DMF Administrator's Guide for SGI® InfiniteStorage

007–3681–008

New Features

Note: This is the last release that will support the tape media-specific process (MSP). The tape MSP has been made obsolete by the library server (LS).

For information on changes in DMF functionality, including bugs fixed in recent releases, refer to the **Dependencies** and **News** buttons on the DMF installation interface (`dmmaint`(8)).

The major new feature of DMF 3.0 is the *disk cache manager (DCM)*, which provides *n*–tier capability. This new feature lets you configure the disk media-specific process (MSP) to manage data on secondary storage disk, allowing you to further migrate the data to tape as needed. The DCM provides an automated method of using economical disks of modest performance (such as serial ATA disks) as a fast-access DMF cache for files whose activity levels remain high, while also providing migration to tape for those files requiring less frequent access. See:

- "How DMF Works" on page 6
- "Disk Cache Manager (DCM) Commands" on page 23
- Table 2-1 on page 33
- "DCM Policies" on page 61
- "Setting Up Disk MSPs in DCM Mode" on page 105
- Table 2-4 on page 111
- "Disk MSP and Disk Cache Manager (DCM)" on page 173
- "Space Management and the Disk Cache Manager" on page 119

This guide also incorporates information from the obsolete *DMF Release and Installation Guide*. For installation instructions, see the `/CDROM/`*platform*`/DMF.Install` file on the CD-ROM for each operating system.

Record of Revision

Version	Description
2.6.1	October 1997 Original printing. Supports Data Migration Facility (DMF) 2.6.1 running under SGI IRIX systems.
2.6.2	August 1998 Supports DMF 2.6.2 running under SGI IRIX systems.
2.6.2.2	December 1998 Supports DMF 2.6.2.2 running under SGI IRIX systems.
004	May 2000 Supports DMF 2.6.3 running under SGI IRIX systems.
005	October 2000 Supports DMF 2.6.3.2 running under SGI IRIX systems. Only minor editing changes are included.
006	May 2002 Supports DMF 2.7 running under SGI IRIX systems.
007	June 2003 Supports DMF 2.8 running under SGI IRIX and Linux systems.
008	October 2003 Supports DMF 3.0 running under SGI IRIX and Linux systems.

Contents

About This Guide . . . . **xxiii**
Related Publications xxiii
Obtaining Publications xxiii
Conventions xxiv
Reader Comments xxiv

1. Introduction **1**
What Is DMF? 1
DMF Client and Server 3
Hardware and Software Requirements 3
DMAPI Requirement 5
Licensing Requirement 5
How DMF Works 6
Ensuring Data Integrity 11
DMF Architecture 12
Capacity and Overhead 13
DMF Administration 14
The User's View of DMF 17
DMF File Concepts and Terms 17
Migrating a File 18
Recalling a Migrated File 18
Command Overview 19
Configuration Commands 19
DMF Daemon and Related Commands 20
Space Management Commands 21

MSP/LS Commands 22
Disk Cache Manager (DCM) Commands 23
Commands for Other Utilities 23
2. Configuring DMF 25
Overview of the Configuration Steps 25
Configuration Considerations 26
Configuration File Requirements 27
Filesystem Mount Options 27
Mounting Service 28
Inode Size Configuration 28
Configuring Daemon Database Record Length 29
Interprocess Communication Parameters 31
Automated Maintenance Tasks 31
Using `dmmaint` To Install the License and Configure DMF 34
Overview of `dmmaint` 34
Installing the License, Reading News, and Defining the Configuration File 36
Configuration Objects 37
Configuring the Base Object 38
Configuring the DMF Daemon 42
Configuring Daemon Maintenance Tasks 44
Configuring Filesystems 51
DMF Policies 52
Automated Space Management Parameters 53
File Weighting and MSP or Volume Group Selection Parameters 55
Configuring Policies 57
DCM Policies 61
Setting Up Tape MSPs 62

MSP Objects 62
Device Objects 66
Device Objects for OpenVault As Mounting Service 68
Device Objects for TMF as Mounting Service 69
Setting Up Library Servers 70
Library Server Objects 71
Drive Group Objects 72
Volume Group Objects 77
Resource Scheduler Objects 80
Resource Watcher Objects 82
Example 82
Using OpenVault for Tape MSPs and LS Drive Groups 85
Using TMF tapes with Tape MSPs and LS Drive Groups 91
Configuring Maintenance Tasks for Tape MSP and LS 91
Library Server and MSP Database Records 94
Setting up FTP MSPs 96
Setting up Disk MSPs 101
Setting Up Disk MSPs in DCM Mode 105
Verifying the Configuration 108
Initializing DMF 108
General Message Log File Format 108
Parameter Table 110

3. Automated Space Management 115

Generating the Candidate List 116
Selection of Migration Candidates 116
Space Management and the Disk Cache Manager 119

Automated Space Management Log File 119

4. The DMF Daemon 121

Daemon Processing 121

DMF Daemon Database and `dmdadm` 122

`dmdadm` Directives 123

`dmdadm` Field and Format Keywords 125

`dmdadm` Text Field Order 128

Daemon Logs and Journals 129

5. The DMF Lock Manager 131

`dmlockmgr` Communication and Log Files 131

`dmlockmgr` Individual Transaction Log Files 133

6. Media-Specific Processes and Library Servers 135

Tape MSP and LS Operations 136

Tape MSP/LS Directories 137

Media Concepts 137

CAT Database Records 140

VOL Database Records 140

Tape MSP/LS Journals 141

Tape MSP/LS Logs 142

Volume Merging 145

`dmcatadm` Command 147

`dmcatadm` Directives 147

`dmcatadm` Keywords 150

`dmcatadm` Text Field Order 154

`dmvoladm` Command 155

`dmvoladm` Directives 155

dmvoladm Keywords . . . 157
dmvoladm Text Field Order . . . 165
dmatread Command . . . 166
dmatsnf Command . . . 167
dmaudit verifymsp Command . . . 168
FTP MSP . . . 168
Processing of Requests . . . 168
Activity Log . . . 169
Messages . . . 170
Disk MSP . . . 171
Processing of Requests . . . 171
Activity Log . . . 172
Disk MSP and Disk Cache Manager (DCM) . . . 173
Moving Migrated Data between MSPs and Volume Groups . . . 173
Converting from an IRIX DMF to a Linux DMF . . . 174
Converting from a Tape MSP to a Library Server . . . 178
Library Server Error Analysis and Avoidance . . . 181
Library Server Drive Scheduling . . . 183
Library Server Status Monitoring . . . 184

7. DMF Maintenance and Recovery . . . 187

Retaining Old DMF Daemon Log Files . . . 187
Retaining Old DMF Daemon Journal Files . . . 187
Soft- and Hard-deletes . . . 188
Using xfsdump and xfsrestore with Migrated Files . . . 189
Dumping and Restoring Files without the dump_tasks Object . . . 190
Filesystem Consistency with xfsrestore . . . 191
Using dmfill . . . 191

Database Recovery 192
Database Backups 192
Database Recovery Procedures 193

Appendix A. Messages 197
Message Format 197
Message Format for Catalog (CAT) Database and Daemon Database Comparisons . . 197
Message Format for Volume (VOL) Database and Catalog (CAT) Database and Daemon Database Comparisons 198
`dmcatadm` Message Interpretation 199
`dmvoladm` Message Interpretation 201

Appendix B. DMF User Library (`libdmfusr.so`) 203
Overview 203
Data Types 205
`DmuAllErrors_t` 205
`DmuByteRange_t` 205
`DmuByteRanges_t` 206
`DmuCompletion_t` 206
`DmuCopyRange_t` 207
`DmuCopyRanges_t` 207
`DmuErrHandler_f` 207
`DmuError_t` 208
`DmuErrorInfo_t` 208
`DmuFhandle_t` 208
`DmuFullstat_t` 208
`DmuReplyOrder_t` 208
`DmuReplyType_t` 209
`DmuReqid_t` 209

DmuRounding_t . . . 209
User-Accessible API Subroutines . . . 210
Context Manipulation Routines . . . 210
DmuCreateContext Subroutine . . . 210
DmuDestroyContext Subroutine . . . 211
DMF File Request Subroutines . . . 211
Copy File Requests . . . 212
Fullstat Requests . . . 213
Put File Requests . . . 215
Get File Requests . . . 217
Request Completion Subroutines . . . 219
DmuAwaitReplies Subroutine . . . 220
DmuGetNextReply Subroutine . . . 221
DmuGetThisReply Subroutine . . . 222
DmuFullstatCompletion Subroutine . . . 223
Memory Management Subroutines . . . 224

Appendix C. DMF Directory Structure Prior to Release 2.8 . . . 227

Appendix D. Differences from UNICOS DMF and UNICOS/mk DMF . . . 229

Glossary . . . 231

Index . . . 241

Figures

Figure 1-1 Application Data Flow 2
Figure 1-2 MSP Types 8
Figure 1-3 Library Server and Tape MSP Architecture 9
Figure 1-4 DMF Architecture 13
Figure 3-1 Relationship of Automated Space Management Targets 118
Figure 6-1 Media Concepts 139

Tables

Table 2-1 Automated Maintenance Task Summary 33

Table 2-2 NAME_FORMAT Strings 100

Table 2-3 DMF Log File Message Types 109

Table 2-4 Parameters for the DMF Configuration File 111

Table 5-1 dmlockmgr Token Files 132

Table D-1 Differences From UNICOS and UNICOS/mk 229

Examples

Example 6-1 Tape MSP Statistics Messages 143

Example 6-2 LS Statistics Messages 144

Example 6-3 `dmcatadm list` directive 153

Example 6-4 `dmvoladm list` directives 162

Example 6-5 Restoring Hard-deleted Files Using `dmatread` 167

Example 6-6 IRIX to Linux Conversion (Single Tape LS) 176

Example 6-7 IRIX to Linux Conversion (Two Tape MSPs) 176

Example 7-1 Database Recovery Example 194

Procedures

Procedure 2-1 Configuration Steps 25
Procedure 2-2 Daemon Database Record Length Configuration 29
Procedure 2-3 Running `dmmaint` 36
Procedure 2-4 Base Object Configuration 40
Procedure 2-5 Daemon Configuration 43
Procedure 2-6 Configuring the `daemon_tasks` Object 45
Procedure 2-7 Configuring the `dump_tasks` Object 48
Procedure 2-8 Configuring `filesystem` Objects 52
Procedure 2-9 Configuring Objects for Automated Space Management 57
Procedure 2-10 Configuring Objects for MSP or volume group Selection 59
Procedure 2-11 Configuring Tape MSPs 65
Procedure 2-12 Configuring Devices for TMF 70
Procedure 2-13 Configuring a Library Server and Its Components 83
Procedure 2-14 Configuring DMF to Use OpenVault 85
Procedure 2-15 Configuring the `msp_tasks` Object 92
Procedure 2-16 Creating MSP/LS Database Records 94
Procedure 2-17 Creating LS Database Records 95
Procedure 2-18 Configuring the `ftp` Object 101
Procedure 2-19 Configuring the `dsk` Object 105
Procedure 6-1 IRIX to Linux Conversion 174
Procedure 6-2 Tape MSP/LS Conversion 178
Procedure 7-1 Recovering the Databases 193

About This Guide

This publication documents administration of the Data Migration Facility (DMF) 3.0 on SGI Altix 3000 systems running the Linux operating system and other SGI systems running the IRIX operating system 6.5 and later releases.

Related Publications

The *DMF Recovery and Troubleshooting Guide for SGI InfiniteStorage* describes how to solve problems with DMF should you encounter them.

Also see the following files on the CD-ROM:

- `/CDROM/`*platform*`/DMF.Readme` contains general information about DMF
- `/CDROM/`*platform*`/DMF.News` contains a history of features and bug fixes provided with each DMF release
- `/CDROM/`*platform*`/DMF.Install` contains installation instructions

For example, for the Solaris platform see:

```
/CDROM/solaris/DMF.Readme
/CDROM/solaris/DMF.News
/CDROM/solaris/DMF.Install
```

Obtaining Publications

You can obtain SGI documentation in the following ways:

- 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, release notes, man pages, and other information.
- If it is installed on your SGI system, you can use InfoSearch, an online tool that provides a more limited set of online books, release notes, and man pages. With an IRIX system, select **Help** from the Toolchest, and then select **InfoSearch**. Or you can type `infosearch` on a command line.

- You can also view release notes by typing either `grelnotes` or `relnotes` on a command line.
- You can also view man pages by typing `man` *title* on a command line.

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

- Use the Feedback option on the Technical Publications Library Web page:

  http://docs.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:

  Technical Publications
  SGI
  1600 Amphitheatre Parkway, M/S 535
  Mountain View, California 94043–1351

- Send a fax to the attention of "Technical Publications" at +1 650 932 0801.

SGI values your comments and will respond to them promptly.

Chapter 1

Introduction

This chapter provides an overview of the Data Migration Facility (DMF) and its administration. It discusses the following:

- "What Is DMF?"
- "How DMF Works" on page 6
- "Ensuring Data Integrity" on page 11
- "DMF Architecture" on page 12
- "Capacity and Overhead" on page 13
- "DMF Administration" on page 14
- "The User's View of DMF" on page 17
- "DMF File Concepts and Terms" on page 17
- "Command Overview" on page 19

What Is DMF?

DMF is a hierarchical storage management system for SGI environments. DMF allows you to oversubscribe your online disk in a manner that is transparent to users; a user cannot determine, by using POSIX-compliant commands for filesystem enquiry, whether a file is online or offline. Only when special commands or command options are used can a file's actual residence be determined. This transparent migration is possible because DMF leaves inodes and directories intact within the native filesystem.

DMF automatically detects a drop below the filesystem free-space threshold and migrates selected data from expensive online disk to cheaper secondary storage, such as tapes. DMF automatically recalls the file data from offline media when the user accesses the file with normal operating system commands.

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

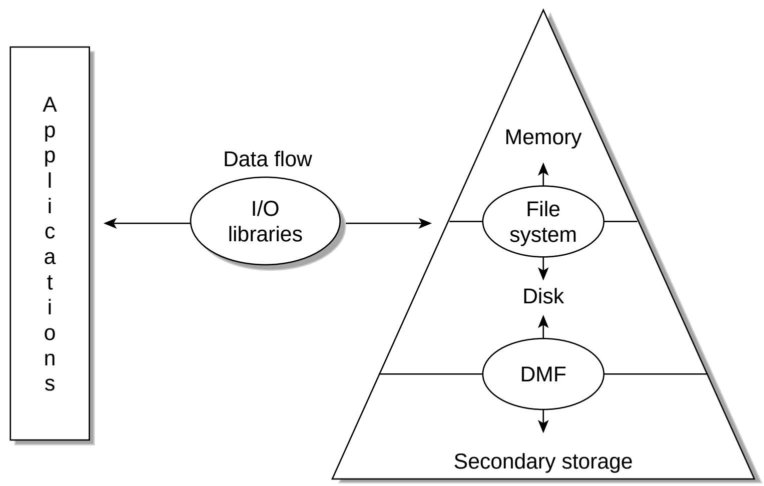

Figure 1-1 Application Data Flow

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's 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 file server.

DMF transports large volumes of data on behalf of many users. Because system interrupts and occasional storage device failures cannot be avoided, it is essential that the safety and integrity of data be verifiable. Therefore, DMF also provides tools necessary to validate your storage environment.

DMF has evolved around these customer requirements for scalability and the safety of data. As a filesystem migrator, DMF manages the capacity of online disk resources by transparently moving file data from disk to offline media. Most commonly, the secondary storage is tape, managed by OpenVault or the Tape Management Facility (TMF). However, the secondary storage can be any bulk-storage device accessible locally through NFS or FTP.

DMF Client and Server

DMF includes the following software subsystems:

- **Server**, which provides the full set of DMF functionality, including the DMF daemon, infrastructure, user and administrator commands, online manuals, and all man pages. This applies to IRIX systems and SGI ProPack for Linux 64-bit on SGI Altix 3000 systems.
- **Client**, which provides the limited set of user commands, libraries, and a subset of the man pages. This applies to all supported operating systems (see "Hardware and Software Requirements" on page 3).

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

Hardware and Software Requirements

The DMF server subsystem runs on the following:

- SGI ProPack 2.2.1 or later for Linux on SGI Altix 3000 systems
- IRIX 6.5.2 or later

The DMF client subsystem supports nodes running the following operating systems:

- SGI ProPack 2.2.1 or later for Linux
- IRIX 6.5.2 or later
- IBM AIX 5L version 5.1 ML 4 (64-bit kernel mode) APAR number IY42428

- Red Hat Linux on supported IA32 platforms:
  - Red Hat Linux 7.3
  - Red Hat Linux 8.0
  - Red Hat Linux 9
- Sun Microsystems Solaris:
  - Solaris 8 plus appropriate patch (see the release notes)
  - Solaris 9 plus appropriate patch
- Microsoft Windows:
  - Windows 2000 Service Pack 3 or Service Pack 4
  - Windows XP Service Pack 1

The client-only user commands are as follows:

```
dmattr
dmcopy
dmget
dmfind
dmls
dmput
```

The DMF `libdmfusr.so` user library lets you write your own custom DMF user commands that use the same application program interface (API) as the above DMF user commands.

