,
const   int             user_flags,
const   DmuVolGroups_t  *user_volgrps,

        DmuPriority_t   *operative_priority,
        int             *operative_flags,
        DmuVolGroups_t  *operative_volgrps);
```

The parameters are as follows:

dmacontext	Refers to the context established when sitelib.so was loaded.
fstat	Specifies the DmuFullstat_t information of the target file for the archive request.
src_path	Specifies the pathname of the source file for the archive request.
dst_path	Specifies the pathname of the destination file for the archive request.
dst_fhandle	Specifies the DmuFhandle_t of the destination file for the archive request.
flags	Specifies whether the SiteArchiveFile() subroutine is called for the first time (0) or is replayed (nonzero). SiteArchiveFile() can be called multiple times for the same request. For example, if dmfdaemon is not running, a dmarchive request will periodically try to establish a connection with it, and SiteArchiveFile() may be called. If flags is 0, this is the first time that SiteArchiveFile() has been called for a particular request. When a request is replayed, DMF reevaluates the parameters to SiteArchiveFile() before calling it.

`policy_volgrps`
: Specifies an input parameter that contains the VGs and MSPs that have been selected by the policy statements in the DMF configuration file.

`user_priority`
`user_flags`
`user_volgrps`
: Contains information entered by the user as a `dmarchive` parameter (where supported) or as a parameter to one of the following `libdmfusr.so` subroutines:

  `DmuArchiveAsync()`
  `DmuArchiveSync()`

`operative_priority`
`operative_flags`
`operative_volgrps`
: Contains the information that will be used when the request is made to `dmfdaemon`. These are all both input and output parameters. You can alter the `operative_flags` and `operative_volgrps` values. (Currently, `operative_priority` is ignored. For compatibility with future releases of DMF, it is recommended that you do not alter the value of this parameter.)

`SiteCreateContext()`

The `SiteCreateContext()` subroutine provides the opportunity to create a site-specific setup. It is called when `sitelib.so` is loaded. If no such setup is required, it need not be implemented. If this subroutine returns anything other than `DmuNoError`, no other subroutines in `sitelib.so`, including `SiteDestroyContext()`, will be called by the current process, unless `sitelib.so` is changed and therefore reloaded.

This subroutine may not issue messages to the user because the user details are unknown at the time it is invoked. If it is invoked by a program with a log file, such as `dmfdaemon`, it can issue log messages by calling `DmaSendLogFmtMessage()`. You can call `DmaGetContextFlags()` to determine if it can issue log messages.

The prototype is as follows:

```
typedef DmuError_t  (*SiteCreateContext_f)(
        const   DmaContext_t    dmacontext);
```

The parameter is as follows:

dmacontext	Refers to the context established when sitelib.so was loaded.

SiteDestroyContext()

The SiteDestroyContext() subroutine provides the opportunity for site-specific cleanup. It is called when sitelib.so is unloaded. If no such cleanup is required, it need not be implemented. This subroutine may not issue messages to the user because the user details are no longer valid at the time it is invoked. If it is invoked by a program with a log file, such as dmfdaemon, it can issue log messages by calling DmaSendLogFmtMessage(). You can call DmaGetContextFlags() to determine if it can issue log messages.

The prototype is as follows:

```
typedef void  (*SiteDestroyContext_f)(
        const   DmaContext_t    dmacontext);
```

The parameter is as follows:

dmacontext	Refers to the context established when sitelib.so was loaded.

SiteKernRecall()

The SiteKernRecall() subroutine allows sites some control over kernel requests to recall a file. It is invoked when DMF receives a kernel request to recall a file. For example, a read() system call for a file that is currently in OFL state would result in SiteKernRecall() being called. The dmget command or the equivalent libdmfusr.so library call would not result in a call to SiteKernRecall().

This subroutine may accept or reject the request or change its priority; no other changes are possible. If the subroutine returns a value other than DmuNoError, the request will be rejected. Changing the priority has no effect at this time.

Note: `offset` and `length` pertain to the range of the file that the user's I/O request referenced, not the byte range that `dmfdaemon` will actually recall.

The subroutine may not issue messages to the user, but it can issue messages to the DMF daemon log.

The prototype is as follows:

```
typedef  DmuError_t (*SiteKernRecall_f) (
                DmaContext_t    dmacontext,
        const   DmuFullstat_t   *fullstat,
        const   DmuFhandle_t    *fhandle,
                uint64_t        offset,
                uint64_t        length,
                DmaRecallType_t recall_type,
                DmuPriority_t   *operative_priority);
```

The parameters are as follows:

`dmacontext`	Refers to the context established when `sitelib.so` was loaded.
`fullstat`	Specifies the `DmuFullstat_t` of the file being recalled
`fhandle`	Specifies the `DmuFhandle_t` of the file being recalled
`offset`	Pertains to the range of the file that the user's I/O request referenced.
`length`	Pertains to the length of the file that the user's I/O request referenced.
`recall_type`	Specifies the type of recall.
`operative_priority`	(Deferred implementation.)

SitePutFile()

The `SitePutFile()` subroutine allows sites some control over the DMF `put` requests. It is invoked when a `dmput` command is issued or when one of the following `libdmfusr.so` subroutines is called:

`DmuPutByPathAsync()`

DmuPutByPathSync()
DmuPutByFhandleAsync()
DmuPutByFhandleSync()

This subroutine is not called when automated space management migrates a file.

Caution: If SitePutFile() is implemented, it takes precedence over any when clause being used to control MSP or volume group (VG) selection, whether or not SiteWhen() has been implemented.

If this subroutine returns a value other than DmuNoError, the put request will be rejected. The subroutine may not issue log messages, but it can issue messages to the user.

The prototype is as follows:

```
typedef  DmuError_t (*SitePutFile_f) (
        const   DmaContext_t    dmacontext,
        const   DmuFullstat_t   *fstat,
        const   char            *path,
        const   DmuFhandle_t    *fhandle,
        const   int             flags,
        const   DmuVolGroups_t  *policy_volgrps,

        const   DmuPriority_t   user_priority,
        const   int             user_flags,
        const   DmuByteRanges_t *user_byteranges,
        const   DmuVolGroups_t  *user_volgrps,

                DmuPriority_t   *operative_priority,
                int             *operative_flags,
                DmuByteRanges_t *operative_byteranges,
                DmuVolGroups_t  *operative_volgrps);
```

The parameters are as follows:

dmacontext	Refers to the context established when sitelib.so was loaded.
fstat	Specifies the DmuFullstat_t information of the target file for the put request.

`path`	Specifies the pathname of the target file for the `put` request (if known) or `NULL`.
`fhandle`	Specifies the `DmuFhandle_t` of the target file for the `put` request.
`flags`	Specifies whether the `SitePutFile()` subroutine is called for the first time (`0`) or is replayed (nonzero). `SitePutFile()` can be called multiple times for the same request. For example, if `dmfdaemon` is not running, a `dmput` request will periodically try to establish a connection with it, and `SitePutFile()` may be called. If `flags` is `0`, this is the first time that `SitePutFile()` has been called for a particular request. When a request is replayed, DMF reevaluates the parameters to `SitePutFile()` before calling it.
`policy_volgrps`	Specifies an input parameter that contains the VGs and MSPs that have been selected by the policy statements in the DMF configuration file.

`user_priority`
`user_flags`
`user_byteranges`
`user_volgrps`
	Contains information entered by the user as a `dmput` parameter (where supported) or as a parameter to one of the following `libdmfusr.so` subroutines:
	
	`DmuPutByPathAsync()`
	`DmuPutByPathSync()`
	`DmuPutByFhandleAsync()`
	`DmuPutByFhandleSync()`

`operative_priority`
`operative_flags`
`operative_byteranges`
`operative_volgrps`
	Contains the information that will be used when the request is made to `dmfdaemon`. These are all both input and output parameters. You can alter the `operative_flags`, `operative_byteranges`, and `operative_volgrps` values. (Currently,

operative_priority is ignored. For compatibility with future releases of DMF, it is recommended that you do not alter the value of this parameter.)

SiteWhen()

The SiteWhen() subroutine provides the opportunity to supply the value for the sitefn variable in when clauses in the following parameters:

AGE_WEIGHT
SPACE_WEIGHT
SELECT_MSP
SELECT_VG

This subroutine and the sitefn variable in when clauses are not supported for the SELECT_LOWER_VG parameter.

Caution: If SitePutFile() or SiteArchiveFile() is implemented, it takes precedence over any when clause being used to control MSP or VG selection, whether or not SiteWhen() has been implemented.

For example,

```
SELECT_VG       tp9840  when uid = archive or sitefn = 6
```

If this subroutine is unavailable, either because it was not implemented or because the sitelib.so library is not accessible, the expression using sitefn is evaluated as being false. Therefore, the example above would be treated as if it were the following:

```
SELECT_VG       tp9840  when uid = archive or false
```

Or:

```
SELECT_VG       tp9840  when uid = archive
```

If a policy stanza contains multiple references to sitefn, it is possible that the subroutine is only called once and the value returned by that call may be used for several substitutions of sitefn. Therefore, a policy that contains the following will not necessarily call the subroutine three times:

```
AGE_WEIGHT      -1     0        when sitefn < 10
AGE_WEIGHT      1     .1
```

```
SPACE_WEIGHT    1     1e-6      when sitefn != 11
SPACE_WEIGHT    2     1e-9      when sitefn > 19
SPACE_WEIGHT    3.14  1e-12
```

The subroutine can issue log messages in some circumstances and user messages in others. You can call `DmaGetContextFlags()` to determine what kind of messages are possible.

The prototype is as follows:

```
typedef int (*SiteWhen_f) (
        const DmaContext_t  dmacontext,
        const DmuFullstat_t *fstat,
        const DmuFhandle_t  *fhandle,
              DmaFrom_t     fromtyp);
```

The parameters are as follows:

`dmacontext`	Refers to the context established when `sitelib.so` was loaded
`fstat`	Specifies the `DmuFullstat_t` of the file being evaluated.
`fhandle`	Specifies the `DmuFhandle_t` of the file being evaluated.
`fromtyp`	Indicates what kind of policy is being evaluated.

Helper Subroutines for `sitelib.so`

This section describes optional subroutines that may be called from `sitelib.so` and are present in the processes that load `sitelib.so`.

`DmaConfigStanzaExists()`

The `DmaConfigStanzaExists()` subroutine checks whether a specified stanza exists in the DMF configuration file.

Note: Values in the configuration file may change while DMF is running.

The prototype is as follows:

```
DmaBool_t
DmaConfigStanzaExists(
        const   DmaContext_t    dmacontext,
        const   char            *type,
        const   char            *stanza);
```

The parameters are as follows:

dmacontext	Specifies the DmaContext_t parameter passed as input to all site-defined policy subroutines, such as SitePutFile().
type	Specifies the type of the stanza being checked.
stanza	Specifies the name of the stanza being checked.

For example, if the DMF configuration file contained the following:

```
define  /dmf1
        TYPE            filesystem
        POLICIES        space_policy vg_policy
enddef
```

Then the following call would return true:

```
DmaConfigStanzaExists(dmacontext, "filesystem","/dmf1");
```

DmaGetConfigBool()

The DmaGetConfigBool() subroutine extracts parameter values of type DmaBool_t from the specified stanza in the DMF configuration file. If there is no such parameter definition or if it exists but with a missing or improper value, then the default is used.

Note: Values in the configuration file may change while DMF is running.

The prototype is as follows:

```
DmaBool_t
DmaGetConfigBool(
        const   DmaContext_t    dmacontext,
        const   char            *stanza,
        const   char            *param,
                DmaBool_t       default_val);
```

The parameters are as follows:

`dmacontext`	Specifies the `DmaContext_t` parameter passed as input to all site-defined policy subroutines, such as `SitePutFile()`.
`stanza`	Specifies the name of the stanza being searched.
`param`	Specifies the name of the parameter for which `DmaGetConfigBool()` is searching.
`default_val`	Specifies the value to use if `param` is not found in `stanza` or if `param` has a missing or invalid value.

`DmaGetConfigFloat()`

The `DmaGetConfigFloat()` subroutine extracts parameter values of type `float` from the specified stanza in the DMF configuration file. If there is no such parameter definition or if it exists but with a missing or invalid value, the default is used.

Note: Values in the configuration file may change while DMF is running.

The prototype is as follows:

```
float
DmaGetConfigFloat(
        const   DmaContext_t    dmacontext,
        const   char            *stanza,
        const   char            *param,
                float           default_val,
                float           min,
                float           max);
```

The parameters are as follows:

dmacontext	Specifies the DmaContext_t parameter passed as input to all site-defined policy subroutines, such as SitePutFile().
stanza	Specifies the name of the stanza being searched.
param	Specifies the name of the parameter for which DmaGetConfigFloat() is searching.
default_val	Specifies the value to use if param is not found in stanza or if param has a missing or invalid value.
min	Defines the minimum valid value.
max	Defines the maximum valid value.

DmaGetConfigInt()

The DmaGetConfigInt() subroutine extracts parameter values of type int64_t from the specified stanza in the DMF configuration file. If there is no such parameter definition or if it exists but with a missing or invalid value, then a default value is used.

Note: Values in the configuration file may change while DMF is running.

The prototype is as follows:

```
int64_t
DmaGetConfigInt(
        const   DmaContext_t    dmacontext,
        const   char            *stanza,
        const   char            *param,
                int64_t         default_val,
                int64_t         min,
                int64_t         max);
```

The parameters are as follows:

dmacontext	Specifies the DmaContext_t parameter passed as input to all site-defined policy subroutines, such as SitePutFile().

`stanza`	Specifies the name of the stanza being searched.
`param`	Specifies the name of the parameter for which `DmaGetConfigInt()` is searching.
`default_val`	Specifies the value to use if `param` is not found in `stanza` or if `param` has a missing or invalid value.
`min`	Defines the minimum valid value.
`max`	Defines the maximum valid value.

DmaGetConfigList()

The `DmaGetConfigList()` subroutine returns a pointer to an array of words found in the parameter in the specified stanza. The `items` value points to a block of memory containing an array of string pointers are well as the strings themselves; the end of the array is marked by a `NULL` pointer. The block of memory has been allocated by the `malloc()` subroutine and can be released with the `free()` subroutine if desired. The caller is responsible for releasing this memory.

Note: Values in the configuration file may change while DMF is running.

The prototype is as follows:

```
DmaBool_t
DmaGetConfigList(
        const   DmaContext_t    dmacontext,
        const   char            *stanza,
        const   char            *param,
                char            *** items);
```

The parameters are as follows:

`dmacontext`	Specifies the `DmaContext_t` parameter passed as input to all site-defined policy subroutines, such as `SitePutFile()`.
`stanza`	Specifies the name of the stanza being searched.
`param`	The name of the parameter for which `DmaGetConfigList()` is searching.

items	Specifies an output value that points to a block of memory containing an array of string pointers as well as the strings themselves; the end of the array is marked by a NULL pointer.

DmaGetConfigStanza()

The DmaGetConfigStanza() subroutine return a pointer to an array of parameters and values for the specified stanza in the DMF configuration file. (That is, it provides the entire stanza, after comments have been removed.) The items value points to a block of memory containing an array of structures with string pointers as well as the strings themselves; the end of the array is marked by a NULL pointer. The block of memory has been allocated by the malloc() subroutine and can be released with the free() subroutine if desired. The caller is responsible for releasing this memory.

Note: Values in the configuration file may change while DMF is running.

The prototype is as follows:

```
DmaBool_t
DmaGetConfigStanza(
        const   DmaContext_t    dmacontext,
        const   char            *stanza,
                DmaConfigData_t **items);

}
```

The parameters are as follows:

dmacontext	Specifies the DmaContext_t parameter passed as input to all site-defined policy subroutines, such as SitePutFile().
stanza	Specifies the name of the stanza being searched.
items	Specifies an output value that points to a block of memory containing an array of structures with string

pointers as well as the strings themselves; the end of the array is marked by a NULL pointer.

DmaGetConfigString()

Extracts a string from the specified stanza in the DMF configuration file and returns it. If there is no such parameter definition, the default is used. If the parameter exists but with a missing value, the null string (which is a valid value) is returned.

Note: Values in the configuration file may change while DMF is running.

The prototype is as follows:

```
void
DmaGetConfigString(
        const   DmaContext_t    dmacontext,
        const   char            *stanza,
        const   char            *param,
        const   char            *default_val,
        DmuStringImage_t        &result);
```

The parameters are as follows:

dmacontext	Specifies the DmaContext_t parameter passed as input to all site-defined policy subroutines, such as SitePutFile().
stanza	Specifies the name of the stanza being searched.
param	Specifies the name of the parameter for which DmaGetConfigString() is searching.
default_val	Specifies the value to use if param is not found in stanza. If param is found in stanza but has a missing value, the null string is returned.
result	Specifies an output parameter, containing the result.

DmaGetContextFlags()

The DmaGetContextFlags() determines if a given subroutine can issue log messages or issue user messages.

Note: If DmaFlagContextValid() is not set in the return value, no use should be made of any other bits.

DmaGetContextFlags() can return the following values, which may be OR'd together:

DmaFlagContextValid	Indicates that the context is valid.
DmaFlagLogAvail	Indicates that DmaSendLogFmtMessage may be called.
DmaFlagMsgAvail	Indicates that DmaSendUserFmtMessage may be called.

The prototype is as follows:

```
uint64_t
DmaGetContextFlags(
        const   DmaContext_t    dmacontext);
```

The parameter is as follows:

dmacontext	Specifies the DmaContext_t parameter passed as input to all site-defined policy subroutines, such as SitePutFile().

DmaGetCookie()

The DmaGetCookie() subroutine returns the cookie that was stored in dmacontext by a call to DmaSetCookie(). If a NULL value is returned, either the context is invalid or the cookie was not set.

The prototype is as follows:

```
void *
DmaGetCookie(
        const   DmaContext_t    dmacontext);
```

The parameter is as follows:

`dmacontext`	Specifies the `DmaContext_t` parameter passed as input to all site-defined policy subroutines, such as `SitePutFile()`.

`DmaGetDaemonVolGroups()`

The `DmaGetDaemonVolGroups()` subroutine returns the VGs and MSPs that the `dmfdaemon` is currently configured to use.

Note: Values in the configuration file may change while DMF is running.

The prototype is as follows:

```
const DmuVolGroups_t *
DmaGetDaemonVolGroups(
        const   DmaContext_t    dmacontext);
```

The parameter is as follows:

`dmacontext`	Specifies the `DmaContext_t` parameter passed as input to all site-defined policy subroutines, such as `SitePutFile()`.

`DmaGetProgramIdentity()`

The `DmaGetProgramIdentity()` subroutine returns a pointer to the program `DmaIdentity_t` object in the `dmacontext` parameter.

Note: The program `DmaIdentity_t` object should not be confused with the user `DmaIdentity_t` object that is returned by "`DmaGetUserIdentity()`" on page 458. The user identity is usually of much more interest when applying site policies because it defines who is actually making the request as opposed to what process is negotiating the site policies.

The prototype is as follows:

```
const DmaIdentity_t *
DmaGetProgramIdentity(
        const   DmaContext_t    dmacontext);
```

The parameter is as follows:

dmacontext	Specifies the DmaContext_t parameter passed as input to all site-defined policy subroutines, such as SitePutFile().

DmaGetUserIdentity()

The DmaGetUserIdentity() subroutine returns a pointer to the user DmaIdentity_t object in the dmacontext parameter.

The user DmaIdentity_t object contains as much information as could be reliably gathered regarding the identity of the originator of the request. For example, the user identity in the SitePutFile() policy subroutine would identify the process (such as dmput) that made the original DmuPutByPathSync() libdmfusr call.

If DmaGetUserIdentity() is called from within SiteKernRecall(), it will return the identity of dmfdaemon. The identity of the user who initiated the read request that caused SiteKernRecall() to be called is unknown to DMF.

Within SiteCreateContext(), the user details may be as yet unknown; therefore, DmaGetUserIdentity() may return different values than if it is called with the same context from another site-defined policy subroutine. In most cases, the user identity is determined after the call to SiteCreateContext().

Under certain circumstances, some elements of the DmaIdentity_t structure may be unknown. For example, if a site-defined subroutine is called as a result of a command entered on a client machine running a release prior to DMF 3.1, some elements of the user identity may be unknown.

The prototype is as follows:

```
const DmaIdentity_t *
DmaGetUserIdentity(
        const   DmaContext_t    dmacontext);
```

The parameter is as follows:

dmacontext	Specifies the DmaContext_t parameter passed as input to all site-defined policy subroutines, such as SitePutFile().

DmaSendLogFmtMessage()

The DmaSendLogFmtMessage() subroutine formats and issues log messages, if log messages are possible. The messages will potentially appear in the calling program's log depending upon the DmaLogLevel_t of the message and the log level selected by the administrator in the DMF configuration file. If log messages are not possible, DmaSendLogFmtMessage() silently discards the message.

The prototype is as follows:

```
void
DmaSendLogFmtMessage(
        const   DmaContext_t    dmacontext,
                DmaLogLevel_t   log_level,
        const   char            *name,
        const   char            *format,
                ...);
```

The parameters are as follows:

dmacontext	Specifies the DmaContext_t parameter passed as input to all site-defined policy subroutines, such as SitePutFile().
log_level	Specifies the level of the message.
name	Specifies a string that is included as part of the log message.
format	Specifies the format for the message that will be printed in the log. It looks like a printf(3S) format. Do not include \n as part of the message. If you want to print

more than one line to the log, make multiple calls to DmaSendLogFmtMessage().

For example, the following will issue an error message to the calling program's log:

```
DmaSendLogFmtMessage (dmacontext, DmaLogErr,
               "SiteCreateContext", "sitelib.so problem errno %d",
               errno);
```

DmaSendUserFmtMessage()

The DmaSendUserFmtMessage() subroutine formats and sends messages to the user, if user messages are possible. The messages will potentially appear as output from commands such as dmput and dmget, depending upon the severity of the message and the level of message verbosity selected by the user. If user messages are not possible, DmaSendUserFmtMessage() silently discards the message.

The prototype is as follows:

```
void
DmaSendUserFmtMessage(
        const   DmaContext_t    dmacontext,
                DmuSeverity_t   severity,
        const   char            *position,
                int             err_no,
        const   char            *format,
                ...);
```

The parameters are as follows:

dmacontext	Specifies the DmaContext_t parameter passed as input to all site-defined policy subroutines, such as SitePutFile().
severity	Specifies the severity of the message.
position	Specifies a string that can be included in the message. This string may be set to NULL.
err_no	Specifies that if err_no is non-zero, the results of strerror(err_no) will be included in the message.

`format`	Specifies the format for the message that will be sent to the user. It looks like a `printf`(3S) format. It is not necessary to put a `\n` at the end of the message.

DmaSetCookie()

The `DmaSetCookie()` subroutine stores a pointer to site-defined subroutine information in `dmacontext`. This pointer may be retrieved by a call to `DmaGetCookie()`. The site-defined subroutines are responsible for memory management of the space pointed to by the `cookie` parameter.

The prototype is as follows:

```
void
DmaSetCookie(

        const   DmaContext_t    dmacontext,
                void            *cookie);
```

The parameters are as follows:

`dmacontext`	Specifies the `DmaContext_t` parameter that is passed as input to all site-defined policy subroutines, such as `SitePutFile()`.
`cookie`	Specifies a pointer to information that `sitelib.so` subroutines want to retain while the `dmacontext` is valid.

Third-Party Backup Package Configuration

The following third-party backup packages are known to be DMF-aware:

EMC® LEGATO NetWorker®

Note: EMC Networker only operates with Linux standard `st` tape devices. DMF and OpenVault only operate with SGI `ts` tape devices. A given tape drive can be managed as either an `st` device or as a `ts` device, not both. To learn how to use `ts` and `st` tape devices for different tape drives on the same system (where each tape drive is assigned to one device or the other), see the `/etc/ts/README.apd` file on the DMF server.

If OpenVault manages the library for DMF, Networker and OpenVault will each have their own set of tape devices but they are unaware of each other's devices. To allow each software package to access its own set of tape volumes and tape devices, you must partition the library.

To use EMC LEGATO NetWorker to back up DMF-managed filesystems, add each filesystem to the NetWorker client's save set list and enable `dmfasm` on each filesystem.

Note: Only `root` can restore migrated files because DMF uses an extended (system) attribute owned by `root`.

You can enable the `dmfasm` module by creating a file named `.nsr` in the root directory of each DMF-managed filesystem. The contents of this file should be the following, which specifies that `dmfasm` should be used on all files (including hidden files) and subdirectories:

```
+dmfasm: * .?*
```

Note: As of NetWorker 7.1.2, the `nwbackup` and `nwrecover` commands do not include `dmfasm`, and therefore backups and recovers performed with those commands will not be DMF-aware. Only the `save`, `savepnpc`, and `recover` commands use `dmfasm`.

An alternative method for enabling `dmfasm` on DMF-managed filesystems is to create a directive resource using `nwadmin`. For example, with two DMF filesystems `/dmfusr1` and `/dmfusr2`, the directive resource would contain the following:

```
<< /dmfusr1 >> +dmfasm: * .?*
<< /dmfusr2 >> +dmfasm: * .?*
```

After creating the directive, you must update the NetWorker client's `Directive` field to use the new directive.

See the NetWorker documentation for more information about ASMs, `.nsr` files, and directives.

To use DMF's `do_predump.sh` script with NetWorker, set up the NetWorker client to use a precommand as follows:

1. Set the client's `Backup command` field to `savepnpc`.
2. Create a file named `/nsr/res/`*grpname*`.res`, where *grpname* is the NetWorker group to which the client belongs. The file should contain the following:

   ```
   type: savepnpc;
   precmd: "/usr/lib/dmf/do_predump.sh daemon dump_tasks";
   ```

   where:

   - *daemon* is the name of the `dmdaemon` object in the DMF configuration file
   - *dump_tasks* is the name of the task group specifying parameters related to backups

Note: DMF's `DUMP_RETENTION` parameter should match the value of the NetWorker client's `Retention Policy` parameter.

For more information about Networker, see www.emc.com and the Networker manuals.

Atempo® Time Navigator™

Atempo Time Navigator is high-performance backup and recovery software designed with intuitive graphical user interfaces (GUIs) to manage data in heterogeneous environments.

Time Navigator is DMF-aware and supports a broad range of servers and client operating systems including SGI IRIX and 64-bit Linux running on Intel® Itanium® 2 processors. It also supports a wide range of SAN hardware and tape libraries. Time Navigator by default uses Atempo's proprietary Time Navigator protocol for all data transfers.

To make Time Navigator aware of a DMF filesystem, add a line resembling the following to the *full-Time-Navigator-installation-path*`/Conf/parameters` file, where */dmfusr* is the DMF user filesystem:

`parameter:bapi_fs=`*`/dmfusr`*

You can specify more DMF filesystems by adding a similar line for each DMF filesystem.

Using the Time Navigator GUI, you can define **backup classes** to select which directories you want to back up. You can also vary the granularity for backup and restore, such as file, directory, or class level.

To use DMF's `do_predump.sh` script with Time Navigator, set up Time Navigator to use a precommand as follows:

- In the **Advanced** settings of the backup strategy, specify the following as the preprocessing command:

  `/usr/lib/dmf/do_predump.sh` *daemon dump_tasks*

  where:

  daemon Name of the `dmdaemon` object in the DMF configuration file

  dump_tasks Name of the task group specifying the parameters related to backups

- Ensure that DMF's `DUMP_RETENTION` parameter matches the retention value of the cartridge pool associated with backing up the DMF filesystem.

For more information about Time Navigator, see www.atempo.com and the Time Navigator manuals.

Converting from IRIX DMF to Linux® DMF

Note: This procedure must take place during a planned outage of the systems and filesystems managed by DMF. It is assumed that sites converting DMF from IRIX to Linux will obtain the help of SGI customer support. The following documentation is offered to familiarize you with the necessary steps.

This appendix describes the necessary steps to convert an IRIX DMF system to a Linux DMF system and provides an example using a single library server (LS).

You cannot copy DMF databases from an IRIX system to a Linux system because of binary incompatibility. Instead, you must dump the IRIX DMF databases to text on the IRIX system and load the resulting text file into the databases on the Linux system. However, you can move DMF-managed filesystems (that is, filesystems containing user files that DMF has migrated) from an IRIX system to a Linux system.

Procedure E-1 Converting from IRIX DMF to Linux DMF

1. Discontinue all user activity for the duration of the IRIX to Linux conversion process.
2. If you have a tape MSP, you must convert it to a volume group (VG) in an LS **while still on IRIX** using `dmmsptols`.

   Note: The tape MSP is not available in the Linux DMF release.

   For more information, see the DMF 3.0 version of the *DMF Administrator's Guide for SGI InfiniteStorage* (007-3681-008).
3. Prepare the DMF databases on the IRIX system:
   a. Change the filesystem migration levels in the `dmf.conf` file to `none`.
   b. Run `dmdidle` and wait for activity to cease.
   c. Use `dmsnap` to back up the DMF databases.

      Alternatively, if time or disk space considerations are critical, it is acceptable to use the snapshot of the DMF databases that is generated in the

dmaudit_working_dir as part of step 4 below as the database backup copy, allowing you to skip this `dmsnap` step.

4. Audit the DMF databases to ensure that they are valid:

Caution: Do not proceed until you have obtained clean results for each step in turn.

   a. Run `dmaudit snapshot` and resolve all errors before moving on to step 4.b.

   b. Run `dmatvfy` *dmaudit_working_dir* and resolve all errors before moving on to step 4.c.

   c. Run `dmdskvfy` against all DCM and disk MSPs and resolve all errors before moving on to step 5.

   For more information, see the `dmaudit`(8) man page and *DMF Filesystem Audit Guide for SGI InfiniteStorage.*

5. Stop DMF on the IRIX system.

Caution: If DMF is started again on the IRIX system during or after this procedure, the databases captured during step 7 might not reflect reality, and loss of data might result if you use them.

6. Use `dmdbcheck` to verify the consistency of the DMF databases.

7. Dump all of the DMF databases to text from the snapshot taken in step 3c above. This should include the daemon database and the CAT and VOL tables for each LS database. For more information, see the `dmdump`(8) man page.

8. Sort the daemon and CAT text database records for better overall performance of the text-record load process. (The time to sort and load will be less than the time to load unsorted text records when the number of records is in the millions.) Do the following:

   - To sort the daemon text record file, use a command similar to the following, where *tmpdir* is a directory in a filesystem with sufficient free space for `sort` to complete the sort:

     ```
     # /bin/sort -t"|" -y -T tmpdir -k 1,1 -o sorted_daemontext daemontext
     ```

- To sort the CAT text record file, use a command similar to the following, where *tmpdir* is a directory in a filesystem with sufficient free space for `sort` to complete the sort:

  ```
  # /bin/sort -t"|" -y -T tmpdir -k 2,2 -o sorted_cattext cattext
  ```

For more information, see the `sort`(1) man page.

9. Set up the `/etc/dmf/dmf.conf` file on the Linux system. The conversion will be simpler if you name all of the FTP MSPs, disk MSPs, tape VGs, and LSs with the same names used on IRIX. This assumes that you do not already have MSPs, LSs, or VGs with these names on your Linux system.

   If you do change the name of an MSP or VG, you must convert the daemon database. For more information on how to perform this conversion, see the documentation in the `dmconvertdaemon` script.

10. Use `dmcheck` to ensure that your new `/etc/dmf/dmf.conf` file is valid on the Linux system.

11. Copy the text versions of the databases (which you created in step 7 and sorted in step 8) to the Linux system.

12. Load the database files from the text files on the Linux system. Use the following commands:

    Note: If you are loading the text records into an empty database, use the `-j` option on the `dmdadm`(8) and `dmcatadm`(8) commands to eliminate the unnecessary overhead of database journal records. If you are loading the records into a nonempty database, SGI recommends that you make a copy of the database before running the `dmdadm` and `dmcatadm` commands and that you do not use `-j` option.

    - `dmdadm` to load the daemon database file
    - `dmcatadm` to load the CAT records for each of the LS databases
    - `dmvoladm` to load the VOL records for each of the LS databases

13. Use `dmdbcheck` to check the consistency of databases on the Linux system.

14. Move all of the DMF-managed user filesystems and DCM filesystems from the IRIX system to the Linux system:

- If reusing the existing disks and the IRIX filesystem blocksize is supported by Linux (512, 1024, 2048, 4096, 8192, or 16384), you can simply move the disks from the IRIX system to the Linux system.
- If there is a disk resource upgrade or if the IRIX block size greater than what is supported in Linux, there will be new user filesystems built under Linux. The old data must then be restored to these new filesystems. For information, see "Using SGI xfsdump and xfsrestore with Migrated Files" on page 350.

15. Start DMF on the Linux system.
16. Run `dmaudit` to verify the filesystems.

Example E-1 IRIX to Linux Conversion (Single LS)

In the following example, the IRIX system has a single LS named `ls1`. The example assumes that the `/tmp/dmfdatabases` directory has been created, is initially empty, and contains enough space to accommodate the text versions of the databases. The example also assumes that the `HOME_DIR` configuration parameter is set to `/dmf/home` on both systems. After completing steps 2 through 6 of Procedure E-1 on page 467, the daemon database and the CAT and VOL tables of the LS database are dumped to text, as follows:

```
$ dmdump -c /dmf/home/daemon > /tmp/dmfdatabases/daemon_txt
$ dmdump /dmf/home/ls1/tpcrdm.dat > /tmp/dmfdatabases/ls1_cat_txt
$ dmdump /dmf/home/ls1/tpvrdm.dat > /tmp/dmfdatabases/ls1_vol_txt
```

Next, the files in `/tmp/dmfdatabases` on the IRIX system are copied to `/tmp/dmftxtdb` on the Linux system. After creating the DMF configuration file on the Linux system, the databases are loaded on the Linux system, as follows:

```
$ dmdadm -u -c "load /tmp/dmftxtdb/daemon_txt"
$ dmcatadm -m ls1 -u -c "load /tmp/dmftxtdb/ls1_cat_txt"
$ dmvoldadm -m ls1 -u -c "load /tmp/dmftxtdb/ls1_vol_txt"
```

Now `dmdbcheck` is run to verify the consistency of the databases, as follows:

```
$ cd /dmf/home/daemon; dmdbcheck -a dmd_db
$ cd /dmf/home/ls1; dmdbcheck -a libsrv_db
```

Considerations for Partial-State Files

This section discusses the following:

Performance Cost Due to Lack of Linux Kernel Support

The Linux kernel does not provide underlying support for partial-state files. A partial-state file looks exactly like an offline file to the filesystem, and so all read requests for a partial-state file generate a DMF daemon read event, whether the byte range being read is actually already online or not. The DMF daemon will write an attribute to a partial-state file that includes the number and boundaries of each region so that any read event whose byte range is completely contained in an online region will return immediately to the kernel with no intervening recall. A read event whose byte range is not completely contained in an online region will result in the entire file being recalled.

Because there is no underlying support in the Linux kernel, the DMF partial-state file feature has a performance cost. The kernel cannot detect when a read request could be satisfied without a read event being generated to the DMF daemon, resulting in pseudo read events that cannot be absorbed by the system and therefore impact the system's performance. A performance degradation will be noticed if thousands of pseudo read events are being generated in a short period of time.

For example, if a very large file has a very large online region followed by a very small offline region and a process is doing a sequential read through the file using a small buffer size, each of the reads for the online region will result in a pseudo read event until finally a read for the offline region will cause the rest of the file to be brought back online. A single process doing this kind of operation might not impact the system, but tens or hundreds of simultaneous similar processes may. In this situation, it might be better to manually recall the file before doing the read.

Additionally, the pseudo read events will result in DMF daemon log-file entries for each read, and so the DMF `SPOOL_DIR` directory may experience a very significant increase in the amount of disk space that is consumed each day. If this is the case, the `SPOOL_DIR` directory will require maintenance (file removal) on a more frequent basis.

Inability to Fulfill Exact Byte Range Requests

User files can become partial-state either manually or automatically. The manual method involves using the byte-range parameters on the `dmput`(1) and `dmget`(1) commands. (See the man pages for a full description of the syntax of the byte-range specifications). You can use these commands to manually control which regions of a user file should be made online or offline, subject to the restrictions of the underlying filesystem and the maximum number of regions allowed in that filesystem.

All currently supported filesystems have a restriction that punching a hole in a file (to make a region offline) must take place on a fixed boundary size, usually on a 4096–byte block boundary. If a user requested an offline region from byte 10000 to byte 20000, the resulting offline byte range would be from byte 12288 to byte 16384. Offline regions are rounded inward, which might result in fewer bytes than specified being made offline, but will never result in more bytes than specified being made offline.

When requesting online regions, the byte addresses are rounded outward. So in the 10000-20000 byte address example, the resulting online region would be from byte 8192 to byte 20480 based on the idea that it is better to bring some extra bytes online than to not bring all of the bytes that were requested online.

It is entirely possible that a `dmput` or `dmget` request that specifies a byte-range parameter will result in no action on the file taking place. This is possible if the file is already in the requested state (just like using `dmget` on a `DUALSTATE` file before the introduction of partial-state files) or if the requested state would result in more than the maximum number or regions allowed by the filesystem per file. (See the `MAX_MANAGED_REGIONS` configuration file parameter in "`filesystem` Object" on page 187.) Because of the general inability of DMF to deliver the exact byte ranges requested, requests that do not deliver exact byte range results do not return an error. It is up to the caller to determine the exact state of the file after the request.

Case Study: Impact of Zone Size on Tape Performance

This appendix details an experiment with a 100 MB/s LTO4 drive, which is in the same performance class as the STK T10000A. The purpose of the test was to show the cost of having a small zone size (the `ZONE_SIZE` parameter, see "`volumegroup` Object" on page 224).

The `moverlog.`*yyyymmdd* log traces show two tests:

- In the first test, we migrated 200 512–MB files to tape using a `ZONE_SIZE` of `10g` (10 GB). This resulted in 10 zones.
- In the second test, we recalled all the files, changed the `ZONE_SIZE` to `499m` (499 MB), and remigrated the same 200 files. In the second test, each migrated file became its own zone (200 zones).

In the first test (with a `ZONE_SIZE` of `10g`), the tape drive achieved 118–MB/s per zone. This is the drive's full streaming rate. For example, the drive spent 89.6 seconds doing I/O to the first zone and only 1.48 seconds flushing the first zone:

```
12:49:54-V   102037-dmatwc process_completed_zone: Zone 1 written, chunks=21, bytes=10752000000
12:49:54-V   102037-dmatwc stats: idle=0.00, mount=32.27, skip=0.00, io=89.60, zone=1.48
12:49:54-V   102037-dmatwc stats: total chunks=21, mb=10752.000000, rate=118.05 mb/s
```

When the first migration test was complete, the `dmatwc` final statistics showed that the drive consistently achieved 114 MB/s, and the effective rate (if you include mount/unmount/zone/close/rewind time) was 89 MB/s (line breaks shown here for readability):

```
13:06:55-I   102037-dmatwc final_stats: idle=107.66, mount=32.27, skip=0.00, io=868.54,
   zone=20.84, close=81.29, unmount=34.19
13:06:55-I   102037-dmatwc final_stats: total sec = 1144.78, totalmb=101911.101562, rate=114.59 mb/s, effective
   rate=89.02 mb/s
```

In the second test (with a `ZONE_SIZE` of `499m`), the increased stop/start behavior of the drive meant that the drive only achieved about half of its native rate, or 67.28 MB/s (line breaks shown here for readability):

```
13:19:53-V   104013-dmatwc process_completed_zone: Req=4,6dc90 done, chunk=7, zone=4,
   chunklength=512000000, bytes=512000000
13:19:53-V   104013-dmatwc process_completed_zone: Zone 4 written, chunks=1, bytes=512000000
```

```
13:19:53-V   104013-dmatwc stats: idle=0.01, mount=31.88, skip=0.00, io=23.70, zone=5.93
13:19:53-V   104013-dmatwc stats: total chunks=1, mb=512.000000, rate=67.28 mb/s
```

When the second migration test was complete, the `dmatwc` final statistics show that the drive was only able to achieve 66 MB/s when it was doing I/O. Furthermore, 304.58 seconds were spent just flushing data (versus 20 seconds in the first test). Thus the effective rate in the second case was only 56 MB/s (line breaks shown here for readability):