The most commonly used devices on IRIX and Linux systems are DLT 4000/7000, SCSI versions of IBM 3590, and STK TimberLine and RedWood drives. All STK robots, Grau, and IBM 3494 are supported.

Note: On operating systems other than IRIX, the DMF client commands rely on the `xinetd` daemon to communicate with the DMF server machine. That communication is based on the `TCPMUX` functionality in `xinted`, which is not present in versions of `xinetd` prior to `xinetd`–2.3.11. If you want to export DMF–managed filesystems, the machine doing the exporting must run the appropriate level of `xinetd`.

The following releases contain the required version of `xinetd` and are supported for exporting DMF-managed filesystems:

- `xinetd` version 2.3.11 available with SGI ProPack 2.3
- `xinetd` 2.3.10 in Solaris

DMAPI Requirement

For filesystems to be managed by DMF, they must be mounted on the DMF server in order to enable the Data Management (DMAPI) interface. Do one of the following for each platform:

- IRIX:
  - Use the following command:

    ```
    mount -o dmi
    ```

  - Declare parameter 4 in the `fstab` entry to be `dmi`
- Linux:
  - Use the following command:

    `mount -o dmapi -o mtpt =` *mountpoint*

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

For more information, see the `mount` and `fstab` man pages.

Licensing Requirement

The software licensing used by DMF servers is based on the FLEXlm product from Macrovision Corporation. You must have a separate license for each DMF server. (No licensing is required on the client host.)

Software keys are used to enforce licensing. DMF licenses apply to a single specific system. DMF license fees vary depending on the amount of data being managed.

When you order DMF, you will receive an entitlement ID and the URL to a key generation webpage. You must submit the system host ID, hostname, and entitlement ID when requesting your permanent DMF license. To determine the hostname and host ID number: launch `dmmaint` and select **License Info**.

To obtain your permanent DMF license, follow the instructions on the key generation page. After the required information is provided, a key will be generated and displayed on the webpage along with installation instructions. You can use the **Update License** button on the `dmmaint` GUI to install the license.

For more information about licensing, see the following webpage:

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

How DMF Works

As a DMF administrator, you determine how disk space capacity is handled by selecting which filesystems DMF will manage and by specifying the volume of free space that will be maintained on each filesystem. Space management begins with a list of user files that are ranked according to criteria you define. 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 with all offline copies completed is called *fully migrated*. A file that is fully 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 can trigger file migration, or file owners can issue manual migration requests.

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

Note: Linux systems do not support the tape MSP. Tape support on Linux is available only via the LS.

The following types of MSPs are supported:

- Disk
- FTP
- MSP

In addition, you can configure the disk MSP to run in *disk cache manager (DCM)* mode for *n*–tier capability. 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 FTP MSP (`dmftpmsp`) uses the FTP protocol to transfer to and from disks of another system on the network. The disk MSP (`dmdskmsp`) is similar, but uses a filesystem mounted on the DMF server itself. This can be a local filesystem or a remote one mounted through NFS or similar filesharing protocol. If the disk MSP is configured as a DCM, the filesystem used by the DCM must be a local XFS filesystem.

Most commonly, the offline media is magnetic tape, usually in a tape library (also known as a *robotic library* or *silo*). DMF has two tape components: the tape MSP (`dmatmsp`) and the library server (`dmatls`).

Note: The tape MSP has limitations in some environments and has been superseded by the newer LS.

Figure 1-2 summarizes the various MSP types.

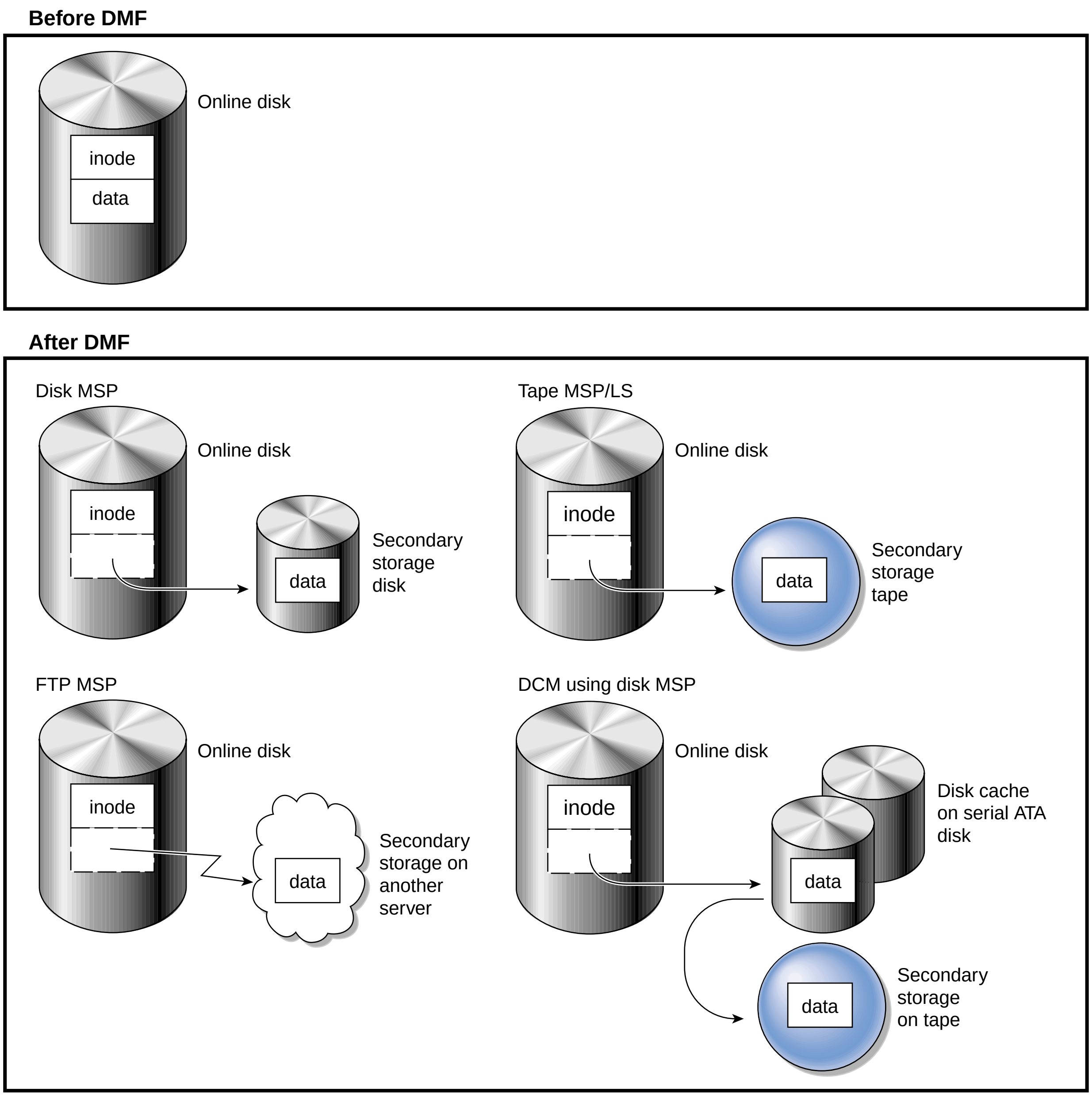

Figure 1-2 MSP Types

Figure 1-3 shows the architecture of the LS and tape MSP.

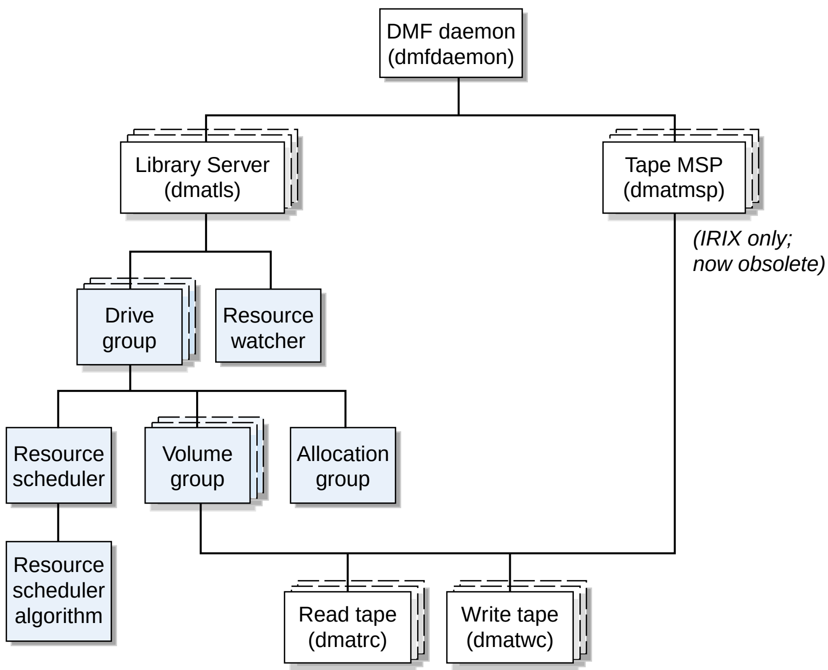

Figure 1-3 Library Server and Tape MSP Architecture

There is one LS process (`dmatls`) per tape library, which maintains a pair of databases that all of its components share. The entities in the shaded boxes in Figure 1-3 on page 9 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 one tape library. These drives can be used by multiple volume groups (see volume groups below) and by non-LS users, such as MSPs, and 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 task of the drive group is to monitor tape I/O for errors, attempt to classify them as volume, drive, or mounting service problems, and to take preventive action.
Volume group	The volume group 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 volume group, used to hold a communal pool of empty tapes. These tapes can be transferred to a volume group 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 volume groups to exceed the number available. The purpose of the resource scheduler is to decide which volume groups 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.
Resource scheduler algorithm	Given the wide variety of site requirements, sites can write their own scheduling routines in C++. These routines are packaged in a dynamically-loadable Dynamic Shared Object library (DSO or `.so` file). When loaded, these routines are an internal component of the `dmatls` process. In the absence of a site-supplied algorithm, 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. The main format is HTML files viewable by a web browser, but text files

designed for use by `awk` or `perl` scripts are also maintained.

In contrast to the LS process, each tape MSP has its own database of tape volumes it controls and the user files (at most one copy of each) that they contain. It is somewhat similar to the volume group previously described. Tape MSPs refer to a "device object," which controls a group of tape drives in a similar, but less flexible, way as the drive group previously described. A site can use any combination of the various MSPs or LSs; they are not mutually exclusive. **Exception:** The tape MSP is not supported on Linux systems.

The `dmatrc` and `dmatwc`. These processes are called the read- and write-children, and are created by MSPs and volume groups 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. For details, see "Hardware and Software Requirements" on page 3.

Ensuring Data Integrity

DMF provides capabilities ensure the integrity of offline data. For example, you can have multiple MSPs or volume groups 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. (You can also convert other file servers to IRIX or Linux file servers running DMF.) 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 tape MSP or 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 database provides for full transaction journaling and employs two-phase commit technology. The combination of these two features ensures that DMF applies only whole transactions to its database. 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.

DMF Architecture

DMF consists of the DMF daemon and one or more MSPs or LSs. The DMF daemon accepts requests from the DMF administrator or from users to migrate filesystem data, 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 MSP/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 kernel interface defined by the Data Management Interface Group (DMIG). DMAPI is also supported by X/Open, where it is evolving as the XDSM standard.

Figure 1-4 illustrates the DMF architecture.

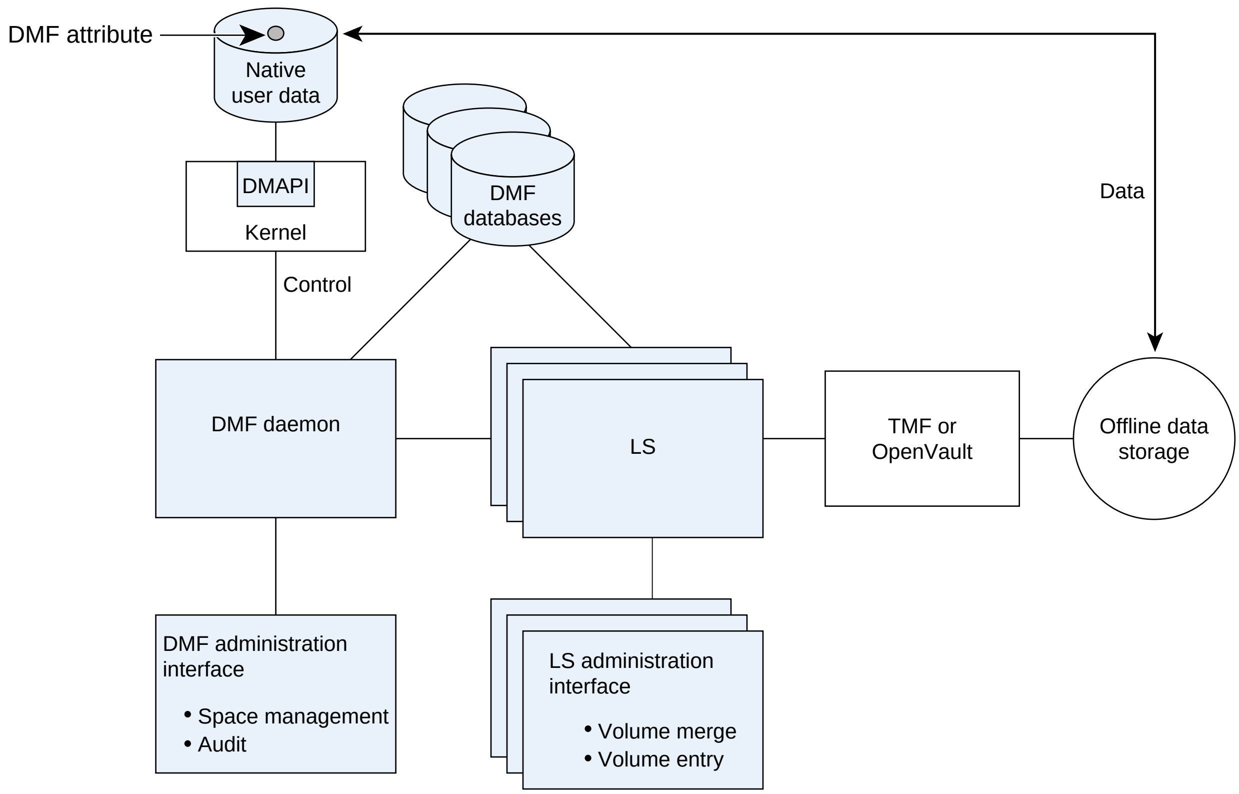

Figure 1-4 DMF Architecture

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 DMF 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

DMF can be configured for a variety of environments including the following:

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

DMF manages two primary resources: pools of offline media and free space on native filesystems.

As a DMF administrator, you must characterize and determine the size of 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 and the native filesystem. You should select autoloaders and media transports that are suitable for the data volume and delivery rates you anticipate.

Beyond initial planning and setup, DMF requires that you perform recurring administrative duties. DMF allows you to configure tasks that automate these duties. A *task* is a cron-like process initiated on a time schedule you determine. Configuration tasks are defined with configuration file parameters. The tasks are described in detail in "Configuring Daemon Maintenance Tasks" on page 44, and "Configuring Maintenance Tasks for Tape MSP and LS" on page 91.

DMF requires administrative duties to be performed in the following areas:

- File ranking. You must decide which files are most important as migration candidates. When DMF migrates and frees files, it chooses files based on criteria you chose. The ordered list of files is called the DMF *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 116.

- Automated 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 3, "Automated Space Management" on page 115.

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

  Complex strategies are possible when using multiple MSPs, LSs, or volume groups. 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 volume group selection. When setting up a tape MSP or volume group, you assign a pool of tapes for use by that MSP/volume group. The `dmvoladm`(8) utility provides management of the tape MSP/LS 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. Two utilities are provided that support management of failing media. The `dmatsnf`(8) utility is used to scan a DMF volume for flaws, and `dmatread`(8) is used for recovering data. Additionally, the volume merge process built into the MSP/LS is capable of effectively recovering data from failed media.

  Chapter 6, "Media-Specific Processes and Library Servers" on page 135, provides more information on administration.

- Integrity and reliability. Integrity of data is a central concern to the DMF administrator. You must understand and monitor processes in order to achieve the highest levels of data integrity, as follows:

  – Even though you are running DMF, you must still run backups because DMF moves only the data associated with files, not the file inodes or directories. You can configure DMF to automatically run backups of your DMF-managed filesystems.

    The `xfsdump` and `xfsrestore` utilities understand when a file is fully migrated. The `xfsdump` utility has an option that allows for dumping only files that are not fully 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.