```
13:48:57-I   104013-dmatwc final_stats: idle=114.54, mount=31.88, skip=0.00, io=1237.52,
   zone=304.58, close=82.74, unmount=34.09
13:48:57-I   104013-dmatwc final_stats: total sec = 1805.36, totalmb=102248.742188, rate=66.30 mb/s, effective
   rate=56.64 mb/s
```

Had we done a larger test and written an entire tape in each case, the mount, unmount, and close (rewind) time would have contributed much less to the effective bandwidth, and so the numbers would be even more dramatic.

You can obtain the statistics discussed in this appendix from the following log file:

SPOOL_DIR`/`*ls_name*`/moverlogs/`*hostname*`/moverlog.`*yyyymmdd*

For more information, see:

- "Improve Tape Drive Performance with an Appropriate Zone Size" on page 70
- "LS Logs" on page 308

Historical Feature Information

This appendix contains the following:

End of Life for the Tape Autoloader API with DMF 2.6.3

With the release of DMF 2.6.3, DMF dropped support for the tape autoloader API. DMF supports OpenVault and TMF as tape mounting services. If you have not yet acquired OpenVault or TMF, do not upgrade to any version of DMF 2.6.3 or later.

DMF Directory Structure Prior to DMF Release 2.8

Beginning with DMF 2.8, DMF no longer supports multiple installed versions of DMF that can be made active via the `dmmaint`(8) program. While it is not necessary to delete any existing pre-2.8 versions of DMF, they will not accessible by the DMF 2.8 or later software and they can be removed at the convenience of the administrator.

The reason for this change is that the pre-2.8 DMF directory hierarchy of `/usr/dmf/dmbase` is no longer the target installation directory of DMF. Rather, DMF 2.8 and later binaries, libraries, header files, and man pages are installed directly into

the proper system locations and they are accessed directly from those locations without the use of symbolic file links.

When DMF 2.8 or later is installed, if the symbolic file link `/etc/dmf/dmbase` exists, it will be deleted. This link was used in pre-2.8 versions of DMF to access the active version of DMF, and as such, it was part of the administrators' initialization procedure to add this link to their `PATH` environment variable. Because it is no longer used in DMF 2.8 and later versions, it could cause an incorrect copy of a DMF command to be executed if an administrator's path included the link to be searched before the normal system binary locations. This way, even if the administrator neglects to remove the link from the path, it should not make any difference.

End of Life for the Tape MSP after DMF 3.0

DMF 3.0 was the last major release cycle that contained support for the tape MSP. The `dmatmsp` command is not included as part of any DMF 3.5 or later package. When the library server (LS) was introduced in DMF 2.7, the intention was for all existing tape MSPs to be converted to LSs eventually.

It is mandatory that you complete the conversion from tape MSPs to LSs before installing DMF 4.0 or later. SGI highly recommends that you install DMF 3.0.1 for the purpose of doing the conversion because the `dmmsptols` command in that release is much more efficient in terms of time and disk space than in any earlier release.

For more information regarding converting tape MSPs to LSs, see Chapter 13, "Media-Specific Processes and Library Servers" on page 301 or contact SGI Support.

DMF User Library (`libdmfusr.so`) Update in DMF 3.1

The DMF user library (`libdmfusr.so`) was modified significantly in DMF 3.1 and is not backwards compatible with applications written and linked with pre-3.1 versions of `libdmfusr.so`. The library's naming convention has also changed.

This change only impacts sites with site-written applications that link with `libdmfusr.so`. Any site that does have any such applications should immediately refer to Appendix B, "DMF User Library `libdmfusr.so`" on page 385 to find the steps required to keep your site applications operational.

Downgrading and the Site-Tag Feature Introduced in DMF 3.1

DMF 3.1 introduced the site tag feature; see dmtag(1). Site tags are stored in the DMF extended attribute on files. This means that if you have installed and run DMF 3.1 or later and wish to run an earlier version of DMF (pre-DMF 3.1), you must ensure that there are no nonzero site tags on files before installing the earlier version of DMF. Failure to do this will cause errors when running the earlier version of DMF.

Note: Restoring a file that had a site tag from a filesystem backup created while DMF 3.1 or later was running to a system running a pre-3.1 version of DMF is not recommended, because the attribute will appear invalid to the pre-3.1 version of DMF.

To ensure that there are no nonzero site tags, do the following:

1. While DMF is running, execute the following script to clear all site tags in DMF-managed filesystems:

   # **/usr/lib/dmf/support/dmcleartag**

   This command can take some time to run. If there are other DMF requests active for files whose site tags must be cleared, the request to clear the site tag may be queued behind the other request.

2. If the dmcleartag script completed without errors, stop DMF.

3. It is possible that a site tag was set on a file while the dmcleartag script was running, and so there may still be files with nonzero site tags. To verify that there are no nonzero site tags in the DMF-managed filesystems, run the following script:

   # **/usr/lib/dmf/support/dmanytag**

   The script will print a message to stderr if any nonzero site tags are found. If any are found, restart DMF, and repeat step 1. Otherwise, proceed to step 4.

4. Site tags may also be put on files in the DCM or disk MSP STORE_DIRECTORY. The dmcleartag script run in step 1 will clear the site tags on many of these files. However, if there are any soft-deleted files in the DCM or disk MSP STORE_DIRECTORY that have a non-zero site tag, they must be handled while the DMF daemon is not running. Run the following script to clear the tags on soft-deleted DCM copies while the dmfdaemon is stopped:

   # **/usr/lib/dmf/support/dmcleardcmtag**

The DMF attributes should now be in a proper state for running a previous version of DMF.

Downgrading and the Partial-State File Feature Introduced in DMF 3.2

DMF 3.2 introduced the partial-state file feature. Partial-state (PAR) files are not handled by earlier versions of DMF. If customers have installed and run DMF 3.2 or later and then wish to run an earlier version of DMF (pre-DMF 3.2), they must ensure that there are no partial-state files in the DMF-managed filesystems before installing the earlier version of DMF. Failure to do this will cause errors when running the earlier version of DMF.

Follow these steps to ensure that there are no partial-state files:

1. While DMF 3.2 is running, execute the following script to change all partial-state files in DMF-managed filesystems to be offline:

   # **/usr/lib/dmf/support/dmclearpartial**

   This command may take some time to run. If there are other DMF requests active for the partial-state files, the request to make them offline may be queued behind the other request.

2. If the `dmclearpartial` script completed without errors, stop DMF.

3. It is possible that a file was changed to partial-state while the `dmclearpartial` script was running, and so there may still be partial-state files. Verify that there are no partial-state files in the DMF-managed filesystems by running the following script:

   # **/usr/lib/dmf/support/manypartial**

   This script will print a message to `stderr` if any partial-state files are found. If any are found, restart DMF and repeat step 1. Otherwise, proceed to step 4.

4. The partial-state files should now be offline and in a proper state for running a previous version of DMF. If you are installing a version of DMF prior to DMF 3.1, you must also ensure that there are no site tags on DMF-managed files. See the instructions below.

Note: While site tags are being cleared, it is possible that files will be made partial-state. Before running a version of DMF prior to DMF 3.1, check (while DMF is stopped) both that there are no partial-state files and that there are no files with site tags.

`dmaudit`(8) Changes in DMF 3.2

The format of some of the files that `dmaudit` writes changed in DMF 3.2. The DMF 3.2 or later version of `dmaudit` is unable to read the files written by pre-DMF 3.2 versions of `dmaudit`. This means that after upgrading DMF to version 3.2 or later from a pre-DMF 3.2 version, the first time you use `dmaudit`, you must select the `snapshot` option before you can use the `inspect` option.

Logfile Changes in DMF 3.2

A change was made in DMF 3.2 to the way that the DMF daemon and the library server (LS) and MSPs refer to the daemon request number. This change should make it easier for administrators to extract all of the pertinent messages from the `SPOOL_DIR` logs for a particular request.

In previous releases of DMF, the string `Req=`*xxx* could be used to extract some log messages for daemon request number *xxx*, but there were some messages in the form `Req=`*xxx*`/`*nnn* that would not be found (such as by using the `grep`(1) command) with a pattern of `Req=`*xxx*.

A change was made to standardize all daemon and LS/MSP log messages to use the form `Req=`*xxx* for all messages. As a result, a log message formerly of the form `Req=`*xxx*`/`*nnn* would now take the form `Req=`*xxx*`,`*nnn* so as to be visible via the `grep` pattern `Req=`*xxx*. If your site uses these patterns to search DMF `SPOOL_DIR` logs, please be advised of this change and update any scripts or procedures accordingly.

Possible DMF Database Lock Manager Incompatibility On Upgrades as of DMF 3.8.3

The DMF 3.8.3 version of DMF introduced decreased DMF database lock manager delays when processes are making simultaneous lock requests. This code also introduced a backwards incompatibility between pre-3.8.3 `dmlockmgr` processes and post-3.8.3 `dmlockmgr` clients. If DMF is stopped (as recommended) via `/etc/init.d/dmf stop` immediately before installing DMF 3.8.3 or later (in a non-HA environment), there will be no incompatibility.[1]

If, however, one of the DMF administrator commands (`dmdadm`, `dmvoladm`, or `dmcatadm`) is executed after DMF has been stopped and DMF 3.8.3 or later is installed, new `dmlockmgr` clients will hang when trying to request database locks from an older version of `dmlockmgr` that was executing as the result of the administrator command.

For this reason, it is important to make sure that DMF, including the `dmlockmgr` process, is stopped via `/etc/init.d/dmf stop` immediately before installing DMF 3.8.3 or later even if the DMF daemon is not running, if you are upgrading from a pre-3.8.3 version of DMF.

[1] In an HA environment, you must first remove Heartbeat control of the resource group before stopping DMF and the mounting service. See the SGI HA documentation.

Glossary

accelerated access to first byte

A partial-state file feature capability that allows you to access the beginning of an offline file before the entire file has been recalled.

active database entry

A valid daemon database entry. See also *soft-deleted database entry* and *hard-deleted database entry*.

active metadata server

A CXFS server-capable administration node chosen from the list of potential metadata servers. There can be only one active metadata server for any one filesystem. See also *metadata*.

active parallel data mover node

A parallel data mover node that has been enabled using `dmnode_admin`(8), has not exceeded the number of parallel data mover node licenses on the DMF server, and is connected to the `dmnode` service on the DMF server. See also *parallel data mover node* and *parallel data mover node license*.

`ADMIN_EMAIL`

The e-mail address to receive output from administrative tasks. See "`base` Object" on page 152.

administrative filesystems

See *DMF administrative filesystems*.

`AGE_WEIGHT`

A floating-point constant and floating-point multiplier to use to calculate the weight given to a file's age (for MSP/VG user filesystem). See "File Weighting Parameters for a User Filesystem" on page 198.

ALGORITHM

The resource scheduling algorithm to be used. See "resourcescheduler Object" on page 230.

allocation group

A source of additional volumes for a VG that runs out of media. An allocation group defines a logical pool of volumes, and is different from an actual operational VG. Normally, one allocation group is configured to serve multiple VGs. If a VG has an associated allocation group, when the VG runs out of empty volumes, the LS assigns one from the allocation group to it, subject to configuration restrictions. Similarly, when a volume's hfree flag is cleared in a VG, it is returned to the allocation group, subject to configuration restrictions. The use of allocation groups is optional. Allocation groups are defined in the DMF configuration file (/etc/dmf/dmf.conf).

ALLOCATION_GROUP

The group that serves as a source of additional volumes if a VG runs out of media. See "volumegroup Object" on page 224.

ALLOCATION_MAXIMUM

The maximum size in number of volumes to which a VG can grow by borrowing volumes from its allocation group. See "volumegroup Object" on page 224.

ALLOCATION_MINIMUM

The minimum size in number of volumes to which a VG can shrink by returning volumes to its allocation group. See "volumegroup Object" on page 224.

alternate media

The media onto which migrated data blocks are stored, usually tapes.

automated space management

The combination of utilities that allows DMF to maintain a specified level of free space on a filesystem through automatic file migration.

`BANDWIDTH_MULTIPLIER`

(OpenVault only) A floating point number used to adjust the amount of bandwidth that the LS assumes a drive in the drive group will use. See "`drivegroup` Object" on page 217.

`base` **object**

The configuration file object that defines pathname and file size parameters necessary for DMF operation. See "`base` Object" on page 152.

basic DMF

DMF without the Parallel Data Mover Option.

BFID

See *bit-file identifier*.

BFID set

The collection of database entries and the user file associated with a particular bit-file identifier.

BFID-set state

The sum of the states of the components that constitute a bit-file identifier set: the file state of any user file and the state of any database entries (incomplete, complete, soft-deleted, or active).

bitfile ID

See *bit-file identifier*.

bit-file identifier

(BFID) A unique identifier, assigned to each file during the migration process, that links a migrated file to its data on alternate media.

block

Physical unit of I/O to and from media, usually tape. The size of a block is determined by the type of device being written. A tape block is accompanied by a header identifying the chunk number, zone number, and its position within the chunk.

BLOCK_SIZE

The maximum block size to use when writing from the beginning of a tape. See "drivegroup Object" on page 217.

BUFFERED_IO_SIZE

The size of I/O requests when reading from a filesystem using buffered I/O. See:

- "filesystem Object" on page 187
- "Disk Cache Manager (DCM) msp Object" on page 254

CACHE_AGE_WEIGHT

The floating-point constant and floating-point multiplier used to calculate the weight given to a file's age (for DCM STORE_DIRECTORY). See "Automated Space Management Parameters for a DCM STORE_DIRECTORY" on page 201.

CACHE_DIR

The directory in which the VG stores chunks while merging them from sparse tapes. See "libraryserver Object" on page 215.

CACHE_SPACE

The amount of disk space (in bytes) that dmatls can use when merging chunks from sparse tapes. See "libraryserver Object" on page 215.

CACHE_SPACE_WEIGHT

The floating-point constant and floating-point multiplier to use to calculate the weight given to a file's size (for DCM STORE_DIRECTORY). See "Automated Space Management Parameters for a DCM STORE_DIRECTORY" on page 201.

candidate list

A list that contains an entry for each file in a filesystem eligible for migration, or for a file or range of a files that are eligible to be made offline. This list is ordered from largest file weight (first to be migrated) to smallest. This list is generated and used internally by dmfsmon(8). The dmscanfs(8) command prints similar file status information to standard output.

capability license

See *server capability license.*

capacity license

One or more cumulative DMF licenses that permit DMF migration, corresponding to the amount of data that DMF is currently managing. See also *server capability license.*

CAT record

An entry in the catalog (CAT) table of the LS database that tracks the location of migrated data on a tape volume. There is one CAT record for each migrated copy of a file. (If a migrated copy of a file is divided between two physical media, there will be a CAT record for each portion.) See also *VOL record.*

CAT table

A table in the LS database that contains CAT records. See also *VOL table.*

`CHILD_MAXIMUM`

The maximum number of child processes that the MSP is allowed to fork. See:

- "FTP `msp` Object" on page 245
- "Disk `msp` Object" on page 250
- "Disk Cache Manager (DCM) `msp` Object" on page 254

chunk

That portion of a user file that fits on the current media volume. Most small files are written as single chunks. When a migrated file cannot fit onto a single volume, the file is split into chunks.

client-only node

A node that is installed with the `cxfs_client.sw.base` software product; it does not run cluster administration daemons and is not capable of coordinating CXFS metadata. See also *server-capable administration node.*

`COMMAND`

The binary file to execute in order to initiate an MSP or LS. See:

- "`libraryserver` Object" on page 215
- "FTP `msp` Object" on page 245
- "Disk `msp` Object" on page 250
- "Disk Cache Manager (DCM) `msp` Object" on page 254

complete daemon-database entry

An entry in the daemon database whose `path` field contains a key returned by its MSP or VG, indicating that the MSP or VG maintains a valid copy of the user file.

compression

The mechanism by which data is reduced as it is written to tape.

configuration file object

A series of parameter definitions in the DMF configuration file that controls the way in which DMF operates. By changing the parameters associated with objects, you can modify the behavior of DMF.

configuration parameter

A string in the DMF configuration file that defines a part of a configuration object. By changing the values associated with these parameters, you can modify the behavior of DMF. The parameter serves as the name of the line. Some parameters are reserved words, some are supplied by the site.

configuration stanza

A sequence of configuration parameters that define a configuration object.

CXFS

Clustered XFS, a parallel-access shared clustered filesystem for high-performance computing environments.

daemon

A program that is run automatically by the system for a specific purpose.

daemon database

A database maintained by the DMF daemon. This database contains information such as the bit-file identifier, the MSP or VG name, and MSP or VG key for each copy of a migrated file.

`dmdaemon` **object**

The configuration file object that defines parameters necessary for `dmfdaemon`(8) operation. See "`dmdaemon` Object" on page 160.

DASD

See *direct-access storage device*.

`DATA_LIMIT`

The maximum amount of data (in bytes) that should be selected for merging at one time. See "LS Tasks" on page 240.

`DATABASE_COPIES`

One or more directories into which a copy of the DMF databases will be placed. See "`taskgroup` Object" on page 170.

data mover

A node running *data mover processes* to migrate and recall data to tape, either a *DMF server* or a *parallel data mover node*.

data mover processes

The individual processes that migrate data to tape (using the *write child*) and recall data from tape (using the *read child*).

data-pointer area

The portion of the inode that points to the file's data blocks.

`device` **object**

The configuration file object that defines parameters for the DMF backup scripts' use of tape devices other than those defined by a drive group. See "`device` Object" on page 186.

DCM

The *disk cache manager* is a disk MSP that is configured for *n*–tier capability, which lets you manage data on secondary storage, allowing you to further migrate the data to tape as needed.

DG

See *drive group*.

`DIRECT_IO_MAXIMUM_SIZE`

The maximum size of I/O requests when using `O_DIRECT` I/O to read from any primary filesystem or when migrating files down the hierarchy from the `STORE_DIRECTORY` of a DCM. See "`base` Object" on page 152.

`DIRECT_IO_SIZE`

The size of I/O requests when reading from this filesystem using direct I/O. See:

- "`filesystem` Object" on page 187
- "Disk Cache Manager (DCM) `msp` Object" on page 254

`DISCONNECT_TIMEOUT`

Specifies the number of seconds after which the LS will consider a mover process to have exited if it cannot communicate with the process. See "`libraryserver` Object" on page 215.

disk cache

Data on secondary storage.

disk cache manager

See *DCM*.

DMF administrative filesystems and directories

The set of filesystems and directories in which DMF stores databases, log and journal files, and temporary file directories. The DMF configuration file specifies these filesystems using the following parameters:

```
HOME_DIR
JOURNAL_DIR
SPOOL_DIR
TMP_DIR
MOVE_FS
CACHE_DIR
STORE_DIRECTORY
```

DMF daemon

The program that accepts requests to migrate data, communicates with the operating system kernel in order to maintain a file's migration state, determines the destination of migrated data, and requests the return of offline copies.

DMF direct archiving

The DMF feature that lets users manually archive files from an unmanaged POSIX filesystem directly to secondary storage via the `dmarchive`(1) command. See "DMF Direct Archiving: Copying Unmanaged File Data to Secondary Storage" on page 13.

DMF server

A node running the required DMF server software that provides DMF administration, configuration, and data mover functionality. (When using the Parallel Data Mover Option, data mover functionality is optional on the DMF server.)

DMF state

See *file state*.

drive group

(DG) One of the components of an LS. The drive group is responsible for the management of a group of interchangeable tape drives located in the tape library. These drives can be used by multiple VGs and by non-DMF processes, such as backups and interactive users. However, in the latter cases, the drive group has no management involvement; the mounting service (TMF or OpenVault) is responsible

for ensuring that these possibly competing uses of the tape drives do not interfere with each other. The main tasks of the drive group are to monitor tape I/O for errors, to attempt to classify them (as volume, drive, or mounting service problems), and to take preventive action.

`drivegroup` **object**

The configuration file object that defines a drive group, one for each pool of interchangeable drives in a single library. See "`drivegroup` Object" on page 217.

`DRIVE_GROUPS`

One or more drive groups containing drives that the LS can use for mounting and unmounting volumes. See "`libraryserver` Object" on page 215.

`DRIVE_MAXIMUM`

The maximum number of drives within this drive group that the LS is allowed to attempt to use simultaneously. See:

- "`drivegroup` Object" on page 217
- "`volumegroup` Object" on page 224

`DRIVE_SCHEDULER`

The resource scheduler that the drive group should run for the scheduling of tape drives. See:

- "`drivegroup` Object" on page 217
- "`volumegroup` Object" on page 224

`DRIVES_TO_DOWN`

An integer value that controls the number of "bad" drives the drive group is allowed to try to configure down. See "`drivegroup` Object" on page 217.

`DRIVETAB`

This optional parameter provides the name of a file that is used with the `tsreport --drivetab` option, which causes the `run_daily_drive_report` and `run_daily_tsreport` output to contain the more readable drive name instead of the device name.

DSK_BUFSIZE

The transfer size in bytes used when reading from and writing to files within the disk MSP's STORE_DIRECTORY. See:

- "Disk msp Object" on page 250
- "Disk Cache Manager (DCM) msp Object" on page 254

DUALRESIDENCE_TARGET

The percentage of DCM cache capacity that DMF maintains as a reserve of dual-state files whose online space can be freed if free space reaches or falls below FREE_SPACE_MINIMUM (for DCM STORE_DIRECTORY). See:

- "Automated Space Management Parameters for a DCM STORE_DIRECTORY" on page 201
- "Disk Cache Manager (DCM) msp Object" on page 254

dual-resident file

In a DCM, a cache-resident copy of a migrated file that has already been copied to tape, and can therefore be released quickly in order to prevent the cache filling, without any need to first copy it to tape (analogous to a *dual-state file*).

dual-state file

A file whose data resides both online and offline.

DUMP_DATABASE_COPY

The path to a directory where a snapshot of the DMF databases will be placed when do_predump.sh is run. See "taskgroup Object" on page 170.

DUMP_DEVICE

The name of the drive group in the configuration file that defines how to mount the tapes that the dump tasks will use. See "taskgroup Object" on page 170.

DUMP_FILE_SYSTEMS

One or more filesystems to dump. If not specified, the tasks will dump all the DMF-managed user filesystems configured in the configuration file. See "taskgroup Object" on page 170.

DUMP_FLUSH_DCM_FIRST

Specifies whether or not the dmmigrate command is run before the dumps are done to ensure that all non-dual-resident files in the DCM caches are migrated to tape. See "taskgroup Object" on page 170.

DUMP_INVENTORY_COPY

The pathnames of one or more directories into which are copied the XFS inventory files for the backed-up filesystems. See "taskgroup Object" on page 170.

DUMP_MAX_FILESPACE

The maximum disk space used for files to be dumped, which may be larger or smaller than the length of the file. See "taskgroup Object" on page 170.

DUMP_MIGRATE_FIRST

The parameter that specifies whether or not the dmmigrate command is run before the dumps are done to ensure that all migratable files in the DMF-managed user filesystems are migrated, thus reducing the number of tapes needed for the dump and making it run much faster. See "taskgroup Object" on page 170.

DUMP_RETENTION

The length of time that the backups of the filesystem will be kept before the tapes are reused. See "taskgroup Object" on page 170.

DUMP_TAPES

The path of a file that contains tape volume serial numbers (VSNs), one per line, for the dump tasks to use. See "taskgroup Object" on page 170.

DUMP_VSNS_USED

A file in which the VSNs of tapes that are used are written. See "taskgroup Object" on page 170.

DUMP_XFSDUMP_PARAMS

Passes parameters to the xfsdump program. See "taskgroup Object" on page 170.

EXPORT_METRICS

Enables DMF's use of the common arena for collecting DMF statistics for use by dmstat(8), dmarenadump(8), and other commands. See "base Object" on page 152.

EXPORT_QUEUE

Instructs the daemon to export details of its internal request queue to *SPOOL_DIR*/daemon_exports every two minutes, for use by dmstat (8)and other utilities. See "dmdaemon Object" on page 160.

FADV_SIZE_MSP

Specifies the size of files in the MSP's STORE_DIRECTORY for which posix_fadvise() will be called with advice POSIX_FADV_DONTNEED. See:

- "Disk msp Object" on page 250
- "Disk Cache Manager (DCM) msp Object" on page 254

fhandle

See *file handle.*

file

An inode and its associated data blocks; an empty file has an inode but no data blocks.

file handle

The DMAPI identification for a file. You can use the dmscanfs(8), dmattr(1), and dmfind(1) commands to find file handles.

file state

The migration state of a file as indicated by the dmattr(1) command. A file can be regular (not migrated), migrating, dual-state, offline, partial-state, unmigrating, never-migrated, or have an invalid DMF state.

file tag

A site-assigned 32-bit integer associated with a specific file, allowing the file to be identified and acted upon.

`filesystem` **object**

The configuration file object that defines parameters necessary for migrating files in that filesystem. See "`filesystem` Object" on page 187.

`FREE_DUALRESIDENT_FIRST`

Specifies whether `dmdskfree` will free dual-resident files in the DCM before freeing files it would have to move to tape first (for DCM `STORE_DIRECTORY`). See "Automated Space Management Parameters for a DCM `STORE_DIRECTORY`" on page 201.

`FREE_DUALSTATE_FIRST`

Specifies whether or not `dmfsmon` will first free dual-state and partial-state files before freeing files it must migrate (for MSP/VG user filesystem). See "Automated Space Management Parameters for a User Filesystem" on page 197.

`FREE_SPACE_DECREMENT`

The percentage of filesystem space by which `dmfsmon` will decrement `FREE_SPACE_MINIMUM` (if it cannot find enough files to migrate) so that the value is reached. The decrement is applied until a value is found that `dmfsmon` can achieve. See:

- "Automated Space Management Parameters for a User Filesystem" on page 197
- "Automated Space Management Parameters for a DCM `STORE_DIRECTORY`" on page 201

`FREE_SPACE_MINIMUM`

The minimum percentage of free filesystem space that `dmfsmon` maintains. `dmfsmon` will begin to migrate files when the available free space for the filesystem falls below this percentage value. See:

- "Automated Space Management Parameters for a User Filesystem" on page 197
- "Automated Space Management Parameters for a DCM `STORE_DIRECTORY`" on page 201

FREE_SPACE_TARGET

The percentage of free filesystem space that dmfsmon will try to achieve if free space reaches or falls below FREE_SPACE_MINIMUM. See:

- "Automated Space Management Parameters for a User Filesystem" on page 197
- "Automated Space Management Parameters for a DCM STORE_DIRECTORY" on page 201

freed file

A user file that has been migrated and whose data blocks have been released.

FTP_ACCOUNT

The account ID to use when migrating files to the remote system. See "FTP msp Object" on page 245.

FTP_COMMAND

Additional commands to send to the remote system. See "FTP msp Object" on page 245.

FTP_DIRECTORY

The directory to use on the remote system. See "FTP msp Object" on page 245.

FTP_HOST

The Internet hostname of the remote machine on which files are to be stored. See "FTP msp Object" on page 245.

FTP MSP

The daemon-like media-specific process (MSP) that copies data blocks onto alternate media and assigns keys to identify the location of the migrated data using the file transfer protocol (FTP) to transfer to and from disks of another system on the network.

FTP_PASSWORD

The file containing the password to use when migrating files to the remote system. This file must be owned by root and be only accessible by root. See "FTP msp Object" on page 245.

FTP_PORT

The port number of the FTP server on the remote system. See "FTP msp Object" on page 245.

FTP_USER

The user name to use when migrating files to the remote system. See "FTP msp Object" on page 245.

GUARANTEED_DELETES

The number of child processes that are guaranteed to be available for processing delete requests. See:

- "FTP msp Object" on page 245
- "Disk msp Object" on page 250
- "Disk Cache Manager (DCM) msp Object" on page 254

GUARANTEED_GETS

The number of child processes that are guaranteed to be available for processing dmget(1) requests. See:

- "FTP msp Object" on page 245
- "Disk msp Object" on page 250
- "Disk Cache Manager (DCM) msp Object" on page 254

HA

High availability

HA resource

A service, associated with an IP address, that is managed by Linux-HA Heartbeat. Also see *resource* for DMF Manager.

HA_VIRTUAL_HOSTNAME

The virtual hostname. See "node Object" on page 163.

hard-deleted database entry

An MSP or VG database entry that has been removed from the daemon database and whose MSP or VG copy has been discarded. See also *active database entry* and *soft-deleted database entry*.

HBA_BANDWIDTH

(OpenVault only) The I/O bandwidth capacity of an HBA port that is connected to tape drives on a node. See:

- "base Object" on page 152
- "node Object" on page 163

HFREE_TIME

The minimum number of seconds that a tape no longer containing valid data must remain unused before the VG overwrites it. See "volumegroup Object" on page 224.

HOME_DIR

The base pathname for directories in which DMF databases and related files reside. See "base Object" on page 152.

HTML_REFRESH

The refresh rate (in seconds) of the generated HTML pages. See "resourcewatcher Object" on page 231.

IMPORT_DELETE

A parameter that specifies that the MSP should honor hard-delete requests from the DMF daemon. See:

- "FTP msp Object" on page 245
- "Disk msp Object" on page 250

IMPORT_ONLY

A parameter that specifies that the MSP is used for importing only. See:

- "FTP msp Object" on page 245
- "Disk msp Object" on page 250

incomplete daemon-database entry

An entry in the daemon database for a MSP or VG that has not finished copying the data, and therefore has not yet returned a key. The path field in the database entry is NULL.

incompletely migrated file

A file that has begun the migration process, but for which one or more copies on alternate media have not yet been made.

inode

The portion of a file that contains the bit-file identifier, the state field, and the data pointers.

integrated data mover functionality

The ability of the DMF server to move data. See also *parallel data mover node*.

JOURNAL_DIR

The base pathname for directories in which the daemon database and LS journal files will be written. See "base Object" on page 152.

JOURNAL_RETENTION

The length of time to keep journals. See "taskgroup Object" on page 170.

JOURNAL_SIZE

The maximum size (in bytes) of the database journal file before DMF closes it and starts a new file. See "base Object" on page 152.

LABEL_TYPE

The label type used when writing tapes from the beginning. See "drivegroup Object" on page 217.

library server

See *LS*.

LS