  – DMF databases record all information about stored data. The DMF databases must be synchronized with the filesystems DMF manages. Much of the work done by DMF ensures that the DMF databases remain aligned with the filesystems.

    You can configure DMF to automatically examine the consistency and integrity of the DMF daemon and MSP/LS databases. 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 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. 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` directory from overflowing.

You can configure the `run_hard_delete.sh` task to automatically perform hard-deletes, which are described in "Recalling a Migrated File" on page 18.

The User’s View of DMF

While the administrator has access to a wide variety of commands for controlling DMF, the end user sees very little. Migrated files remain cataloged in their original directories and are accessed as if they were still on online disk. The only difference users might notice is a delay in access time.

However, commands are provided for file owners to affect the manual storing and retrieval of data. Users can do the following:

- Explicitly migrate files by using the `dmput`(1) command
- Explicitly recall files by using the `dmget`(1) command
- Copy all or part of the data from a migrated file to an online file by using the `dmcopy`(1) command
- Determine whether a file is migrated by using the `dmfind`(1) or `dmls`(1) commands
- Test in shell scripts whether a file is online or offline by using the `dmattr`(1) command

DMF File Concepts and Terms

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* can be either of the following:
  - *Dual-state files* are files whose data resides both on online disk and on secondary storage
  - *Offline files* are files whose data is no longer on online disk
  - *Unmigrating files* are previously offline files in the process of being recalled to online disk

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

Like a regular file, a migrated file has an inode. Only an offline file requires the intervention of the DMF daemon to access its 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.

Migrating a File

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

The DMF daemon keeps a record of all migrated files in its 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.

When the daemon receives a request to migrate a file, it adjusts the state of the file, ensures that the necessary MSPs or volume groups are active, and sends a request to the MSPs or volume groups. MSPs or volume groups then copy data to the offline storage media.

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

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 volume group from its internal list and sends that MSP/volume group a request to recall a copy of the file. If more than one MSP or volume group 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 bit file identifier (BFID) is soft-deleted. A file is *soft-deleted* when 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 physically removed from the DMF database. 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 physically deleted (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 volume groups so that copies of the file can be removed from media. For a tape MSP/volume group, this involves compression (or merging).

Command Overview

The following section provides definitions for administrator commands grouped by function.

Configuration Commands

The 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.

For information about editing the configuration file, see Chapter 2, "Configuring DMF" on page 25. The following man pages are 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, dmfdaemon 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 4, "The DMF Daemon" on page 121, for a detailed description of the DMF daemon.

Caution: If used improperly, commands that make changes to the DMF 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.
dmfdaemon(8)	Starts the DMF daemon. The preferred method is to use the /etc/init.d/dmf script.
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 MSP/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.
`dmdbrecover`(8)	Updates the daemon and tape MSP/LS databases with journal entries.
`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.
`dmhdelete`(8)	Deletes expired daemon database entries and releases corresponding MSP or volume group 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 DMF daemon and the MSP/LS databases to a specified location. If you configure the `run_copy_database.sh` task, this command is executed automatically as part of the task.
`dmversion`(8)	Reports the version of DMF that is currently executing.

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 is enabled (as `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 3, "Automated Space Management" on page 115, for details.

MSP/LS Commands

The DMF tape MSP and LS maintain a database that contains volume (VOL) records and catalog (CAT) records. VOL records contain information about tape volumes, and CAT records contain information about offline copies of migrated files.

The disk and FTP MSPs allow the use of local or remote disk storage for storing migrated data. They use no special commands, utilities, or databases. For more information, see "Disk MSP" on page 171, and "FTP MSP" on page 168.

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

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

Most data transfers to and from tape media are performed by components internal to the MSP/LS. However, there are also two utilities that can read tape MSP/LS volumes directly:

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

There are also tools that check for MSP database inconsistencies:

Command	Description
`dmatvfy`(8)	Verifies the MSP/LS database contents against the `dmfdaemon`(8) 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.

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.

Commands for Other Utilities

The following utilities 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 DMF daemon database.
`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 database, 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/volume groups.
`dmmaint`(8)	Performs DMF maintenance and provides interfaces for licensing and initial configuration.
`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 an MSP/volume group. The selected tapes are imported into the OpenVault database, assigned to the DMF application, and added to the MSP's/LS's database.
`dmov_makecarts`(8)	Makes the tapes in one or more MSP/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.
`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.
`dmxfsrestore`(8)	Calls the `xfsrestore`(1M) command to restore files dumped to tape volumes that were produced by DMF administrative maintenance scripts.

Configuring DMF

This chapter describes how to configure DMF, verify the configuration, and perform some periodic maintenance tasks:

- "Overview of the Configuration Steps"
- "Configuration Considerations" on page 26
- "Using `dmmaint` To Install the License and Configure DMF" on page 34
- "Configuration Objects" on page 37
- "Verifying the Configuration" on page 108
- "Initializing DMF" on page 108
- "General Message Log File Format" on page 108
- "Parameter Table" on page 110

Overview of the Configuration Steps

To configure DMF, you will perform the following steps. Before starting, read "Configuration Considerations" on page 26.

Procedure 2-1 Configuration Steps

1. Install DMF according to the platform-specific instructions in the `/CDROM/`*platform*`/DMF.Install` file.
2. Determine how you want to complete periodic maintenance tasks. See "Automated Maintenance Tasks" on page 31.
3. Invoke `dmmaint`(8) (see "Overview of `dmmaint`" on page 34) to do the following:
   a. Install the FLEXlm license on each DMF server. (DMF clients do not require a license.)
   b. Create or modify your configuration file and define the following objects:
      - Base object

- Daemon object
- Daemon maintenance tasks
- Automated space management
- Media-specific process (MSP) or library server (LS)

You must also define the object for MSP/LS maintenance tasks, set up the MSPs and/or LSs, and configure your mounting service. See "Configuration Objects" on page 37.

4. Verify the configuration by clicking the **Inspect** button, which runs the `dmcheck`(8) script. See "Verifying the Configuration" on page 108.

   If there are errors, fix them by clicking the **Configure** button to edit the configuration file. Repeat these steps until there are no errors.

5. Start DMF. See "Initializing DMF" on page 108.

Configuration Considerations

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

- "Configuration File Requirements"
- "Filesystem Mount Options" on page 27
- "Mounting Service" on page 28
- "Inode Size Configuration" on page 28
- "Configuring Daemon Database Record Length" on page 29
- "Interprocess Communication Parameters" on page 31
- "Automated Maintenance Tasks" on page 31

Configuration File Requirements

The DMF server uses a set of pathnames in which it stores databases, log and journal files, and temporary file directories. These filesystems have the following requirements:

- HOME_DIR is the base pathname for DMF directories in which databases reside. It must be a separate filesystem.
- JOURNAL_DIR is the base pathname for DMF directories in which the daemon and tape MSP/LS database journal files reside. It must be a separate filesystem on a different disk from HOME_DIR.
- SPOOL_DIR is the base pathname used to construct the directory names for DMF directories in which DMF log files reside. It must be a separate filesystem.
- TMP_DIR is the base pathname used to construct the directory names for DMF directories in which DMF puts temporary files such as pipes. It should exist, but does not necessarily need to be a separate filesystem.
- MOVE_FS is the base pathname for the scratch filesystem used to move files between MSPs or volume groups. This is a requirement only if you configure more than one MSP or volume group. If you have more than one MSP or volume group, MOVE_FS must be a separate filesystem, and it must be mounted to enable the Data Management (DMAPI) interface.

All of these configuration requirements are checked by the dmcheck(8) command, which can be invoked with the dmmaint GUI's **Inspect** button.

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 as listed in the files accessed by the **News** button on the dmmaint(8) GUI..

Caution: For filesystems to be managed by DMF, they must be mounted to enable the DMAPI interface. Failure to enable DMAPI for DMF-managed filesystems will result in a configuration error. See "DMAPI Requirement" on page 5.

Mounting Service

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

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

Inode Size Configuration

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.

SGI recommends that you configure your filesystems so that the extended attribute is always inode-resident by using the IRIX `mkfs_xfs` command or the Linux `mkfs.xfs` command. Declare the inode size to be 512 bytes (`-i size=512`). Filesystems that already exist must be dumped, recreated, and restored.

Configuring Daemon Database Record Length

A daemon database entry is composed of one or more fixed-length records: a base record (`dbrec`) and zero or more path segment extension (`pathseg`) records. The `dbrec` consists of several fields, including the `path` field.

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 `dsk` MSP definition) can fit into the `path` field of the daemon's `dbrec` record, DMF does not require `pathseg` records. If the MSP 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 `dmatmsp`, `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.

The advantage of having very few `pathseg` records lies in increased efficiency for retrieving daemon database records. There is no need to access the `pathseg` key and data files to retrieve a complete daemon database record.

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 2-2 Daemon Database Record Length Configuration

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

1. If the `dmfdaemon` is running, use the following command to halt processing:

   ```
   /etc/init.d/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 database
- *backup_dir* is the name of the directory that will hold the old version of the database

3. Changed 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. For the most efficient use of disk space for the `dmatmsp`, set the `dbrec path` size to 26.

6. Regenerate the new database definition, as follows:

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

7. Backup 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:

   ```
   /etc/init.d/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:

- IRIX:
  - `MSGMAX`
  - `MSGMNI`
  - `MSGSEG`
  - `MSGSSZ`

  For more information, see *IRIX Admin: System Configuration and Operation* and the `msgop`(2) man page.

- Linux:
  - `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
- Running hard deletes
- Running `dmaudit`(8)
- Monitoring the status of tapes in tape MSPs and LSs

- 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 through the use of `TASK_GROUPS` parameters for the DMF daemon and the tape MSP/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 for you in the `/usr/lib/dmf` directory.

The automated tasks are described in "Configuring Daemon Maintenance Tasks" on page 44, for the daemon tasks and in "Configuring Maintenance Tasks for Tape MSP and LS" on page 91, for the tape MSP.

Table 2-1provides a summary of the automated maintenance tasks.

Table 2-1 Automated Maintenance Task Summary

Object Type	Task	Purpose	Parameters
Daemon	run_audit	Audit databases	
	run_copy_databases	Backup DMF databases	DATABASE_COPIES
	run_full_dump	Full backup of filesystems For restores, see dmxfsrestore(8)	DUMP_DEVICE DUMP_INVENTORY_COPY DUMP_FILE_SYSTEMS DUMP_MIGRATE_FIRST DUMP_RETENTION DUMP_VSNS_USED DUMP_TAPES
	run_hard_deletes	Hard-delete files	Uses DUMP_RETENTION
	run_partial_dump	Partial backup of filesystem(s)	Uses parameters set for run_full_dump
	run_remove_journals	Remove old journal files	JOURNAL_RETENTION
	run_remove_logs	Remove old log files	LOG_RETENTION
	run_scan_logs	Scan log files for errors	
MSP/LS	run_compact_tape_report	Create tape reports	
	run_merge_stop	Stop tape merges	
	run_tape_merge	Merge sparse tapes	DATA_LIMIT THRESHOLD VOLUME_LIMIT
	run_tape_report	Create tape reports	
LS	run_merge_mgr	Merge sparse tapes	DATA_LIMIT THRESHOLD VOLUME_LIMIT
DCM-mode disk MSP	run_dcm_admin	Routine disk cache manager (DCM) administration	

Using dmmaint To Install the License and Configure DMF

On DMF servers, you can use the dmmaint utility to view DMF release-specific news and to view information related to the dependencies you should be aware of before you start DMF. (You can also view these files directly from the CD-ROM by using an editor such as vi on the /CDROM/*platform*/DMF.News and /CDROM/*platform*/DMF.Readme files.)

You can also 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.

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 processes (MSP) or the library servers (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 2, "Configuring DMF" on page 25, 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.

Dependencies Displays the dependencies file, which contains information such as supported releases, patch requirements, and so on. The file is installed on the server platform in the following file:

- IRIX: `/usr/relnotes/dmf/ch1.z`
- Linux: `/usr/share/doc/dmf-`*version_number*`/Readme`

News Displays the news file, which contains information such as new DMF features, changes in the products, descriptions of fixed bugs, and future product plans. The file is installed on the server platform in the following file:

- IRIX: `/usr/relnotes/dmf/ch2.z`

- Linux:
  `/usr/share/doc/dmf-`*version_number*`/news`

License Info
: Displays the host name and FLEXlm host ID (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 FLEXlm 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 "Licensing Requirement" on page 5.

Installing the License, Reading News, and Defining the Configuration File

The following procedure uses `dmmaint` to complete the initial configuration of DMF:

Procedure 2-3 Running `dmmaint`

1. Select **Dependencies** to read about all the hardware and software requirements that must be fulfilled before running DMF.

2. Select **News** to read about what is new with this revision of DMF.

3. 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.

4. Select **Configure** to edit the configuration file. The first time that you select this button, `dmmaint` will prompt you for the file you want to use as a basis for the configuration. Choose to use your existing configuration file or one of the sample files provided. If you choose to use your existing configuration, you may need to add new parameters to implement new features.

   If a configuration file exists, a window appears that asks if you would like to modify the existing configuration 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.

   `dmmaint` then opens an editing window containing the configuration file, allowing you to modify the configuration to suit your needs.

When you exit the window, `dmmaint` will ask if you want to make your changes permanent. If so, click `OK`.

5. Click the **Inspect** button, which runs `dmcheck` to report any errors in that 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.

6. If you do not want DMF to be automatically started and stopped, enter the following command (you must be running as `root`):

```
chkconfig dmf off
```

For information about how to start and stop DMF, see the `dmfdaemon`(8) and `dmdstop`(8) man pages.

Configuration Objects

The configuration file consists of configuration objects and parameters. The file uses the following types of configuration objects:

- The *base object*, which defines pathname and file size parameters necessary for DMF operation
- The *daemon object*, which defines parameters necessary for `dmfdaemon`(8) operation
- The *filesystem object*, which defines parameters necessary for migrating files in that filesystem
- The *policy objects*, which specify parameters to determine MSP or volume group selection, automated space-management policies, and/or file weight calculations in automatic space management
- The *MSP objects*, which define parameters necessary for that MSP's operation
- The *device objects*, which define parameters for the MSP's use of tape devices
- The *taskgroup objects*, which define parameters necessary for automatic completion of specific maintenance tasks
- The *library server (LS) object*, which defines parameters relating to a tape library
- The *drive group object*, which defines parameters relating to a pool of tape devices in a specific library

- The *volume group object*, which defines parameters relating to a pool of tape volumes mountable on the drives of a specific drive group, capable of holding, at most, one copy of user files
- The *resource scheduler object*, which defines parameters relating to scheduling of tape devices in a drive group when requests from volume groups exceed the number of devices available
- The *resource watcher object*, which defines parameters relating to the production of files informing the administrator about the status of the LS and its components

DMF configuration objects and parameters are also defined in the `dmf.conf`(5) man page and in Table 2-4 on page 111.

Each object is configured by a sequence of lines called a *configuration stanza*. These have the following general form:

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

For filesystems, *object_name* is the mount point. Otherwise, it is chosen by the administrator. *object_type* identifies the type (detailed in the following subsections). The parameters and their values depend on the type of the object. These stanzas are case-sensitive and 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. Before placing a new configuration into production, it is important to check it by running `dmcheck`(8).

Configuring the Base Object

The base configuration 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.

Parameter	Description
`TYPE`	`base` (required name for this type of object)

`ADMIN_EMAIL`
: E-mail address to receive output from administrative tasks. The mail can include errors, warnings, and output from any configured tasks. You can specify a list of addresses, separated by spaces.

`HOME_DIR`
: Base pathname used to construct directory names for DMF directories in which databases and related files reside. The value of this parameter is generally referred to as *HOME_DIR*.

`JOURNAL_DIR`
: Base pathname used to construct directory names for DMF directories in which the daemon and tape MSP/LS database journal files will be written. To provide the best chance for database recovery, this directory should be a separate filesystem and a different physical device from *HOME_DIR*. The value of this parameter is generally referred to as *JOURNAL_DIR*.

`JOURNAL_SIZE`
: Maximum size (in bytes) of the database journal file before DMF closes it and starts a new file.

`LICENSE_FILE`
: Full pathname of the file containing the FLEXlm license used by DMF. The default is as follows:

- IRIX: `/var/flexlm/license.dat`
- Linux: `/etc/flexlm/license.dat`

There is no need to use this parameter if the default is being used.

`OV_KEY_FILE`
: File containing the OpenVault keys used by DMF. It is usually located in *HOME_DIR* and called `ovkeys`. There is no default. (Use this parameter only if you are using OpenVault as your tape mounting service.)

`OV_SERVER`
: Name returned by the `hostname`(1) command on the machine on which the OpenVault server is running. This parameter only applies when OpenVault is used as the mounting service. The default value is the host name of the machine on which you are running.

`SPOOL_DIR`
: Base pathname used to construct the directory names for DMF directories in which DMF log files are kept. The value of this parameter is generally referred to as *SPOOL_DIR*.

`TMP_DIR`	Base pathname used to construct the directory names for DMF directories in which DMF puts temporary files such as pipes. It is also used by scripts for temporary files and is the directory used by default by the tape MSP for caching files if the `CACHE_DIR` parameter is not defined. The value of this parameter is generally referred to as *TMP_DIR*.

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

If you intend to run the OpenVault library management facility as the mounting service for DMF, you must configure the `OV_KEY_FILE` and `OV_SERVER` parameters. If you are running a different mounting service, you do not need these parameters. More configuration steps are necessary to configure DMF to use OpenVault; see "Using OpenVault for Tape MSPs and LS Drive Groups" on page 85.

Procedure 2-4 Base Object Configuration

The following example defines a base object:

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

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. dmcheck(8) reports an error if any of the HOME_DIR, SPOOL_DIR, or JOURNAL_DIR parameters are also configured as DMF-managed filesystems. Configure the daemon_tasks object to manage old log files and journal files in these directories (you can change the namedaemon_tasks to be anything you prefer). See "Configuring Daemon Maintenance Tasks" on page 44, for more information.

The following steps explain pertinent information for configuring the base object:

1. Ensure that TYPE is set to base.
2. Configure the e-mail address specified by the ADMIN_EMAIL parameter to be the user to whom you want to send the output of the configured tasks described in "Automated Maintenance Tasks" on page 31.
3. Configure the filesystem specified by the HOME_DIR configuration parameter (referred to as *HOME_DIR*) as a separate filesystem, and restrict its contents to DMF databases and relatively static files such as DMF scripts.

   DMF cannot run if *HOME_DIR* runs out of space, and such an event is more likely to happen if it is simply another directory in /usr.
4. Set TMP_DIR to be any filesystem that can store temporary files. /tmp or a directory below /tmp is a common choice.
5. Configure the log file directory (referred to as *SPOOL_DIR*) as a separate filesystem so that log file growth does not impact the rest of the system.
6. Ensure that the journal file directory (referred to as *JOURNAL_DIR*) resides on a physical device completely separate from the one on which *HOME_DIR* resides. Backup copies of DMF databases should also be stored on the *JOURNAL_DIR* filesystem.
7. Configure the JOURNAL_SIZE parameter to be the maximum size allowable for a journal file before DMF closes it.
8. If you plan to run OpenVault, do the following:
   - Configure the OV_KEY_FILE parameter to be the name of the key file that holds security information for OpenVault.

- Configure the OV_SERVER parameter to the name of the server that runs OpenVault.

For more information, see Procedure 2-14, page 85.

Configuring the DMF Daemon

The daemon 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) **Note:** This cannot be specified as dmfdaemon. It must be dmdaemon.
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 "General Message Log File Format" on page 108.
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(1) or dmmigrate(8) only) • auto (automated space management) The default is auto.
MOVE_FS	Names the scratch filesystem used by dmmove(8) to move files between MSPs or volume groups. There is no default. Necessary only if you wish to use dmmove.

LS_NAMES or MSP_NAMES	Names the MSPs or LSs used by the DMF daemon. You must specify either LS_NAMES or MSP_NAMES (you cannot specify both.) There is no default. The order of the values specified for this parameter is integral to the determination of the MSP or volume group 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 LS object's DRIVE_GROUPS parameter. It then searches for the copy in volume group order from the drivegroup object's VOLUME_GROUPS parameter.
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 "Configuring Daemon Maintenance Tasks" on page 44. SGI recommends that you use the task groups specified in the sample configuration file, changing the parameters as necessary for your site.

Procedure 2-5 Daemon Configuration

The following example defines a daemon object:

```
define  daemon
        TYPE                    dmdaemon
        MOVE_FS                 /move_fs
        MIGRATION_LEVEL         auto
        MSP_NAMES               cart1 cart2
        TASK_GROUPS             daemon_tasks dump_tasks
enddef
```

The following steps explain pertinent information for configuring the daemon object:

1. Ensure that TYPE is set to dmdaemon. There is no default.

Note: This cannot be set to dmfdaemon. It must be dmdaemon.

---

2. If you have more than one MSP or volume group, ensure that the MOVE_FS parameter is set to a filesystem that can accept temporary files. This must be the root of a DMAPI filesystem. There is no default.

3. The MIGRATION_LEVEL parameter determines the level of service for migration **to** offline media. Migration **from** offline media (either automatic or manual recall) is not affected by the value of MIGRATION_LEVEL.

   Configure MIGRATION_LEVEL to be one of the following:

   - none (no migration will take place on any DMF filesystem)
   - user (users/administrators can perform dmput(1) or dmmigrate(8) commands and no other migration will take place)
   - auto (automated space management on at least one DMF filesystem)

   This value is the highest level you want to allow anywhere in your DMF environment. The default is auto. See "DMF Policies" on page 52, for information about configuring automated space management.

4. Configure LS_NAMES or MSP_NAMES to be the names of the MSPs or LSs to be used by this daemon. You will use these names when defining the MSP/LS objects and, for MSPs only, in SELECT_MSP parameters within policies. See Procedure 2-11, page 65. You must specify a value for LS_NAMES or MSP_NAMES (but not both); there is no default.

5. Configure the TASK_GROUPS parameter to the names of the objects used to define how periodic maintenance tasks are completed. 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. You can change the object names themselves (dump_tasks and daemon_tasks) to be any name you like. There is no default value for the object. See "Configuring Daemon Maintenance Tasks" for more information.

Configuring Daemon Maintenance Tasks

You can configure daemon_tasks parameters to manage how the DMF daemon performs the following maintenance tasks:

- Auditing databases (the `run_audit.sh` task)
- Scanning recent log files for errors (the `run_scan_logs.sh` task)
- Removing old log files (the `run_remove_logs.sh` task and the `LOG_RETENTION` parameter)
- Removing old journal files (the `run_remove_journals.sh` task and the `JOURNAL_RETENTION` parameter)
- Backing up DMF databases (the `run_copy_databases.sh` task and the `DATABASE_COPIES` parameter)

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

You can configure `dump_tasks` parameters to manage how the daemon completes the following tasks to back up the DMF-managed filesystems:

- Fully backing up DMF-managed filesystems (the `run_full_dump.sh` task)
- Partially backing up DMF-managed filesystems (the `run_partial_dump.sh` task)
- Hard-deleting files no longer on backup tape (the `run_hard_deletes.sh` task)
- Managing 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, and locations for inventory files. Table 2-1 on page 33, provides a summary of automated maintenance tasks.

Procedure 2-6 Configuring the `daemon_tasks` Object

The following steps explain how to define a `daemon_tasks` object. You can change the object name itself (`daemon_tasks`) to be any name you like.

Do not change the script names.

You may comment out the `RUN_TASK` parameters for any tasks you do not want to run.

The following example configures a `daemon_tasks` object:

```
define daemon_tasks
        TYPE                taskgroup
        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
        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
```

1. Define the object to have the same name that you provided for the `TASK_GROUPS` parameter of the `daemon` object. In the example it is `daemon_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 actual directory containing auxiliary programs and scripts (that is, `/usr/lib/dmf`). 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 `daemon_tasks`) as the second parameter. The task itself may use the `dmconfig`(8) command to obtain further parameters from either of these objects.

   All of the `RUN_TASK` parameters require that you provide a *time_expression*.

   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
```

Some of the tasks defined by the `RUN_TASK` parameters require more information. The following steps specify what you must provide:

a. The `run_audit.sh` task runs `dmaudit`. For this task, provide a *time_expression*. 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 "Configuring the Base Object" on page 38).

b. The `run_scan_logs.sh` task scans the DMF log files for errors. For this task, provide a *time_expression*. 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.

c. 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. Valid values for `LOG_RETENTION` are a number followed by `m[inutes]`, `h[ours]`, `d[ays]`, or `w[eeks]`.

   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. Valid values for `JOURNAL_RETENTION` are a number followed by `m[inutes]`, `h[ours]`, `d[ays]`, or `w[eeks]`.

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.

---

d. The `run_copy_databases.sh` task makes a copy of the DMF databases. For this task, in addition to a value for *time_expression*, provide a value for the `DATABASE_COPIES` parameter that specifies one or more directories. If you specify multiple directories, breaking the directories among multiple disk devices minimizes the chance of losing all the copies of the database.

The task copies a snapshot of the current DMF databases to the directory with the oldest copy. 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 e-mail address defined by the `ADMIN_EMAIL` parameter of the `base` object.

Procedure 2-7 Configuring the `dump_tasks` Object

The following steps explain how to define a `dump_tasks` object. You can change the object name itself (`dump_tasks`) to be any name you like.

Do not change the script names.

You may comment out the `RUN_TASK` parameters for any tasks you do not want to run.

The following example would configure a `dump_tasks` object:

```
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
```

1. Define the object to have the same name that you provided for the `TASK_GROUPS` parameter of the `daemon` object. In the example it is `dump_tasks`.

2. Ensure that `TYPE` is set to `taskgroup`. There is no default.

3. Configure the `RUN_TASK` parameters. See step 3 in Procedure 2-6, page 45, for information about `$ADMINDIR` and *time_expression*.

The following steps specify the information you must provide for the tasks to run correctly.

a. The `run_full_dump.sh` task runs a full backup of DMF-managed filesystems at intervals specified by the *time_expression*. In the example, the full backup is run each week on Sunday morning one minute after midnight.

b. 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.

c. The `run_hard_deletes.sh` task removes from the database any files that have been deleted but can no longer be restored because the backup tapes have been recycled (that is, it 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 188.

d. Manage the data from the filesystem dumps by configuring the following parameters:

```
DUMP_TAPES
DUMP_RETENTION
DUMP_DEVICE
DUMP_MIGRATE_FIRST
DUMP_INVENTORY_COPY
DUMP_FILE_SYSTEMS
DUMP_VSNS_USED
```

The `DUMP_TAPES` parameter specifies the path of a file that contains tape volume serial numbers (one per line) for the dump tasks to use.

The `DUMP_RETENTION` parameter specifies how long the backups of the filesystem will be kept before the tapes are reused. This is also the value used by the `run_hard_deletes.sh` task to determine how old soft-deleted database entries must be before removing them from the database. Valid values for `DUMP_RETENTION` are a number followed by `m[inutes]`, `h[ours]`, `d[ays]`, or `w[eeks]`.

The `DUMP_DEVICE` parameter specifies the name of the device object in the configuration file that defines how to mount the tapes that the dump tasks will use. See "Device Objects" on page 66, for information about device objects.

If you set DUMP_MIGRATE_FIRST to YES, the dmmigrate command is run before the dumps are done to ensure that all migratable files are migrated, thus reducing the tapes needed for the dump. The default is NO.

The DUMP_INVENTORY_COPY parameter specifies the pathname of a directory into which are copied the xfsdump(1M) inventory files for the backed-up filesystems.

The DUMP_FILE_SYSTEMS parameter specifies one or more filesystems to dump. If not specified, the task dumps all the 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. It is safest not to specify a value for this parameter and therefore dump all filesystems configured.

The DUMP_VSNS_USED parameter is optional. It specifies the name of a file to which the tasks that dump the filesystems will append the VSN, one per line, of each volume used by xfsdump. If you don't specify this parameter, the task uses /dev/null as the file name.

The dump_tasks object employs scripts that call the xfsdump(1M) command in conjunction with the dmtape DMF support program. This mechanism gives you flexible and efficient use of a predetermined set of backup volumes that are automatically allocated to the xfsdump program as needed during the backup. In order to allow you an equally flexible and efficient method for restoring files backed up by the dump_tasks object, the dmxfsrestore(8) command should be used any time a restore is required for a dump_tasks-managed filesystem. See the dmxfsrestore(8) man page for more information on running the command.

Configuring Filesystems

You must have a `filesystem` object for each filesystem that can migrate files.

The `filesystem` object parameters are as follows:

Parameter	Value
`TYPE`	`filesystem` (required name for this type of object)
`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. For more information on message levels, see "General Message Log File Format" on page 108.
`MIGRATION_LEVEL`	Sets the level of migration service for the filesystem. Valid values are: • `none` (no migration) • `user` (only user-initiated migration) • `auto` (automated space management) The migration level actually used for the filesystem is the lesser of the `MIGRATION_LEVEL` of the daemon object and this value. The default is `auto`.
`POLICIES`	Specifies the names of the configuration objects defining policies for this filesystem.
`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. Currently there are no defined tasks for filesystems.

The following example defines a `filesystem` object:

```
define  /c
        TYPE                    filesystem
        MIGRATION_LEVEL         user
        POLICIES                fs_msp
enddef
```

Procedure 2-8 Configuring `filesystem` Objects

The following steps explain pertinent information for configuring the above `filesystem` object:

1. Ensure that `define` has a value that is the mount point of the filesystem you want DMF to manage. Do not use the name of a symbolic link. There is no default.

2. Ensure that `TYPE` is set to `filesystem`. There is no default.

3. The `MIGRATION_LEVEL` parameter determines the level of service for migration **to** offline media. Migration **from** offline media (either automatic or manual recall) is not affected by the value of `MIGRATION_LEVEL`.

   Configure `MIGRATION_LEVEL` to be one of the following:

   - `none` (no migration will take place on this filesystem)
   - `user` (users/administrators can perform `dmput`(1) or `dmmigrate`(8) commands but no other migration will take place)
   - `auto` (automated space management will be used on this filesystem)

   The default is `auto`.

   See "DMF Policies" and Procedure 2-9, page 57, for information about configuring automated space-management policies.

Note: `user` is the highest migration level that can be associated with a real-time partition.

---

4. Use the `POLICIES` parameter to declare one or more migration policies that will be associated with this filesystem. Policies are defined with `policy` objects (see "DMF Policies"). 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.

DMF Policies

A `policy` object is used to specify a migration policy. The following types of migration policies can be defined:

- Automated space management

- File weighting
- MSP selection
- Disk cache manager (DCM) use (see "DCM Policies" on page 61)

The following rules govern the use of `policy` objects with the `POLICIES` parameter of the `filesystem` object:

- The `POLICIES` parameter for a filesystem must specify one and only one MSP selection policy.
- If the `MIGRATION_LEVEL` for a filesystem is `auto`, the `POLICIES` parameter for that filesystem 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 groups of policy parameters (space management, file weight, and MSP or volume group selection) and share that policy among all the filesystems. Alternatively, you might create an MSP or volume group selection policy for all filesystems and a space-management policy (including weighting parameters) for all filesystems.

The `policy` object parameters described below are grouped by function.

Automated Space Management Parameters

DMF lets you automatically monitor filesystems and migrate data as needed to prevent filesystems from filling. This capability is implemented in DMF with a daemon called `dmfsmon`(8). After the `dmfsmon` daemon has been initiated, it will begin to monitor the DMF-managed filesystem to maintain the level of free space configured (in the configuration file).

Chapter 3, "Automated Space Management" on page 115, describes automated space management in more detail.

The following are parameters that control automated space management on a filesystem:

Note: Ideal values for these parameters are highly site-specific, based largely on filesystem sizes and typical file sizes.

Parameter	Description
TYPE	policy (required name for this type of object)
FREE_DUALSTATE_FIRST	When set to on, dmfsmon will free dual-state files before freeing files it will have to migrate first. The default is off.
FREE_SPACE_DECREMENT	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	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	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	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.

Note: The dump_tasks object employs scripts that call the xfsdump(1M) command in conjunction with the dmtape DMF support program. This mechanism gives you flexible and efficient use of a predetermined set of backup volumes that are automatically allocated to the xfsdump program as needed during the backup. In order to allow you an equally flexible and efficient method for restoring files backed up by the dump_tasks object, the dmxfsrestore(8) command should be used any time a restore is required for a dump_tasks-managed filesystem. See the dmxfsrestore(8) man page for more information on running the command.

File Weighting and MSP or Volume Group Selection Parameters

An important part of automatic space management is selecting files to migrate and determining where to migrate them. 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 using weighting-factor parameters in the configuration file.

DMF can be configured to have many MSPs or volume groups. Each MSP or volume group manages its own set of volumes. The MSP or volume group selection parameters allow you to direct DMF to migrate files with different characteristics to different MSPs or volume groups.

The file weighting and MSP or volume group selection parameters can be used more than once to specify that different files should have different weighting or MSP or volume group selection values.

The policy parameters for file weighting are as follows:

Parameter	Description
`AGE_WEIGHT`	Specifies a floating point constant and floating point multiplier to use to calculate the weight given to a file's age. `AGE_WEIGHT` is calculated as *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.
`SPACE_WEIGHT`	Specifies a floating point constant and floating point multiplier to use to calculate the weight given to a file's size. `SPACE_WEIGHT` is calculated as *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.

The parameter for MSP or volume group selection follows:

Parameter	Description
`SELECT_MSP` or `SELECT_VG`	Specifies the MSPs or volume groups to use for a file. You can list as many MSP or volume group names as you have MSP or volume

group objects defined. A copy of the file will be migrated to each MSP or volume group listed.

The special MSP or volume group name `none` means that the file will not be migrated. If you define more than one MSP or volume group, separate the names with white space.

You can specify either `SELECT_MSP` or `SELECT_VG`, but not both (however, the value of either parameter can be be a a mixture of both forms).

If no `SELECT_MSP` or `SELECT_VG` parameter applies to a file, it will not be migrated. The parameters are processed in the order they appear in the policy. There is no default.

The file weighting and MSP 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`	Days since last modification or last access of the file, whichever is more recent
`space`	Number of bytes the file occupies on disk (always a multiple of the blocksize, which may be larger or smaller than the length of the file)
`gid`	Group ID of one or more files
`uid`	User ID of one or more files

Combine expressions by using `and`, `or`, and `()`.

Use the operators =, >, <, =>, =<, and `in` to specify values.

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 (10 82-110 200)    (file's user ID is 10, in the range 82-110, or 200)
(gid = 55 or uid <= 123) and age < 5
                          (file's age is greater than 5 days and its
                          group ID is 55 or its user ID is higher than 123)
```

Configuring Policies

The following procedures explain how to create policies for automated space management (including file weighting) and MSP or volume group selection.