The library server (LS) is a daemon-like process by which data blocks are copied onto tape and that maintains the location of the migrated data. Each LS has an associated LS database with catalog (CAT) and volume (VOL) records. An LS can be configured to contain one or more drive groups. Each drive group contains one or more VGs. A VG is responsible for copying data blocks onto alternate media. A VG is capable of managing a single copy of a user file.

LS database

The database containing catalog (CAT) and volume (VOL) records associated with a library server (LS). See also *CAT record* and *VOL record*.

libraryserver **object**

The configuration file object that defines parameters relating to a tape library for an LS. See "libraryserver Object" on page 215.

LICENSE_FILE

The full pathname of the file containing the license used by DMF. See "base Object" on page 152.

LOG_RETENTION

The length of time to keep log files. See "taskgroup Object" on page 170.

LS_NAMES

The library servers used by the DMF daemon. See "dmdaemon Object" on page 160.

managed filesystem

A DMAPI-mounted XFS or CXFS filesystem, configured in a `filesystem` object in the DMF configuration file, on which DMF can migrate or recall files. (When using the Parallel Data Mover Option, it must be CXFS.)

`MAX_CACHE_FILE`

The largest chunk (in bytes) that will be merged using the merge disk cache. See "`libraryserver` Object" on page 215.

`MAX_CHUNK_SIZE`

The configuration parameter that specifies that the VG should break up large files into chunks no larger than this value (specified in bytes) as it writes data to tape. See "`volumegroup` Object" on page 224.

`MAX_DRIVES_PER_NODE`

(This parameter has been deprecated and will be ignored.)

`MAX_MANAGED_REGIONS`

The maximum number of managed regions that DMF will assign to a file on a per-filesystem basis. You can set `MAX_MANAGED_REGIONS` to any number that is less than the actual number of regions that will fit in a filesystem attribute. See "`filesystem` Object" on page 187.

`MAX_MS_RESTARTS`

The maximum number of times DMF can attempt to restart the mounting service (TMF or OpenVault) without requiring administrator intervention. See "`drivegroup` Object" on page 217.

`MAX_PUT_CHILDREN`

The maximum number of write child (`dmatwc`) processes that will be simultaneously scheduled for the VG. See "`volumegroup` Object" on page 224.

media-specific process

(MSP) The daemon-like process by which data blocks are copied onto alternate media and that assigns keys to identify the location of the migrated data.

MERGE_CUTOFF

A limit at which the VG will stop scheduling tapes for merging. See "volumegroup Object" on page 224.

merging

See *volume merging*.

MESSAGE_LEVEL

The highest message level that will be written to a log file (the higher the number, the more messages written). See:

- "dmdaemon Object" on page 160
- "services Object" on page 167
- "filesystem Object" on page 187
- "libraryserver Object" on page 215
- "FTP msp Object" on page 245
- "Disk msp Object" on page 250
- "Disk Cache Manager (DCM) msp Object" on page 254
- Chapter 9, "Message Logs" on page 277

metadata

Information that describes a file, such as the file's name, size, location, and permissions.

metadata server

The CXFS server-capable administration node that coordinates updating of metadata on behalf of all nodes in a cluster. There can be multiple potential metadata servers, but only one is chosen to be the active metadata server for any one filesystem.

migrated file

A file that has one or more complete offline copies and no pending or incomplete offline copies.

migrating file

A file that has a bit-file identifier but whose offline copies are in progress.

MIGRATION_LEVEL

The highest level of migration service allowed. See:

- "dmdaemon Object" on page 160
- "filesystem Object" on page 187
- "FTP msp Object" on page 245
- "Disk msp Object" on page 250
- "Disk Cache Manager (DCM) msp Object" on page 254

MIGRATION_TARGET

The percentage of filesystem capacity that DMF maintains as a reserve of dual-state files whose online space can be freed if free space reaches or falls below FREE_SPACE_MINIMUM (for MSP/VG user filesystem). See "Automated Space Management Parameters for a User Filesystem" on page 197.

MIN_ARCHIVE_SIZE

Determines whether direct or buffered I/O is used when reading from this filesystem. See "filesystem Object" on page 187.

MIN_DIRECT_SIZE

Determines whether direct or buffered I/O is used when reading from this filesystem. See:

- "filesystem Object" on page 187
- "Disk Cache Manager (DCM) msp Object" on page 254
- open(2) man page for a description of direct I/O

MIN_VOLUMES

The minimum number of unused volumes that can exist in the LS database for this VG without operator notification. See "volumegroup Object" on page 224.

MODULE_PATH

The path name of a Dynamic Shared Object (library of runtime-loadable routines) containing the scheduling algorithm. For more information, see the dmf.conf(5) man page.

MOUNT_SERVICE

The mounting service. See:

- "device Object" on page 186
- "drivegroup Object" on page 217

MOUNT_SERVICE_GROUP

The name by which the object's devices are known to the mounting service. See:

- "device Object" on page 186
- "drivegroup Object" on page 217

MOUNT_TIMEOUT

The maximum number of minutes to wait for a tape to be mounted. See "drivegroup Object" on page 217.

MOVE_FS

The scratch filesystem used by dmmove(8) to move files between MSPs or VGs. See "dmdaemon Object" on page 160.

MSG_DELAY

The number of seconds that all drives in the drive group can be down before an e-mail message is sent to the administrator and an error message is logged. See "drivegroup Object" on page 217.

MSP

The media-specific process (MSP), a daemon-like process by which data blocks are copied onto alternate media and that assigns keys to identify the location of the migrated data.

MSP database entry

The daemon database entry for a file that contains the path or key that is used to inform a particular media-specific process (MSP) where to locate the copy of the file's data.

`MSP_NAMES`

Names the media-specific processes (MSPs) used by the DMF daemon. See "`dmdaemon` Object" on page 160.

`msp` **object**

The configuration file object that defines parameters necessary for the operation of a media-specific process. There is one `msp` object for each MSP. See:

- "FTP `msp` Object" on page 245
- "Disk `msp` Object" on page 250
- "Disk Cache Manager (DCM) `msp` Object" on page 254

`MVS_UNIT`

The storage device type on an MVS system. See "FTP `msp` Object" on page 245.

`NAME_FORMAT`

The strings that form a template to create names for files stored on remote machines in the `STORE_DIRECTORY`. This parameter is also used by the disk MSP and the DCM MSP, where it provides a template for file names in `STORE_DIRECTORY`. See:

- "FTP `msp` Object" on page 245
- "Disk `msp` Object" on page 250
- "Disk Cache Manager (DCM) `msp` Object" on page 254

near-line storage

Storage in which tapes are mounted by robot.

NODE_ANNOUNCE_RATE

The rate in seconds at which a node will contact the dmnode_service on the DMF server to announce its presence. See "services Object" on page 167.

NODE_BANDWIDTH

(OpenVault only) The I/O bandwidth capacity of the node. See:

- "base Object" on page 152
- "node Object" on page 163

NODE_TIMEOUT

The number of seconds after which the data mover functionality on the DMF server or on a parallel data mover node will be considered inactive if it has not contacted the dmnode_service on the DMF server. See "services Object" on page 167.

nonmigrated file

A file that does not have a bit-file identifier or any offline copies. See *regular file*.

offline file

A file whose inode contains a bit-file identifier but whose disk blocks have been removed. The file's data exists elsewhere in copies on alternate media.

offline pointer

In MSP and VG processing, a character string that the MSP or VG returns to the daemon to indicate how a file is to be retrieved.

OpenVault

A storage library management facility that improves how applications can manage, store, and retrieve removable media.

orphan chunk

An unused area in an LS catalog (CAT) database entry resulting from the removal of migrated files.

orphan database entry

An unused daemon database entry resulting from the removal of a migrated file during a period in which the DMF daemon is not running.

`OV_ACCESS_MODES`

The OpenVault access mode. See:

- "`device` Object" on page 186
- "`drivegroup` Object" on page 217

`OV_INTERCHANGE_MODES`

A list of interchange mode names that control how data is written to tape. See:

- "`device` Object" on page 186
- "`drivegroup` Object" on page 217

`OV_KEY_FILE`

The file containing the OpenVault keys used by DMF. See "`base` Object" on page 152.

`OV_SERVER`

The name of the OpenVault server (required only if it differs from the DMF server). See "`base` Object" on page 152.

parallel data mover node

A node, installed with DMF data mover software and underlying CXFS client-only software, that provides dedicated data mover functionality in addition to the DMF server, increasing data throughput and enhancing resiliency.

parallel data mover node license

A DMF license installed on the DMF server that permits one parallel data mover node to be active when using the Parallel Data Mover Option. There can be multiple licenses installed, one for each parallel data mover node that is active at any one time. See also *parallel data mover node* and *Parallel Data Mover Option*.

Parallel Data Mover Option

Optional software and licenses available for purchase that allow you to run parallel data mover nodes in order to increase data throughput and enhance resiliency.

parameter

See *configuration parameter*.

partial-state file

A file that has more than one region. DMF allows a file to include up to four distinct file regions. See also *region*.

partial-state file online retention

A partial-state file feature capability that allows you to keep a specific region of a file online while freeing the rest of it (for example, if you wanted to keep just the beginning of a file online). See also *partial-state file*.

partial-state file recall

A partial-state file feature capability that allows you to recall a specific region of a file without recalling the entire file. For more information, see the `dmput`(1) and `dmget`(1) man pages. See also *partial-state file*.

`PARTIAL_STATE_FILES`

Enables or disables the DMF daemon's ability to produce partial-state files. See "`dmdaemon` Object" on page 160.

`PENALTY`

A parameter used to reduce the priority of requests from a VG that is not the next one preferred by the round-robin algorithm. See "`resourcescheduler` Object" on page 230.

`POLICIES`

The names of the configuration objects defining policies for this filesystem. See:

- "`filesystem` Object" on page 187
- "Disk Cache Manager (DCM) `msp` Object" on page 254

policy

Rules that tell DMF how to determine MSP or VG selection, automated space-management policies, and/or file weight calculations.

`policy` **object**

The configuration file object that specifies parameters to determine MSP or VG selection, automated space management policies, and/or file weight calculations in automated space management. See "`policy` Object" on page 193.

`POSITIONING`

How the tape should be positioned. See "`drivegroup` Object" on page 217.

`POSITION_RETRY`

The level of retry in the event of a failure during zone positioning. See "`drivegroup` Object" on page 217.

`POSIX_FADVISE_SIZE`

Specifies the number of bytes after which DMF will call `posix_fadvise()` with advice `POSIX_FADV_DONTNEED` when recalling files. See "`filesystem` Object" on page 187.

`PRIORITY_PERIOD`

Specifies the number of minutes after which a migrating file gets special treatment. See "Disk Cache Manager (DCM) `msp` Object" on page 254.

`PUTS_TIME`

The minimum number of seconds a VG waits after it has requested a drive for a write child before it tells a lower priority child to go away. See "`volumegroup` Object" on page 224.

raw time

The time in seconds since January 1, 1970.

read child

A data mover process that recalls data from tape.

READ_ERR_MAXIMUM

The maximum number of I/O errors that will be tolerated when reading from tape to recall a file. See "drivegroup Object" on page 217.

READ_ERR_MINIMUM

The minimum number of I/O errors that will be tolerated when reading from tape to recall a file. See "drivegroup Object" on page 217.

READ_ERR_TIMEOUT

The elapsed number of seconds since the first I/O error was seen. See "drivegroup Object" on page 217.

READ_IDLE_DELAY

The number of seconds an idle tape LS read child (dmatrc) can wait before being told to exit. See "drivegroup Object" on page 217.

READ_TIME