The following example defines a `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
```

Procedure 2-9 Configuring Objects for Automated Space Management

The following steps explain pertinent information for configuring the above `policy` object:

1. Ensure that `define` has a value you set previously in the `POLICIES` parameter of a `filesystem` object. There is no default.

2. Ensure that TYPE is set to policy. There is no default.

3. Configure automated space management as follows:

   a. Configure 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. The default is 30.

   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. The default is 20.

   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. The default is 10.

   d. Configure FREE_DUALSTATE_FIRST to be on if you want DMF to free the space used by dual-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 AGE_WEIGHT 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 they occur in the configuration file. If the when clause is present, DMF determines whether the file matches the criteria in the clause. If no 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 the example policy, 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 which have not been accessed or modified in 120 days or more have a far greater weight than all other files.

b. The syntax of SPACE_WEIGHT parameters 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 the example policy, the size of the file does not affect migration because all files have SPACE_WEIGHT of 0.

   A SPACE_WEIGHT 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
```

Note: DMF calculates the size weight and age weight separately. If either value is less than zero, the file is **not** automatically migrated or freed. Otherwise, the two values are summed to form the file's weight.

The following example defines a policy object for MSP or volume group selection:

```
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
```

Procedure 2-10 Configuring Objects for MSP or volume group 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. There is no default.

2. Ensure that TYPE is set to policy. There is no default.

3. Ensure that the MSP or volume group names you specify as the first value of the SELECT_MSP or SELECT_VG parameter is either the name of an MSP/LS you set previously in the MSP_NAMES or LS_NAMES parameter of the daemon object, or is the name of a volume group that is a component of an LS named in that same parameter. There is no default.

4. Configure MSP or volume group selection criteria as follows:

   a. If you want to select an MSP or volume group based on file size, use parameters such as the following, which send large files to cart1 and small files to cart2:

Note: The order of the SELECT statements is important. The first SELECT statement that applies to the file is honored. For example, if the following statements 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.

```
SELECT_MSP cart1        when space >= 50m
SELECT_MSP cart2        when space >= 65536
```

   b. If you want certain files to be copied to more than one MSP or volume group, use syntax such as the following, which migrates all files that have a group ID of 22 to both of the configured MSPs or volume groups:

```
SELECT_MSP cart1 cart2  when gid = 22
```

   Separate multiple MSP or volume group names with a blank space.

   c. If you want to ensure that some files are never migrated, you can designate the MSP or volume group 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_MSP none when space <= 4096
```

DCM Policies

A *disk cache manager* (DCM) is a disk MSP that has been configured to use a dedicated filesystem as a cache to improve the performance of a tape-based volume group. This cache has similar requirements to those of a disk-MSP-managed filesystem:

- Automatic space management
- File weighting
- Selection of one or more volume groups to provide tape-based storage

DCM uses the following configuration parameters, which are similar to standard disk MSP parameters:

DCM	Standard Disk MSP
DUALRESIDENCE_TARGET	MIGRATION_TARGET
FREE_DUALRESIDENT_FIRST	FREE_DUALSTATE_FIRST
SELECT_LOWER_VG	SELECT_VG
CACHE_AGE_WEIGHT	AGE_WEIGHT

`CACHE_SPACE_WEIGHT` `SPACE_WEIGHT`

The DCM parameters have the same format and acceptable values as their disk-MSP-managed filesystem counterparts, with the following differences:

- The DCM supports the concept of *dualresidence*, 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 dualstate file in the standard disk-MSP-managed filesystem 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.
- `SELECT_LOWER_VG` defines which volume groups should maintain tape-based copies of files in the cache, and under what conditions that would define dualresidence.

The following disk MSP space management parameters are also applicable to DCM:

```
FREE_SPACE_MINIMUM
FREE_SPACE_TARGET
FREE_SPACE_DECREMENT
SITE_SCRIPT
```

Setting Up Tape MSPs

Each MSP you create must have an object defined in the configuration file, which refers to a device object describing the tape drives to be used. Normally, several MSPs share a single device object.

MSP Objects

The tape MSP entry has the following options:

Option	Description
`TYPE`	`msp` (required name for this type of object)
`CACHE_DIR`	Directory in which the MSP 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.

CACHE_SPACE	Amount of disk space (in bytes) that dmatmsp 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. For more information, see Procedure 2-11, step 5 on page 66.
CHILD_MAXIMUM	Maximum number of child processes the MSP is allowed to fork. The maximum value is 100; the default is 4.
COMMAND	Binary file to execute in order to initiate this MSP. For the tape MSP, this value must be dmatmsp.
DISK_IO_SIZE	Transfer size (in bytes) used when reading from or writing to files within a DMF filesystem. The value must be in the range 4096–16m (16 million). The default is 65536.
HFREE_TIME	Minimum number of seconds that a tape no longer containing valid data must remain unused before the MSP overwrites it. The default value is 172,800 seconds (2 days), and the minimum allowed value is 0. When an MSP removes all data from a tape, it sets the hfree (hold free tape) flag bit in the tape's volume (VOL) database entry to prevent that tape from being immediately reused. The next time the MSP scans the database for volumes after HFREE_TIME seconds have passed, the MSP clears the hfree flag, allowing the tape to be rewritten. If HFREE_TIME is set to 0, the MSP 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_CACHE_FILE	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.
MAX_CHUNK_SIZE	Specifies that the MSP 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 MSP only breaks a file into chunks when an end of volume is reached.
`MAX_PUT_CHILDREN`	Maximum number of write child processes the MSP will schedule. The default and the maximum are the value of `CHILD_MAXIMUM`; the minimum is 1.
`MERGE_CUTOFF`	Limit at which the MSP stops scheduling tapes for merging. This number refers to the sum of the active and queued children generated from gets, puts, and merges. The default is `CHILD_MAXIMUM`, which means that if sparse tapes are available, children will be created until there are `CHILD_MAXIMUM` children, thus using tape efficiently. However, if any recall requests arrive, they will be started before new merges. Setting this number below `CHILD_MAXIMUM` reserves some tape units for recalls at the expense of merge efficiency. Setting this number above `CHILD_MAXIMUM` increases the priority of merges relative to recalls.
`MESSAGE_LEVEL`	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 "General Message Log File Format" on page 108.
`MIN_TAPES`	Minimum number of unused tapes that can exist in the MSP VOL database before operator notification. If the number of unused tapes falls below `MIN_TAPES`, the operator will be asked to add new tapes. The default is 10; the minimum is 0.
`TAPE_TYPE`	Specifies the name of a device object that describes how the tapes are accessed and used. There is no default. The `device` object is described in "Device Objects" on page 66.

TASK_GROUPS	Names the task groups that contain tasks the MSP should run. They are configured as objects of TYPE taskgroup. There is no default. See "Configuring Maintenance Tasks for Tape MSP and LS" on page 91, for more information.
TIMEOUT_FLUSH	Minutes after which the MSP will flush files to tape. The default is 120 minutes.

The following example does not use all of the possible options for configuring a tape MSP; it defines two tape MSPs named cart1 and cart2.

```
define  cart1
        TYPE                    msp
        COMMAND                 dmatmsp
        TAPE_TYPE               SILO_1
        CACHE_SPACE             110m
        CHILD_MAXIMUM           3
        MESSAGE_LEVEL           2
        TASK_GROUP              msp_tasks
enddef
#
define  cart2
        TYPE                    msp
        COMMAND                 dmatmsp
        TAPE_TYPE               SILO_2
        CACHE_SPACE             50m
        CACHE_DIR               /cache
        MAX_CACHE_FILE          50m
        CHILD_MAXIMUM           10
        TASK_GROUP              msp_tasks
enddef
```

Procedure 2-11 Configuring Tape MSPs

The following steps explain pertinent information for configuring the msp objects:

1. Ensure that define has a value that you set previously in the MSP_NAMES parameter of the daemon object. There is no default.

2. Ensure that TYPE is set to msp. There is no default.

3. Ensure that COMMAND is set to dmatmsp. There is no default.

4. Define a TAPE_TYPE parameter that names the device type object for the MSP. There is no default. Use the value you set here in defining device objects. See "Device Objects" on page 66.

5. Configure the CACHE_SPACE parameter to be at least twice the configured tape zone size. If you do not set this parameter, DMF will merge tapes via sockets, which means that the read and write children have to synchronize. Using CACHE_SPACE is far more efficient, especially for small files.

   The MSP is able to merge tapes more efficiently if it can stage most of the files to disk. Setting the CACHE_SPACE parameter tells the MSP how much disk space it can use. The MAX_CACHE_FILE parameter specifies the largest file it will place in the CACHE_SPACE. The default for CACHE_SPACE is 0, which causes all data to be transferred by sockets.

   See "Media Concepts" on page 137, for more information on tape zone sizes.

6. Configure the CHILD_MAXIMUM to be the number of tape drives this MSP can use. The default is 4, and the maximum is 24.

7. Configure the MESSAGE_LEVEL of an MSP to be higher than 2 (the default) for debugging purposes only. Valid values are 0 to 6.

8. Configure the MAX_CACHE_FILE to be the size (in bytes) of the largest chunk that will be merged using the merge cache space (defined by CACHE_SPACE). Large files are transferred directly via socket. The largest value you can use is the value of CACHE_SPACE, and the default is 25% of CACHE_SPACE.

9. Configure the TASK_GROUPS parameter to the names of the objects used to define how periodic maintenance tasks are completed. There is no default. See "Configuring Maintenance Tasks for Tape MSP and LS" on page 91, for more information.

Device Objects

Each tape device type name you use in the MSP or in the dump_tasks object should be defined as a device object in the configuration file. The parameters you define are based on which 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)

`BLOCK_SIZE`	Block size used when writing tapes from the beginning. The default depends upon the device, with DMF setting defaults as follows:
	`AMPEX DIS/DST` `1199840`
	`DLT` `131072`
	`STK 9840` `126976`
	`Other devices` `65536`

`LABEL_TYPE`	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`.
`MOUNT_SERVICE`	Specifies the mounting service to use. Supported values are `openvault` and `tmf`. This parameter is required; there is no default.
`MSG_DELAY`	Specifies the number of seconds that all devices in the object 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.
`POSITIONING`	Specifies how the tape should be positioned to a zone. One of: • `skip` (tape mark skipping) • `direct` block ID seek capability if the block ID is known) The default is `direct`.
`POSITION_RETRY`	Level of retry in the event of a failure during zone positioning. Possible values are: • `none`

- `lazy` (the MSP will retry if a reasonably fast alternative method of positioning is available)
- `aggressive` (the MSP may try more costly and time-consuming alternatives)

	If the MSP is unable to position to a zone, the MSP aborts all recalls for files with data in that zone (however, DMF does not abort them if a copy exists in another MSP). The default is `lazy`.
`VERIFY_POSITION`	Specifies whether the tape MSP 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.
`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`.
`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 database. Smaller values allow faster recalls and better recoverability but poorer write performance. The MSP also uses zone size to determine when to start write children. The default is 50 MB.

Device Objects for OpenVault As Mounting Service

The device object may have the following parameters when it is configured for OpenVault:

Parameter	Description
`OV_ACCESS_MODES`	Specifies a list of access mode names that control how data is written to tape. The default value is `readwrite` when migrating and `readonly` when recalling. This parameter is optional.

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.

Examples of the use of these parameters are provided in Procedure 2-14, page 85.

OpenVault requires several configuration steps in addition to configuring the device object. They are described in "Using OpenVault for Tape MSPs and LS Drive Groups" on page 85.

Device Objects for TMF as Mounting Service

Tape mounting can be accomplished by using the Tape Management Facility (TMF). To use TMF as a mounting service, there are no required parameters that you must specify, but the `TMF_TMMNT_OPTIONS` parameter allows you to specify some `tmmnt` options:

Parameter	Description
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 the `BLOCK_SIZE` and `LABEL_TYPE` device parameters instead. Use `-g` if the group name is different from the device object's name. Use `-i` to request compression.

The following example defines a device object for use with TMF:

```
define  SILO_3
        TYPE                    device
        MOUNT_SERVICE           tmf
        BLOCK_SIZE              131072
        LABEL_TYPE              sl
        TMF_TMMNT_OPTIONS       -g DLT
enddef
```

Procedure 2-12 Configuring Devices for TMF

The following steps explain pertinent information for configuring the `device` object for TMF:

1. Ensure that `define` has a value that you set previously in the `TAPE_TYPE` parameter of the `msp` object. There is no default.

2. Ensure that `TYPE` is set to `device`. There is no default.

3. Configure the `MOUNT_SERVICE` to be `tmf`.

Note: DMF uses the `-Z` option to `tmmnt`, so options controlling block size and label parameters would be ignored if you were to specify them for the `TMF_TMMNT_OPTIONS` parameter. Use the `BLOCK_SIZE` and `LABEL_TYPE` device parameters instead.

---

4. Configure the `BLOCK_SIZE` parameter to be the block size used when writing tapes from the beginning. In the example, `131072` is used because DLTs write more efficiently with this blocksize.

5. Configure the `LABEL_TYPE` parameter to be the label type used when writing tapes from the beginning. In the example, `sl` is used to specify standard label for IBM tapes.

6. Configure the `TMF_TMMNT_OPTIONS` parameter to specify command options that should be added to the `tmmnt` command when mounting a tape. In the example, the `-g` option specifies that the TMF tape group is `DLT`. If this option on this parameter had not been specified, DMF would have used the name of this device object (in the example, `SILO_3`).

Setting Up Library Servers

Each object shown in Figure 1-3 on page 9, must have an object defined in the configuration file. The options shown in the following sections are only the most common. For the complete set, see the `dmf.conf`(5) man page. For a summary of the parameters and the object to which they apply, see Table 2-4 on page 111.

Library Server Objects

The entry for an LS, one for each tape library, has the following options:

Option	Description
TYPE	libraryserver (required name for this type of object)
CACHE_DIR	Directory in which the volume group 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.
CACHE_SPACE	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.
COMMAND	Binary file to execute to initiate the LS. This value must be dmatls.
DRIVE_GROUPS	Names one or more drive groups containing drives that the LS can use for mounting and unmounting volumes. They are configured as objects of type drivegroup. 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 or volume group 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 LS object's DRIVE_GROUPS parameter. It then searches for the copy in volume group order from the drivegroup object's VOLUME_GROUPS parameter.
MAX_CACHE_FILE	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	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 "Automated Maintenance Tasks" on page 31.
TASK_GROUPS	Names the task groups that contain tasks the LS should run. They are configured as objects of TYPE taskgroup. There is no default.
WATCHER	Names the resource watcher that the LS should run. They can be configured as objects of type resourcewatcher, but if the default parameters are acceptable, there is no need to do this. The default is no watcher.

Drive Group Objects

The entry for a drive group, one for each pool of interchangeable drives in a single library, has the following options:

Option	Description
TYPE	drivegroup (required name for this type of object)
BLOCK_SIZE	Block size used when writing tapes from the beginning. The default depends upon the device, with DMF setting defaults as follows: AMPEX DIS/DST 1199840 DLT 131072 STK 9840 126976 Other devices 65536
DISK_IO_SIZE	Transfer size (in bytes) used when reading from or writing to files within a DMF filesystem. The value must be in the range 4096 –16m (16 million).The default is 65536.
DRIVE_MAXIMUM	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 objects of type resourcescheduler. The default is a resource scheduler of default type and parameters. For the defaults, see "Resource Scheduler Objects" on page 80.
DRIVES_TO_DOWN	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. The default of 0 prevents the drive group from disabling any.
LABEL_TYPE	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 to use. Possible values are openvault and tmf. The default is openvault.
MOUNT_SERVICE_GROUP	Specifies the name by which the drive group's devices are known to the mounting service. In the case of 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. 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 the mount service, in an attempt to force the hanging subsystem to resume normal operation, or to fail solidly. Do not make this value too restrictive, as any non-LS tape activity (including MSPs and `xfsdump`) can legitimately delay a volume group'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 a list of access mode names that control how data is written to tape. The default value is `readwrite` when migrating and `readonly` when recalling. This parameter is optional.
`OV_INTERCHANGE_MODES` (Open Vault `MOUNT_SERVICE` 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:

- `none`
- `lazy`, which means the volume group retries if a reasonably fast alternative means of positioning is available.
- `aggressive`, which means the volume group can try more costly and time consuming alternatives.

If the volume group is unable to position to a zone, all recalls for files with data in that zone are aborted by the volume group (though not by DMF if a copy exists in another volume group).

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`.

`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).

`RUN_TASK` See "Automated Maintenance Tasks" on page 31.

`TASK_GROUPS`	Names the task groups that contain tasks the drive group should run. They are configured as objects of `TYPE taskgroup`. 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` device parameters instead. Unlike a tape MSP, 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 volume groups containing volumes that can be mounted on any of the drives within this drive group. They are configured as objects of type `volumegroup`. 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 or volume group 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 LS object's `DRIVE_GROUPS` parameter. It then searches for the copy in volume group order from the drivegroup object's `VOLUME_GROUPS` parameter.

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.

Volume Group Objects

The entry for a volume group, one for each pool of tape volumes of the same type, usable on the drives of the associated drive group, and which is capable of holding at most one copy of user files, has the following options:

Option	Description
TYPE	volumegroup (required name for this type of object)
ALLOCATION_GROUP	Name of an allocation group that serves as a source of additional volumes if a volume group runs out of media. Normally, one allocation group is configured to serve multiple volume groups. As a volume's hfree flag is cleared (see HFREE_TIME below) in a volume group, 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 volume group that references the allocation group. It is an error to assign an ALLOCATION_GROUP name that is the same as an existing volume group name. The ALLOCATION_GROUP defines a logical pool of volumes rather than an actual operational volume group. As allocation groups have no configurable parameters, they have no configuration stanzas of their own; a reference to them in a volume group's ALLOCATION_GROUP parameter is all that is needed to activate them. A volume group that does not define ALLOCATION_GROUP will not use one.
ALLOCATION_MAXIMUM	Maximum size in number of volumes to which a volume group can grow by borrowing volumes from its allocation group. The minimum value is 0, the maximum is infinity, and the default is infinity. If the volume group 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`	Minimum size in number of volumes to which a volume group can shrink by returning volumes to its allocation group. The minimum value is 0, which is the default, and the maximum is the current value of `ALLOCATION_MAXIMUM`. If the volume group 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`	Maximum number of drives within this drive group that this volume group is allowed to use simultaneously. The value actually used is the least of the drive group's `DRIVE_MAXIMUM`, this volume group's `DRIVE_MAXIMUM` and the number of drives the drive group can physically detect. The maximum is 100; the default is the drive group's `DRIVE_MAXIMUM`.
`HFREE_TIME`	Minimum number of seconds that a tape no longer containing valid data must remain unused before the volume group overwrites it. The default value is 172,800 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 volume (VOL) database entry to prevent that tape from being immediately reused. The next time 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 volume group 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 volume group 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 volume group. The maximum value is the value of DRIVE_MAXIMUM for the associated drive group. The minimum value is 1. The default is the value that the volume group uses for DRIVE_MAXIMUM.

MERGE_CUTOFF
: Specifies a limit at which the volume group 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 volume group for DRIVE_MAXIMUM. This means that if sparse tapes are available, the volume group will create DRIVE_MAXIMUM number of children, thus using tape resources efficiently. However, if any recall requests arrive for that volume group, 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.

MIN_VOLUMES
: Minimum number of unused volumes that can exist in the LS's volume database for this volume group 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 volume group has an allocation group configured, MIN_VOLUMES is applied to the sum of the number of unused volumes in the volume group and in its allocation group subject to any ALLOCATION_MAXIMUM restrictions.

PUTS_TIME
: Specifies the minimum number of seconds a volume group 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 volume group 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 volume group will not do this evaluation. The default is 0.
`RUN_TASK`	See "Automated Maintenance Tasks" on page 31.
`TASK_GROUP`	Names the task groups that contain tasks the volume group should run. They are configured as objects of `TYPE taskgroup`. There is no default.
`TIMEOUT_FLUSH`	Minutes after which the volume group 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 database. Smaller values allow faster recalls and better recoverability but poorer write performance. The volume group also uses zone size to determine when to start write children. The default is 50 MB.

Resource Scheduler Objects

The entry for a resource scheduler, one for each drive group in a single library, has the following options:

Option	**Description**
`TYPE`	`resourcescheduler` (required name for this type of object)
`ALGORITHM`	The resource scheduling algorithm to be used. Two are currently supplied: a simple one called `fifo`, and a more flexible one called `weighted_roundrobin` (default).

Note: Sites can write their own algorithm to meet specialized needs. Instructions can be found in the `/usr/share/doc/dmf-`*version_number*`/info/sample/RSA.readme` file about the resource scheduling algorithm.

`MODULE_PATH`	The pathname of a Dynamic Shared Object (library of runtime-loadable routines) that contains an RSA whose name was specified by the `ALGORITHM` parameter. The default is to use the built-in RSAs.

Other parameters are specific to a particular RSA. There are no parameters for `fifo`. For `weighted_roundrobin`, the following apply:

Option	**Description**
`PENALTY`	Reduces the priority of requests from a volume group 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 volume group 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 volume groups. The ratio of these weightings to each other (within the one drive group) determines the number of opportunities the volume group has to obtain drives when they are needed. The weightings are integers in the range 1–99, and need not be unique. For efficiency reasons, small numbers are preferred, especially if large numbers of volume groups are defined. Usually, there are multiple `WEIGHT` lines in the configuration, and a given volume group might appear on more than one of them. In such cases, the sum of the weights is used as the effective weight for that volume group. Any volume groups that do not appear on a `WEIGHT` line are assigned the default of 5. If there are no `WEIGHT` lines, all volume groups will use this default, resulting in a strict round-robin behavior.

`WEIGHT` has the following format:

`WEIGHT` *weight vg1 vg2 ...*

Resource Watcher Objects

The entry for a resource watcher is needed only if you wish to change its default parameters; a reference to an resource watcher by the LS is sufficient to activate it. The resource watcher has the following options:

Option	Description
`TYPE`	`resourcewatcher` (required name for this type of object)
`HTML_REFRESH`	The refresh rate (in seconds) of the generated HTML pages. The default is 60.

Example

The following code example does not use all of the possible options for configuring an LS. It defines an LS containing a default resource watcher and one drive group, which in turn contains two volume groups sharing an allocation group, and a resource scheduler to give one volume group twice the priority than the other when competing for drives.

The volume group objects are slightly different, reflecting that the first one handles all of the recalls in normal circumstances as well as migrations, but the second is usually write-only.

```
define  ls1
                TYPE                    libraryserver
                COMMAND                 dmatls
                DRIVE_GROUPS            dg1
                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     drives
        OV_INTERCHANGE_MODES    compression
        DRIVE_SCHEDULER         rs
        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
enddef

define  vg_sec
        TYPE                    volumegroup
        ALLOCATION_GROUP        ag
        DRIVE_MAXIMUM           2
enddef
```

The steps in Procedure 2-13, page 83, explain pertinent information for configuring each of the LS objects in the previous example.

Procedure 2-13 Configuring a Library Server and Its Components

1. Ensure that `define` has a value that you set previously in the `LS_NAMES` or `MSP_NAMES` parameter of the daemon object. There is no default.

2. Ensure that `TYPE` is set to `libraryserver`. There is no default.

3. Ensure that `COMMAND` is set to `dmatls`. There is no default.

4. Specify a `DRIVE_GROUPS` parameter that names a collection of interchangeable tape drives. The assumption in this example is that there is only one such group. There is no default.

5. To tell the LS how much disk space it can use, set the `CACHE_SPACE` parameter. The LS can merge tapes more efficiently if it can stage most of the files to disk. Configure the `CACHE_SPACE` parameter to be at least twice the configured tape zone size. The default for `CACHE_SPACE` is 0, which causes all data to be

transferred by sockets. For more information on tape zone sizes, see "Media Concepts" on page 137.

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 "Configuring Maintenance Tasks for Tape MSP and LS" on page 91.

7. To observe LS operation through a web browser, define a resource watcher. You need only a reference. Define an resource watcher object only if you want to change its default parameters.

   Assuming that SPOOL_DIR was set in the base object to be /dmf/spool, the URL to use in 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 group 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 volume groups using the drives managed by this drive group with the VOLUME_GROUPS parameter.

10. Specify the use of OpenVault. Because Open Vault 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. Override the default resource scheduler behavior by referring to an object called rs, to be defined later.

14. Allow the 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, volume group `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 volume groups. The `VOLUME_GROUPS` parameter of the drive group object and the `SELECT_LS` or `SELECT_MSP` lines in the filesystem objects refer to them.

17. Define a common allocation group called `ag`. allocation groups have no configurable parameters, so they have no defining object; just a reference is sufficient. Use of an allocation group is optional.

18. Include any other volume group parameters that you require. For example, one of the previous steps specified that the secondary volume group `vg_sec` can use, at most, two tape drives, so that other drives in this drive group are immediately available for use by `vg_prim` when it needs them.

Using OpenVault for Tape MSPs and LS Drive Groups

This section describes the steps you must take to configure OpenVault for a tape MSP or a drive group. You must execute OpenVault commands, create security key files, and edit the DMF configuration file.

Procedure 2-14 Configuring DMF to Use OpenVault

The following procedure describes how to make OpenVault and DMF work together. When using OpenVault 1.5 and later versions, you can use the `ov_admin` script to enable the DMF application. When using earlier versions of OpenVault, you can use the `setup` script. See the *OpenVault Operator's and Administrator's Guide* for a description of this script.

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.

---

1. On the OpenVault server, add DMF as both a privileged and unprivileged OpenVault application for this host.

   When using versions of OpenVault prior to 1.5, use the `setup` script, menu item 1, submenu 5.

   When using OpenVault 1.5 or later versions, use the `ov_admin` script, and select the menu option that allows you to manage applications. Create the DMF application, and activate both a privileged and unprivileged instance of it.

   The application name should be `dmf` (in lowercase). The instance name should be `dmf@`*hostname* where `dmf` is in lowercase, and `hostname` is the output of the command `hostname -s`. For example:

   ```
   % hostname -s
   system1
   ```

   In this case, `dmf@system1` would be the instance name.

2. 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 MSP or LS might result in tape mount failures.

   When configuring tape MSPs, ensure that the value for `CHILD_MAXIMUM` does not exceed the number of drives in the OpenVault drive group.

   When using OpenVault 1.4.*x* or earlier releases, it is preferable that you use the OpenVault `setup` script, menu item 2, submenu 7. When using OpenVault 1.5 or later, choose the appropriate item from the **ov_admin** menu. If for some reason you cannot use the `setup` or `ov_admin` script, you can enter the command manually, as follows:

   `ov_drivegroup -a` *drive_group* `-A dmf`

3. Add DMF as a valid application to appropriate cartridge groups.

   For OpenVault versions prior to 1.5, it is preferable that you use the OpenVault `setup` script, menu item 2, submenu 8.

For OpenVault 1.5 and later, 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 `setup` or the `ov_admin` script, you can enter the command manually, as follows:

```
ov_cartgroup -a tape_group -A dmf
```

4. 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
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`.

   b. Configure the `OV_SERVER` parameter to the value returned by the `hostname`(1) command on the machine on which the OpenVault server is running. This parameter only applies when OpenVault is used as the mounting service. The default value is the host name of the machine on which you are running.

5. 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 1.

6. (This step does not apply to LSs). Configure the MSP's device object for use with OpenVault, as follows:

```
define  timber
        TYPE                    device
        MOUNT_SERVICE           openvault
        OV_ACCESS_MODES         readwrite
        OV_INTERCHANGE_MODES    compression
```

```
        ZONE_SIZE                200m
enddef
```

a. Ensure that define has a value that you set previously in the TAPE_TYPE parameter of the msp object. There is no default.

b. Configure TYPE to be device. There is no default.

c. Configure the MOUNT_SERVICE parameter to be openvault.

d. Configure the OV_ACCESS_MODES parameter to be a list of access mode names that control how the tape is used. The parameter is optional. The default value is readwrite when migrating and readonly when recalling. Use this parameter to force readwrite.

   The other possible values that OpenVault can use are not configurable in DMF: for rewind/norewind, DMF uses rewind; for variable/fixed, DMF uses variable.

e. Configure the OV_INTERCHANGE_MODES parameter to be 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 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.

f. Configure other parameters relevant to your site. The example sets the ZONE_SIZE parameter to 200 MB. The target zone size is a major factor in determining how much data is written before writing a tape mark and updating the MSP database. Here, the tapes used by the MSP will, in general, have more data written in a zone than DMF uses as a default. Smaller values allow faster recalls and better recovery, but they cause poorer write performance than larger values. The default is 50 MB. See "Media Concepts" on page 137, for more information on how tape zone sizes are determined.

7. (This step does not apply to MSPs). Configure the LS's drive group object for use with OpenVault. In the drive group 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_ACCESS_MODES` parameter to be a list of access mode names that control how the tape is used. The parameter is optional. The default value is `readwrite` when migrating and `readonly` when recalling. Use this parameter to force `readwrite`.

   The other possible values that OpenVault can use are not configurable in DMF: for `rewind/norewind`, DMF uses `rewind`; for `variable/fixed`, DMF uses `variable`.

d. Configure the `OV_INTERCHANGE_MODES` parameter to be 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 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.

8. Make the appropriate cartridges accessible to the MSPs, allocation groups, or volume groups by assigning the cartridges to the DMF application in OpenVault. To do this, you must know the following:

   - Cartridge type name. To determine the cartridge types allowed by a given drive, enter the following:

     ```
     ov_stat -c -D drive | grep base
     ```

     The fourth column shown in the output is the cartridge type.

   - Cartridge group. To determine the possible cartridge groups, enter the following:

     ```
     ov_cartgroup -l -A dmf
     ```

a. If you already have tapes defined in your MSP or LS database, tell OpenVault about these tapes by entering one of the following:

dmov_makecarts -g *cartgroup* -t *carttype* *mspname*
dmov_makecarts -g *cartgroup* -t *carttype* *lsname*
dmov_makecarts -g *cartgroup* -t *carttype* -v *vg1, vg2 lsname*

You can replace any of the references to a volume group previously mentioned with an allocation group. If the -v parameter is omitted, all volume groups and allocation groups in the specified LS will be processed.

b. If there are unmanaged cartridges in an OpenVault managed library, you can import the unmanaged cartridges, assign them to DMF, and add them to a database by entering one of the following:

dmov_loadtapes -l *library* -g *cartgroup* -t *carttype* *mspname*
dmov_loadtapes -l *library* -g *cartgroup* -t *carttype* *vgname*
dmov_loadtapes -l *library* -g *cartgroup* -t *carttype* *agname*

This command will invoke a vi(1) session. In the vi(1) session, delete any cartridges that you do **not** want added to the database.

c. If neither of the above cases are appropriate, you can manually configure the cartridges. The following commands can be useful in this effort:

- To list cartridges in a library, enter the following:

  ov_stat -s -L *library*

- To list information on cartridges known to OpenVault, enter the following:

  ov_lscarts -f '.*'

- To import cartridges into OpenVault and optionally assign them to DMF use the ov_import command.

- To assign a cartridge known to OpenVault to an application, use the ov_vol command with the -n option.

Using TMF tapes with Tape MSPs and LS Drive Groups

Use one of the following dmvoladm(8) commands to add tapes to the MSP and/or LS databases:

```
dmvoladm  -m mspname  -c 'create vsn001-vsn010'
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 volume group.

There is no special procedure to inform TMF of the tapes' existence. TMF assumes that every tape it deals with is in the library or can be provided by an operator, as needed.

Configuring Maintenance Tasks for Tape MSP and LS

You can configure parameters for how the tape MSP or LS daemon performs the following maintenance tasks:

- Creating tape reports with the run_tape_report.sh and run_compact_tape_report.sh tasks
- Merging sparse tapes with the run_tape_merge.sh task and the THRESHOLD, VOLUME_LIMIT, and DATA_LIMIT parameters
- Stopping tape merges at a specified time with the run_merge_stop.sh task

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 daemon_tasks object, but these tasks also clear the MSP/LS logs and journals. These tasks are described in "Configuring Daemon Maintenance Tasks" on page 44.

Table 2-1 on page 33, provides a summary of automated maintenance tasks.

The following example explains how to define the msp_tasks object. You can change the object name itself (msp_tasks) to be any name you like.

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.

```
define msp_tasks
 TYPE   taskgroup
 RUN_TASK $ADMINDIR/run_tape_report.sh at 00:10
#
 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 2-15 Configuring the msp_tasks Object

1. Define the object to have the same name that you provided for the TASK_GROUPS parameter of the tape msp object. In the example it is msp_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*.

   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
```

The following steps specify the information you must provide for the tasks to run correctly:

a. The `run_tape_report.sh` generates a report on the tapes in the MSP tape pool and on MSP activity. In the example, it runs every day at 10 minutes after midnight.

b. 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.