The interval, in seconds, after which the VG will evaluate whether a read child should be asked to go away (even if it is in the middle of recalling a file) so that a higher priority child can be started. See "volumegroup Object" on page 224.

recall

To request that a migrated file's data be moved back (unmigrated) onto the filesystem disk, either by explicitly entering the dmget(1) command or by executing another command that will open the file, such as the vi(1) command.

RECALL_NOTIFICATION_RATE

The approximate rate, in seconds, at which regions of a file being recalled are put online. This allows for access to part of a file before the entire file is recalled. See "dmdaemon Object" on page 160.

region

A contiguous range of bytes that have the same residency state. The states can be DUALSTATE, OFFLINE, MIGRATING, or UNMIGRATING.

regular file

DMF considers a regular file to be one with no bit-file identifier and no offline copies.

`REINSTATE_DRIVE_DELAY`

The number of minutes after which a drive that was configured down by the drive group will be automatically reinstated and made available for use again. See "`drivegroup` Object" on page 217.

`REINSTATE_VOLUME_DELAY`

The number of minutes after which a volume that had its `HLOCK` flag set by DMF will be automatically reinstated and made available for use again. See "`drivegroup` Object" on page 217.

removable media

A tape cartridge, a tape reel, an optical disc, a digital linear tape, a removable magnetic disk, or a videotape.

resource

A *resource* is a filesystem or hardware component used by DMF. Also see *HA resource*

resource group

A service, associated with an IP address, that is managed by Linux-HA Heartbeat.

`resourcescheduler` **object**

The configuration file object that defines parameters relating to scheduling of tape devices in a drive group when requests from VGs exceed the number of devices available. See "`resourcewatcher` Object" on page 231.

`resourcewatcher` **object**

The configuration file object that defines parameters relating to the production of files informing the administrator about the status of the LS and its components. See "`resourcewatcher` Object" on page 231.

REWIND_DELAY

The number of seconds an idle tape LS read child (dmatrc) can wait before rewinding. See "drivegroup Object" on page 217.

RUN_TASK

A DMF maintenance command to be executed. See:

- "Automated Maintenance Tasks" on page 89
- "taskgroup Object" on page 170
- "libraryserver Object" on page 215
- "drivegroup Object" on page 217
- "volumegroup Object" on page 224

SCAN_FAST

The parameter that specifies whether or not the run_filesystem_scan.sh task will use fast scanning. If set to no, the dmscanfs command will use its recursive option, which is much slower but results in pathnames being included in the output file. See "taskgroup Object" on page 170.

SCAN_FILESYSTEMS

The parameter that specifies for the run_filesystem_scan.sh script the filesystem that dmscanfs will scan. See "taskgroup Object" on page 170.

SCAN_FOR_DMSTAT

The parameter that specifies for the run_filesystem_scan.sh script whether the fhandle2bfid+path file is created. That file may be used by the dmemptytape (8)command. See "taskgroup Object" on page 170.

SCAN_OUTPUT

The parameter that specifies for the run_filesystem_scan.sh script the name of the file into which dmscanfs will place output. See "taskgroup Object" on page 170.

SCAN_PARAMS

The parameter that specifies for the run_filesystem_scan.sh script other dmscanfs parameters, such as for example requesting a nondefault output format. See "taskgroup Object" on page 170.

SCAN_PARALLEL

The parameter that specifies for the run_filesystem_scan.sh script whether dmscanfs will scan filesystems in parallel. See "taskgroup Object" on page 170.

SELECT_LOWER_VG

Defines which VGs should maintain tape-based copies of files in the cache, and under what conditions that would define dual-residence. (It is not used for defining which VG to use for recalls; for that, see the definitions of the LS_NAMES, MSP_NAMES, DRIVE_GROUPS, and VOLUME_GROUPS parameters.) See "VG Selection Parameters for a DCM STORE_DIRECTORY" on page 204.

SELECT_MSP

The media-specific processes (MSPs) to use for migrating a file (for Disk MSP / DCM MSP FTP MSP user filesystem). See "MSP/VG Selection Parameters for a User Filesystem" on page 200.

SELECT_VG

The volume groups (VGs) to use for migrating a file (for LS user filesystem). See "MSP/VG Selection Parameters for a User Filesystem" on page 200.

server capability license

The DMF license that permits DMF migrations to exceed 1 TB when installed in conjunction with one or more DMF capacity licenses. See also *capacity licenses*.

SERVER_NAME

Hostname of the machine on which the DMF server is running (used for HA configurations or configurations using the DMF Parallel Data Mover Option). See "base Object" on page 152.

SERVICES

The name of the `services` object used to configure DMF services on a node. See:

- "`node` Object" on page 163
- "`services` Object" on page 167

SERVICES_PORT

The port number on which DMF starts a locator service, which DMF uses to locate other DMF services. See "`services` Object" on page 167.

site-defined policy

A site-specific library of C++ functions that DMF will consult when making decisions about its operation.

SITE_SCRIPT

The site-specific script to execute when `dmfsfree`, `dmdskfree`, or `dmfsmon` is run. See:

- "Disk Cache Manager (DCM) `msp` Object" on page 254
- "Automated Space Management Parameters for a User Filesystem" on page 197
- "Automated Space Management Parameters for a DCM `STORE_DIRECTORY`" on page 201

snapshot

The information about all bit-file identifier sets that is collected and analyzed by `dmaudit`(8). The snapshot analysis is available from the `report` function.

soft-deleted database entry

A daemon database entry for which the MSP or VG copy of the data is no longer valid. Data remains on the alternate media until the database entry is hard-deleted. See also *active database entry* and *hard-deleted database entry*.

SPACE_WEIGHT

The floating-point constant and floating-point multiplier to use to calculate the weight given to a file's size (for MSP/VG user filesystem). See "File Weighting Parameters for a User Filesystem" on page 198.

sparse tape

A tape containing only a small amount of active information.

special file

A device file in UNIX or Linux. (DMF never migrates special files.)

SPOOL_DIR

The base pathname for directories in which DMF log files are kept. See "base Object" on page 152.

standby metadata server node

A CXFS server-capable administration node that is configured as a potential metadata server for a given filesystem, but does not currently run any applications that will use that filesystem.

state field

The field in the inode that shows the current migration state of a file.

STORE_DIRECTORY

The directory used to hold files for a MSP. See:

- "Disk msp Object" on page 250
- "Disk Cache Manager (DCM) msp Object" on page 254

tape block

See *block*.

tape drive

A hardware device that reads and writes data to tape.

tape chunk

See *chunk*.

tape merging

See *volume merging*.

task

A process initiated by the DMF event mechanism. Configuration tasks that allow certain recurring administrative duties to be automated are defined with configuration file parameters.

`taskgroup`

A type in the DMF configuration file for task groups. See "`dmdaemon` Object" on page 160.

`TASK_GROUPS`

The task groups containing tasks that the daemon or LS should run. See:

- "`dmdaemon` Object" on page 160
- "`services` Object" on page 167
- "`taskgroup` Object" on page 170
- "`filesystem` Object" on page 187
- "`libraryserver` Object" on page 215
- "`drivegroup` Object" on page 217
- "`volumegroup` Object" on page 224
- "FTP `msp` Object" on page 245
- "Disk `msp` Object" on page 250
- "Disk Cache Manager (DCM) `msp` Object" on page 254

TSREPORT_OPTIONS

Additional options that the `run_daily_tsreport.sh` script will add to the end of the `tsreport` command line. See "`taskgroup` Parameters" on page 174.

THRESHOLD

The percentage of active data on a tape. DMF will consider a tape to be sparse when it has less than this percentage of data that is still active. See "LS Tasks" on page 240.

TIMEOUT_FLUSH

The number of minutes after which the VG will flush files to tape. See "`volumegroup` Object" on page 224.

TMF_TMMNT_OPTIONS

Command options that should be added to the `tmmnt` command when mounting a tape. See:

- "`device` Object" on page 186
- "`drivegroup` Object" on page 217

TMP_DIR

The base pathname for DMF directories in which DMF puts temporary files such as pipes. See "`base` Object" on page 152.

TYPE

The required name for the object. See:

- "base Object" on page 152
- "dmdaemon Object" on page 160
- "node Object" on page 163
- "services Object" on page 167
- "taskgroup Object" on page 170
- "device Object" on page 186
- "filesystem Object" on page 187
- "policy Object" on page 193
- "libraryserver Object" on page 215
- "drivegroup Object" on page 217
- "volumegroup Object" on page 224
- "resourcescheduler Object" on page 230
- "resourcewatcher Object" on page 231
- "FTP msp Object" on page 245
- "Disk msp Object" on page 250
- "Disk Cache Manager (DCM) msp Object" on page 254

unmanaged filesystem

A POSIX filesystem (such as Lustre), configured in a filesystem object in the DMF configuration file, that is not managed by DMF but from which you can efficiently copy files to secondary storage via the dmarchive(1) command.

USE_UNIFIED_BUFFER

Determines how DMF manages its buffers when recalling files on this filesystem. See "filesystem Object" on page 187.

VERIFY_POSITION

A parameter that specifies whether the LS write child should (prior to writing) verify that the tape is correctly positioned and that the tape was properly terminated by the last use. See "drivegroup Object" on page 217.

VG

A volume group is a of component of an LS that is responsible for copying data blocks onto alternate media. Each VG contains a pool of tapes, all of the same media type, capable of managing single copies of user files. Multiple copies of the same user files require the use of multiple VGs. See also *LS*.

VOLUME_GROUPS

The VGs containing volumes that can be mounted on any of the drives within this drive group. See "drivegroup Object" on page 217.

unmigratable file

A file that the daemon will never select as a migration candidate.

unmigrate

See *recall*.

volume group

See *VG*

voided BFID-set state

A bit-file-identifier (BFID) set state that consists of one or more soft-deleted daemon database entries, either incomplete or complete. There is no user file.

voiding the BFID

The process of removing the bit-file identifier (BFID) from the user file inode and soft-deleting all associated database entries.

VOL record

An entry in the volume (VOL) table of the LS database that contains information about a tape volume. There is one VOL record for each volume. See also *CAT record*.

VOL table

A table in the LS database that contains VOL records. See also *CAT table*.

`volumegroup`

The configuration object that defines parameters relating to a pool of tape volumes mountable on the drives of a specific drive group that are capable of holding, at most, one copy of user files. See "`volumegroup` Object" on page 224.

VG database entry

The daemon database entry for a file that contains the path or key that is used to inform a particular VG where to locate the copy of the file's data.

`VOLUME_LIMIT`

The maximum number of tape volumes that can be selected for merging at one time. See "LS Tasks" on page 240.

volume merging

The mechanism provided by the LS for copying active data from volumes that contain largely obsolete data to volumes that contain mostly active data. Also known as *tape merging*.

`WATCHER`

The resource watcher that the LS should run. See "`libraryserver` Object" on page 215.

`WEIGHT`

The parameter that assigns a weighting to one or more VGs. See "`resourcescheduler` Object" on page 230.

`WRITE_CHECKSUM`

The parameter that specifies that tape block should be checksummed before writing. See:

- "`drivegroup` Object" on page 217
- "FTP `msp` Object" on page 245
- "Disk `msp` Object" on page 250
- "Disk Cache Manager (DCM) `msp` Object" on page 254

write child

A data mover process that migrates data to tape.

zone

A logical grouping of chunks. Zones are separated by file marks and are the smallest block-addressable unit on the tape volume. The target size of a zone is configurable by media type.

`ZONE_SIZE`

The parameter that specifies about how much data the write child should put in a zone. See "`volumegroup` Object" on page 224.

Index

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V