   - Use the `DATA_LIMIT` parameter to set the maximum amount of data (in bytes) that should be selected for merging at one time.

c. As this might become cumbersome when there are large numbers of volume groups configured, an alternative has been provided to `run_tape_merge.sh`, called `run_merge_mgr.sh`. This script establishes the needs of the volume groups for more tapes, using their `MIN_VOLUMES` parameters as a guide to expected requirements. The script processes the most urgent ones first, minimizing interference with the production workload. To use this script, perform the following steps:

   1.) Define a taskgroup, which is referred to by the drivegroup object (not the volume group or LS object).

   2.) Specify a `RUN_TASK` parameter for `run_merge_mgr.sh` in the taskgroup, and optionally, another for `run_merge_stop.sh`. You can also specify `MESSAGE_LEVEL`, `THRESHOLD`, `VOLUME_LIMIT`, and `DATA_LIMIT` parameters.

   3.) Ensure that the LS object that refers to this drive group has a resource watcher defined via the `WATCHER` parameter.

4.) For each volume group, confirm that the value of its `MIN_VOLUMES` parameter is realistic.

`run_merge_mgr.sh` is not available for use with MSPs because it requires the resource watcher feature of the LS.

d. Use the `run_merge_stop.sh` task to shut down volume merging (tape 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.

Library Server and MSP Database Records

After you have added the tape MSP/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 the volume (VOL) and catalog (CAT) records in the MSP/LS database.

You can follow the steps in Procedure 2-16, page 94, for all of the tape MSPs/LSs you have defined.

Caution: Each tape MSP/LS must have a unique set of volume serial numbers.

Procedure 2-16 Creating MSP/LS Database Records

The following procedure is shown as an example that assumes you have an MSP called `cart1`.

1. Enter the following command and it will respond as shown:

   ```
   % dmvoladm -m cart1
   dmvoladm: at rdm_open - created database atmsp_db
   adm: 1>
   ```
   The response is an informational message indicating that `dmvoladm` could not open an existing MSP database, so it is creating a new and empty one. You should get this message the first time you use `dmvoladm` for an MSP, but never again. The next line is the prompt for `dmvoladm` directives.

2. Assume that you will use 200 tapes of type `CART` with standard labels `PA0001` through `PA0200`.

After the prompt, enter the following directive:

```
adm:1> create PA0001-PA0200
```

After entering this directive, you will receive 200 messages, one for each entry created, beginning with the following:

```
VSN PA0001 created.
VSN PA0002 created.
```

3. Use the following `dmvoladm` directive to list all of the tape VSNs in the newly created library:

```
adm:2> list all
```

Note: The `dmvoladm tapesize` field is purely for site documentation and is not used by the MSP. The `blocksize` field documents the value used when the tape is first written or rewritten. It should **not** be changed in the database; however, if you want another value, change the `BLOCK_SIZE` *nnn* configuration parameter of the `device` object.

---

4. Issue the `dmvoladm quit` directive to complete setting up the MSP.

```
adm:3> quit
```

Procedure 2-17 Creating LS Database Records

The following procedure is shown as an example that assumes you have an LS called `ls1`. This LS contains a volume group 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 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 volume group `vg_pri` for the tapes being added. It is also valid to specify an allocation group name instead of a volume group 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. Use the following `dmvoladm` directive to list all of the tape VSNs in the newly created library:

   ```
   adm:2> list all
   ```

4. Issue the `dmvoladm quit` directive to complete setting up the LS.

   ```
   adm:3> quit
   ```

Setting up FTP MSPs

To enable a file transfer protocol (FTP) MSP, include a name for it on the `MSP_NAMES` parameter in the `daemon` 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 site-determined pace. Contact your customer service representative for information about technical assistance with fileserver conversion.

An FTP MSP object has the following options (defaults are provided here or in Procedure 2-19, page 105):

Parameter	**Description**
`TYPE`	`msp` (required name for this type of object)
`CHILD_MAXIMUM`	Maximum number of child processes the MSP is allowed to fork. The default is 4; the maximum is 100.
`COMMAND`	Binary file to execute in order to initiate this MSP. For the FTP MSP, this value must be `dmftpmsp`.

DISK_IO_SIZE	Transfer size (in bytes) used when reading from or writing to files within a DMF filesystem. The value must be in the range 4096–16m (16 million). The default is 65536.
FTP_ACCOUNT	Account ID to use when migrating files to the remote system.
FTP_COMMAND	Additional commands to send to the remote system. There may be more than one instance of this parameter.
FTP_DIRECTORY	Directory to use on the remote system.
FTP_HOST	Internet host name of the remote machine on which files are to be stored.
FTP_PASSWORD	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	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	User name to use when migrating files to the remote system.
GUARANTEED_DELETES	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	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. Legal values are: • on (or yes) • off (or no)

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 "General Message Log File Format" on page 108.

`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 creates names for files stored on remote machines in the `STORE_DIRECTORY`. For a list of possible strings, see Table 2-2.

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, 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 "Configuring Daemon Database Record Length" on page 29.
TASK_GROUPS	Names the task groups that contain tasks the MSP should run. They are configured as objects of TYPE taskgroup. There is no default. Currently there are tasks defined only for the tape MSP.

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 2-2 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
%g	Group name of the file
%G	Group ID of the file
%%	Literal % character
%d	Current day of month (two characters)
%H	Current hour (two characters)
%m	Current month (two digits)
%M	Current minute (two digits)
%S	Current second (two digits)
%y	Last two digits of the current year (such as 03 for 2003)

The following example defines 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 2-18 Configuring the `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 `daemon` object. There is no default.
2. Ensure that `TYPE` is set to `msp`. There is no default.
3. Ensure that `COMMAND` is set to `dmftpmsp`. There is no default.
4. Set the `FTP_USER` parameter to the user name to use on the remote FTP server during session initialization. There is no default.
5. 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.
6. 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.
7. Set the `FTP_DIRECTORY` parameter to the directory into which files will be placed on the remote FTP server. There is no default.
8. 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.

Setting up Disk MSPs

To enable a disk MSP, include a name for it on the `MSP_NAMES` parameter in the `daemon` 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 options:

Parameter	Description
TYPE	msp (required name for this type of object)
CHILD_MAXIMUM	Maximum number of child processes the MSP is allowed to fork. The default is 4; the maximum is 100.
COMMAND	Binary file to execute in order to initiate this MSP. For the disk MSP, this value must be dmdskmsp.
DISK_IO_SIZE	Transfer size (in bytes) used when reading from or writing to files within a DMF filesystem. The value must be in the range 4096–16m (16 million). The default is 65536.
GUARANTEED_DELETES	Number of child processes that are guaranteed to be available for processing delete requests. The default is 1.
GUARANTEED_GETS	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. Does not apply to DCM-mode.
IMPORT_ONLY	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. Does not apply to DCM-mode.
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 "General Message Log File Format" on page 108.
`NAME_FORMAT`	Specifies the strings that form a template to creates names for files stored on remote machines in the `STORE_DIRECTORY`. For a list of possible strings, see Table 2-2 on page 100. 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, 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 "Configuring Daemon Database Record Length" on page 29.
`POLICIES`	For DCM disk MSP use only. See "Setting Up Disk MSPs in DCM Mode" on page 105.
`STORE_DIRECTORY`	Specifies the directory used to store files for this MSP.
`TASK_GROUPS`	Names the task groups that contain tasks the MSP should run. They are configured as objects of `TYPE taskgroup`. There is no default. Currently there are tasks defined only for the tape MSP.

The following example describes setting up a disk MSP:

```
define  dsk
        TYPE                    msp
        COMMAND                 dmdskmsp
        CHILD_MAXIMUM           8
        GUARANTEED_DELETES      3
        GUARANTEED_GETS         3
        STORE_DIRECTORY         /remote/dir
enddef
```

Procedure 2-19 Configuring the dsk Object

The following steps explain pertinent information for configuring the dsk object:

1. Ensure that define has a value that you set previously in the MSP_NAMES or LS_NAMES parameter of the daemon object. There is no default.
2. Ensure that TYPE is set to msp. There is no default.
3. Ensure that COMMAND is set to dmdskmsp. There is no default.
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 to the directory where files will be stored. This parameter is required; there is no default. (In DCM-mode, the directory specified must be a dedicated XFS or CXFS filesystem; see "Setting Up Disk MSPs in DCM Mode".)

Setting Up Disk MSPs in DCM Mode

To work with the DCM, the disk MSP requires the following:

- The STORE_DIRECTORY field of the configuration stanza for the MSP must be the mount point of a dedicated XFS or CXFS filesystem mounted with DMAPI enabled. See "Filesystem Mount Options" on page 27 for instructions.
- The configuration stanza must contain at least one POLICIES parameter and the configuration stanza for that parameter must contain a SELECT_LOWER_VGS parameter.
- There must also be a taskgroup that runs the run_dcm_admin script during off-peak hours to perform routine maintenance for the MSP.

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 "Configuring Daemon Database Record Length" on page 29.

When using DCM mode, 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.

```
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
        VALID_LOWER_VGS         vg1 vg2 old_vg                  # [See note 4]
        STORE_DIRECTORY         /dcm_cache                      # [See note 5]
        CHILD_MAXIMUM           10                              # [See note 6]
        POLICIES                dcm_policy
        TASK_GROUPS             dcm_tasks
enddef

define  dcm_policy
        TYPE                    policy                          # [See note 7]

        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:

1. The DCM must be specified before the LSs that contain its lower volume groups. (Otherwise, all recalls will attempt to come directly from tape.)

2. Other parameters essential to the use of this stanza but not relevant to DCM have been omitted.

3. The DCM and its lower volume groups should be considered to act as a single high-speed volume group 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 volume group, but could be any type of MSP other than a disk MSP in DCM mode.

   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_VGS` volume groups.

4. All volume groups that could have lower copies of this DCM must be listed, even if no longer in active use. A volume group can be removed from this list when it is known not to contain any dual-resident files still managed by the DCM. It is the presence of the `VALID_LOWER_VGS` parameter that enables DCM mode.

5. A **dedicated** DMAPI-mounted filesystem

6. Any other parameters applicable to a disk MSP may also be used, with the exception of `IMPORT_ONLY` and `IMPORT_DELETE`.

7. Several parameters in DCM policies have functions that are analogous to those in standard disk MSP policies; see "DCM Policies" on page 61.

Verifying the Configuration

To verify the DMF configuration, run the `dmcheck`(8) script. This command checks the configuration file object and parameters, and reports on inconsistencies.

Initializing DMF

The DMF daemon database is created in *HOME_DIR*/`daemon` as `dbrec.dat`, `dbrec.keys`, `pathseg.dat`, and `pathseg.keys`. The database definition file (in the same directory) that describes these files and their record structure is named `dmd_db.dbd`. The database journal file is named `dmd_db.`*yyyymmdd.*`[`*hhmmss*`]`. It is created in the directory *JOURNAL_DIR*/`daemon` (*JOURNAL_DIR* is specified by the `JOURNAL_DIR` configuration parameter).

The `inst`(8) utility on IRIX systems and the `rpm`(8) utility on Linux systems set up system startup and shutdown scripts to start and stop DMF. You can start and stop the DMF daemon manually by executing the following:

```
/etc/init.d/dmf start
/etc/init.d/dmf stop
```

You could also use the the `dmfdaemon`(8) and `dmdstop`(8) commands.

After `dmfdaemon` is activated, the `dmget`(1) and `dmput`(1) user commands can be used to manage filesystem space manually.

General Message Log File Format

The `dmfdaemon`, `dmlockmgr`, `dmfsmon`, MSP, and LS processes all create message files that are used to track various DMF events. These DMF 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:

- See "Automated Space Management Log File" on page 119, for information about `dmfsmon` and `autolog.`*yyyymmdd*
- See "Daemon Logs and Journals" on page 129, for information about `dmfdaemon` and `dmdlog.`*yyyymmdd*
- See "`dmlockmgr` Communication and Log Files" on page 131, for information about `dmlockmgr` and `dmlocklog.`*yyyymmdd*
- See "Tape MSP/LS Logs" on page 142, and "Activity Log" on page 169, for information about `dmatmsp`, `dmdskmsp`, and `dmftpmsp` and `msplog.`*yyyymmdd*
- See Chapter 7, "DMF Maintenance and Recovery" on page 187, for information about log file maintenance.

Messages in the `dmdlog`, `dmlocklog`, 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`, `msplog`, and `dmlocklog` are defined in Table 2-3. The table also lists the corresponding message levels in the configuration file.

Table 2-3 DMF Log File Message Types

Field	Message type	Message level
`E`	Error	0
`O`	Ordinary	0
`I`	Informative	1

Field	Message type	Message level
`V`	Verbose	2
`1`	Debug level 1	3
`2`	Debug level 2	4
`3`	Debug level 3	5
`4`	Debug level 4	6

Parameter Table

Table 2-4 on page 111, lists the parameters that can be specified in the `/etc/dmf/dmf.conf` file and the objects to which they apply.

Note: the most up-to-date list of parameters is in the `dmf.conf`(5) man page.

Legend:

BS: Base
DM: Daemon
DV: Device
DG: Device group
DP: Non-DCM Disk MSP
DC: DCM Disk MSP
FS: Filesystem
FP: FTP MSP
LS: Library server
PO: Policy
RS: Resource scheduler
RW: Resource watcher
TP: Tape MSP
TG: Task group
VG: Volume group

Table 2-4 Parameters for the DMF Configuration File

Parameter	BS	DM	DV	DG	DP	DC	FS	FP	LS	PO	RS	RW	TP	TG	VG
ADMIN_EMAIL	X														
AGE_WEIGHT										X					
ALGORITHM											X				
ALLOCATION_GROUP															X
ALLOCATION_MAXIMUM															X
ALLOCATION_MINIMUM															X
BLOCK_SIZE			X	X											
CACHE_AGE_WEIGHT										X					
CACHE_DIR									X				X		
CACHE_SPACE									X				X		
CACHE_SPACE_WEIGHT										X					
CHILD_MAXIMUM					X	X		X					X		
COMMAND					X	X		X	X				X		
DATABASE_COPIES														X	
DATA_LIMIT														X	
DISK_IO_SIZE				X	X	X		X					X		
DRIVES_TO_DOWN				X											
DRIVE_GROUPS									X						
DRIVE_MAXIMUM				X											X
DRIVE_SCHEDULER				X											
DUALRESIDENCE_TARGET										X					
DUMP_DEVICE														X	
DUMP_FILE_SYSTEMS														X	
DUMP_INVENTORY_COPY														X	
DUMP_MIGRATE_FIRST														X	

Parameter	BS	DM	DV	DG	DP	DC	FS	FP	LS	PO	RS	RW	TP	TG	VG
DUMP_RETENTION														X	
DUMP_TAPES														X	
FREE_DUALSTATE_FIRST										X					
FREE_SPACE_DECREMENT										X					
FREE_SPACE_MINIMUM										X					
FREE_SPACE_TARGET										X					
FTP_ACCOUNT								X							
FTP_COMMAND								X							
FTP_DIRECTORY								X							
FTP_HOST								X							
FTP_PASSWORD								X							
FTP_PORT								X							
FTP_USER								X							
GUARANTEED_DELETES					X	X		X							
GUARANTEED_GETS					X	X		X							
HFREE_TIME													X		X
HOME_DIR	X														
HTML_REFRESH												X			
IMPORT_DELETE					X			X							
IMPORT_ONLY					X			X							
JOURNAL_DIR	X														
JOURNAL_RETENTION														X	
JOURNAL_SIZE	X														
LABEL_TYPE			X	X											
LICENSE_FILE	X														
LOG_RETENTION														X	
LS_NAMES		X													

Parameter	BS	DM	DV	DG	DP	DC	FS	FP	LS	PO	RS	RW	TP	TG	VG
MAX_CACHE_FILE									X				X		
MAX_CHUNK_SIZE													X		X
MAX_MS_RESTARTS				X											
MAX_PUT_CHILDREN													X		X
MERGE_CUTOFF													X		X
MESSAGE_LEVEL		X			X	X	X	X	X				X		
MIGRATION_LEVEL		X				X	X								
MIGRATION_TARGET										X					
MIN_TAPES													X		
MIN_VOLUMES															X
MODULE_PATH											X				
MOUNT_SERVICE			X	X											
MOUNT_SERVICE_GROUP				X											
MOUNT_TIMEOUT				X											
MOVE_FS		X													
MSG_DELAY			X	X											
MSP_NAMES		X													
MVS_UNIT								X							
NAME_FORMAT					X	X		X							
OV_ACCESS_MODES			X	X											
OV_INTERCHANGE_MODES			X	X											
OV_KEY_FILE	X														
OV_SERVER	X														
PENALTY											X				
POLICIES						X	X								
POSITIONING			X	X											
POSITION_RETRY			X	X											

Parameter	BS	DM	DV	DG	DP	DC	FS	FP	LS	PO	RS	RW	TP	TG	VG
PUTS_TIME															X
READ_TIME															X
REINSTATE_DRIVE_DELAY				X											
REINSTATE_VOLUME_DELAY				X											
RUN_TASK														X	*
SELECT_LOWER_VG										X					
SELECT_MSP										X					
SELECT_VG										X					
SITE_SCRIPT										X					
SPACE_WEIGHT										X					
SPOOL_DIR	X														
STORE_DIRECTORY					X	X									
TAPE_TYPE													X		
TASK_GROUPS		X		X	X	X	X	X	X				X		X
THRESHOLD														X	
TIMEOUT_FLUSH													X		X
TMF_TMMNT_OPTIONS			X	X											
TMP_DIR	X														
VERIFY_POSITION			X	X											
VOLUME_GROUPS				X											
VOLUME_LIMIT														X	
WATCHER									X						
WEIGHT											X				
WRITE_CHECKSUM			X	X											
ZONE_SIZE			X												X

* The `run_tape_merge.sh` and `run_merge_stop.sh` tasks and their associated parameters can be specified in the volume group object.

Automated Space Management

The dmfsmon(8) daemon monitors the free space levels in filesystems configured as with automated space management is enabled (auto) and lets you maintain a specified level of free space. 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. Each of these candidates is assigned a file weight. This information is used to create a list, called a *candidate list*, that contains an entry for each file and is ordered by file weight (largest to smallest).
- Selects enough candidates to bring the free space back up to the desired level. Files are selected in order from largest file 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 file 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 file weighting. Only filesystems configured as MIGRATION_LEVEL auto in the configuration file are included in the space-management process. "DMF Policies" on page 52, describes how to configure automated space management.

You can change the migration level of a filesystem by editing the configuration file.

The following sections describe space management and associated processes:

- "Generating the Candidate List" on page 116
- "Selection of Migration Candidates" on page 116
- "Space Management and the Disk Cache Manager" on page 119

- "Automated Space Management Log File" on page 119

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 file weighting policy in the DMF configuration file. The applicable file weighting policy determines a file's total weight. Total file 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 weight (largest to smallest). You can configure the weighting parameters to have a negative value and ensure that certain files are never automatically migrated.

Note: If you use negative weights to exclude files from migration, you must ensure that a filesystem does not fill with files 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 "DMF Policies" on page 52.

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 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 fully migrated and releases their disk blocks. It then selects additional candidates to meet the MIGRATION_TARGET and migrates them.

Figure 3-1 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.

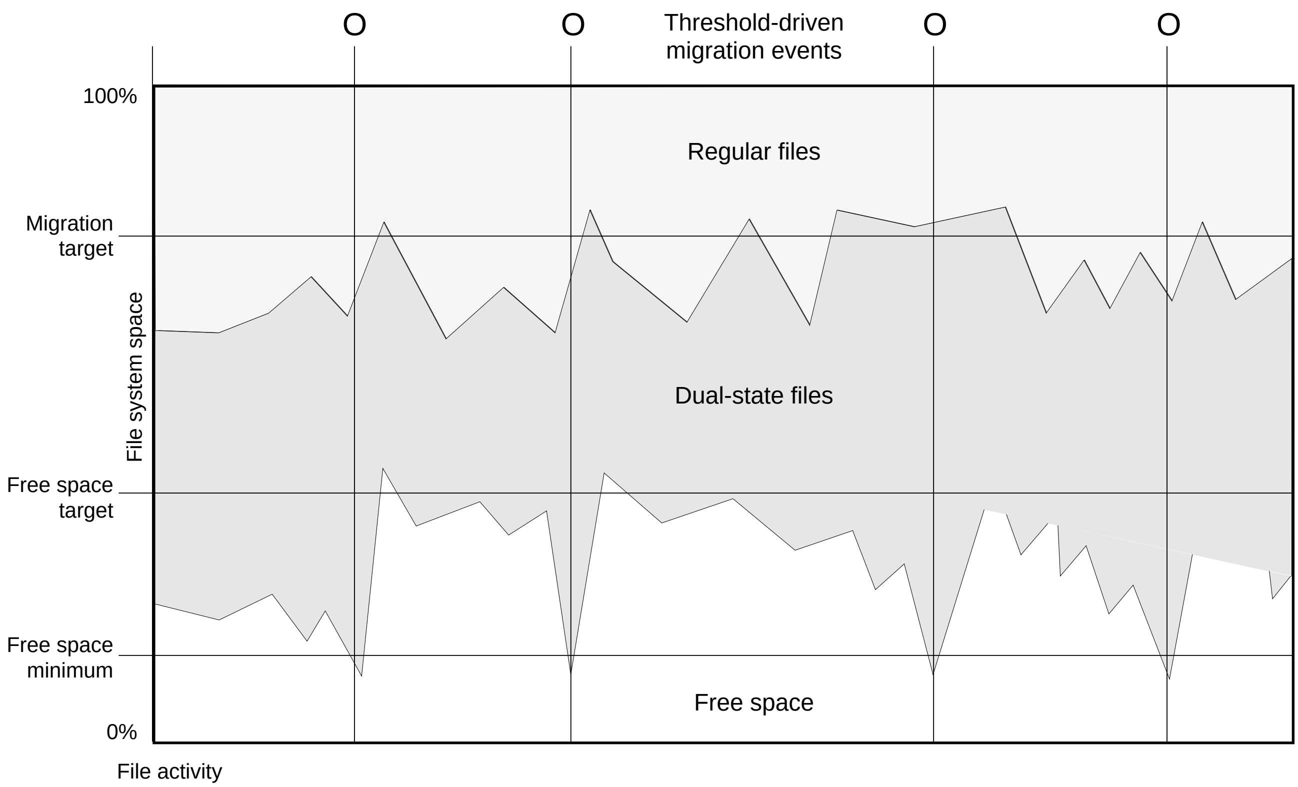

Figure 3-1 Relationship of Automated Space Management Targets

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 disk-MSP-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 similar to disk MSP, although some names are different. See "DCM Policies" on page 61.

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 "Configuring the Base Object" on page 38). 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 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 show the format of an autolog file:

```
11:44:55-V 26968-dmfsmon /dmi - free_space=5.44, minimum=5
11:46:55-V 26968-dmfsmon /dmi - free_space=5.12, minimum=5
11:47:35-I 26968-dmfsmon Started 15135 for execution on /dmi
11:48:55-V 26968-dmfsmon /dmi - free_space=4.79, minimum=5
11:49:48-I 15135-dmfsmon Number of blocks in the filesystem = 17769424
11:49:48-I 15135-dmfsmon Number of blocks in the migration target = 8884712 (50%)
11:49:48-I 15135-dmfsmon Number of blocks currently migrated = 16428664 (92.5%)
11:49:48-I 15135-dmfsmon Number of blocks to migrate = 0 (0.0%)
11:49:48-I 15135-dmfsmon Number of blocks in the free space target = 1776942 (10%)
11:49:48-I 15135-dmfsmon Number of blocks currently free = 886824 (5.0%)
11:49:48-I 15135-dmfsmon Number of blocks to free = 890118 (5.0%)
11:49:48-I 15135-dmfsmon Summary of files: online = 93050, offline = 342836, unmigrating = 0.
11:49:48-I 15135-dmfsmon Number of candidates = 93050, rejected = 0
11:50:55-V 26968-dmfsmon /dmi - free_space=7.26, minimum=5
11:51:49-I 15135-dmfsmon Migrated 272 blocks in 1 files.
11:51:49-I 15135-dmfsmon Freed 890184 blocks in 4197 files
11:51:49-O 15135-dmfsmon Exiting: minimum reached - targets met by outstanding requests.
11:52:55-V 26968-dmfsmon /dmi - free_space=9.73, minimum=5
11:54:55-V 26968-dmfsmon /dmi - free_space=9.73, minimum=5
```

The DMF Daemon

The DMF daemon, `dmfdaemon`(8), is the core component of DMF. The daemon exchanges messages between itself and commands, the MSPs and LSs, and the kernel. It also assigns bit file identifiers (BFIDs) to migrated files and maintains the DMF database entries for offline copies.

When DMF is started, the daemon database is automatically initialized. To start the daemon manually, use the DMF startup script, as follows:

```
/etc/init.d/dmf start
```

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.

After DMF is activated, the `dmget`(1) and `dmput`(1) user commands can be used to manage filesystem space manually.

The following sections provide additional information about the daemon database and daemon processing.

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".

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 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 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.

DMF Daemon Database and `dmdadm`

The DMF daemon maintains a database that resides in the directory *HOME_DIR*/*daemon_name* (*HOME_DIR* is specified by the `HOME_DIR` configuration parameter). This database contains information about the offline copies of a given file, as well as some information about the original file. The 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 name and recall path
- Date and time information, including the following:
  - Time at which the database record was created
  - Time at which the database 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.

`dmdadm` Directives

The `dmdadm` directives are as follows:

Directive	**Description**
`count`	Displays the number of records that match the expression provided.
`create`	Deletes an existing database record.

`dump`	Prints the specified database records to standard out in ASCII; each database field is separated by the pipe character (`|`).
`help`	Displays help.
`list`	Shows the fields of selected database records. You may specify which fields are shown.
`load`	Applies records to the 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 an existing database record.

The syntax for the `dmdadm` directives is summarized as follows:

`count` *selection* [*limit*]
`delete` *selection* [*limit*]
`dump` *selection* [*limit*]
`help`
`list` *selection* [*format*]
`load` *filename*
`quit (or q or exit)`
`set` [*format*]
`update` *selection* [*limit*] `to` *fields*`...`

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 `<` (less than), `>` (greater than), `=` (equal to), `<=` (less than or equal to), or `>=` (greater than or equal to) to compare a field keyword to an appropriate value.

The syntax for *selection* is as follows:

```
selection      ::=   or-expr
or-expr        ::=   and-expr [ or or-expr ]
and-expr       ::=   nested-expr [ and and-expr ]
nested-expr    ::=   comparison | ( expression )
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
mspkey 456 to origuid 2570
```

`dmdadm` Field and Format Keywords

The *field* keywords listed below specify new values for fields. Some of the keywords are valid only if you also specify the `-u` option.

Keyword	**Description**
`checkage` (`ca`)	The time at which the database 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 database record was last checked; an integer that reflects raw UNIX time. Valid only in unsafe (`-u`) mode.
`deleteage` (`da`)	The time at which the database 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 database record was soft-deleted; an integer that reflects raw UNIX time. Valid only in unsafe (`-u`) mode.
`mspname` (`mn`)	The name of the MSP or volume group 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 volume group can use to recall a database record; a string of up to 50 characters. Valid only in unsafe (`-u`) mode.
`origage` (`oa`)	Time at which the database 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 database record was created; an integer that reflects raw UNIX time.
`origuid` (`ou`)	Original user ID of the database record; an integer.
`updateage` (`ua`)	Time at which the database record was last updated; the same as `updatetime`, except that it is specified as an age string.
`updatetime` (`ut`)	Time at which the database record was last updated; an integer that reflects raw UNIX time.

The time field keywords (`checktime`, `deletetime`, `origtime`, and `updatetime`) have a value of either `now` or raw UNIX time (seconds since January 1, 1970). These keywords display their value as raw UNIX 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 in a form such as `8w12d7h16m20s`, meaning 8 weeks, 12 days, 7 hours, 16 minutes, and 20 seconds old. The age keywords display their value as an integer followed by w, d, h, m, or s (weeks, days, hours, minutes, and seconds, respectively). The comparison > used with the age keywords means older than the value given (that is, >5d is older than 5 days).

The *limit* keywords restrict the records acted upon:

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 either bfid or data. bfid specifies that the records are scanned in BFID order. data specifies that the records are scanned in the order in which they are found in the database data file. data is more efficient for large databases, although it is essentially unordered.

The *format* keyword 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 keyword. Values for format can be default, keyword, or a list of field keywords enclosed in quotation marks.

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 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 DMF 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 "Configuring Daemon Maintenance Tasks" on page 44.

The DMF daemon automatically creates journal files that track 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 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 "Configuring Daemon Maintenance Tasks" on page 44.

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 192.

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 dmatmsp, dmatls, dmfdaemon(8), dmvoladm(8), and dmcatadm(8)) communicate through various methods. These methods include 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 Chapter 2, "Configuring DMF" on page 25.

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
- Individual transaction log files

dmlockmgr Communication and Log Files

The dmlockmgr communication and log files are all found in a directory formed by *SPOOL_DIR*/RDM_LM. This directory contains 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 token files in *SPOOL_DIR*/RDM_LM have the form shown in Table 5-1 on page 132.

Table 5-1 `dmlockmgr` Token Files

File	Description
`dmlockmgr`	Used by the `dmlockmgr` and its clients to access `dmlockmgr`'s semaphore and input message queue
`dmatmsp`*msp_or_ls_name*	Used by the MSP/LS *msp_or_ls_name* and `dmlockmgr` to access the MSP's or LS's input message queue
`dmfdaemon`*daemon*	Used by the DMF *daemon* and `dmlockmgr` to access the daemon's input message queue
Process ID files:	Used by the process whose process ID is *PID* to access the process's input message queue
`dmatread`*PID*	
`dmatsnf`*PID*	
`dmcatadm`*PID*	
`dmdbrecover`*PID*	
`dmdbase`*PID*	
`dmvoladm`*PID*	

The `dmlockmgr`, `dmatmsp`, `dmatls`, and `dmfdaemon` token files are limited in number, and they change infrequently. If a `dmlockmgr`, `dmatmsp`, or `dmfdaemon` terminates without removing the file, an existing token file will be used on restart. If a `dmatmsp`, `dmatls`, or `dmfdaemon` fails to remove the file and the MSP or LS name is changed, the file will remain until it is manually removed.

The files of the *PID* versions listed in Table 5-1 are removed from the `lockmgr` directory automatically when the command terminates or when the DMF daemon initializes. Do not create files of this name format in this directory because the daemon is likely to remove them.

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. It will not remove the token file.

Note: Normally, the `dmlockmgr` process is terminated as part of normal shutdown procedures. However, if you wish to stop it manually, you must kill the process by using `kill`(1). Killing the `dmlockmgr` process does not remove the `dmlockmgr` IPC resources or token file. If the `dmlockmgr` is restarted automatically by a DMF process, it will reuse the token file and IPC resources it left behind.

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 `dmatmsp/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:

SPOOL_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 and you should periodically remove older versions. You should maintain the `dmlockmgr` log files for as long as you maintain the database transaction journal files.

`dmlockmgr` Individual Transaction Log Files

The individual transaction log files have the following form:

`dmatmsp`*msp_or_ls_name*`.log`
`dmfdaemondaemon.log`
`dmvoladm`*PID*`.log`
`dmcatadm`*PID*`.log`
`dmdbase`*PID*`.log`
`dmdbrecover`*PID*`.log`
`dmselect`*PID*`.log`

Most of the transaction log files will reside in the database directory (*HOME_DIR*/*daemon_name* for the `dmfdaemon`, *HOME_DIR*/*msp_name* for the `dmatmsp`, *HOME_DIR*/*ls_name* for the `dmatls`). In the case of the `dmfdaemon`, `dmatmsp`, and `dmatls`, each new transaction will reuse the same file generated by the last transaction, and there is no need to remove these files.

In the case of the *PID* transaction log files, the commands that generate them will generally remove them during their normal exit processing code. If there is an abnormal termination, these files will not be removed, and they may be quite large.

Caution: Do **not** delete any orphaned transaction log files until you are sure the database is not actively in use. If a process aborts during a committed but incomplete transaction, the next process that contacts the `dmlockmgr` will use the information in the transaction log file to recover the incomplete transaction.

After you are sure the transaction log file will not be needed, it can be removed.

It is wise to periodically check for these files. Several DMF commands allow accessing of copies of database files in places other than the standard location, which may result in unnecessary transaction log files consuming disk space.

The transaction activity file, *SPOOL_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 *SPOOL_DIR*`/RDM_LM/vista.taf` file.

Media-Specific Processes and Library Servers

Media-specfic processes (MSPs) and library servers (LSs) migrate files from one media to another. There are the following types of MSPs:

- Tape MSP, which copies files from a disk to tape, or copies files from tape to disk.
- File transfer protocol (FTP) MSP, which allows the DMF daemon to manage data by moving it to a remote machine.
- Disk MSP, which migrates data to a directory that is accessible on the current systems and would be a cache disk if used in disk cache manager (DCM) mode.

LSs, like tape MSPs, copy files from a disk to a tape or from a tape to a disk. However, although the tape MSP and the LS have many characteristics in common, one of the primary differences is that while the tape MSP can manage at most one active copy of a migrated file, the LS can manage more than one copy. An LS is comprised of one or more volume groups. When a file is migrated from disk to tape, the selection policy can specify that it be copied to more than one volume group. Each volume group can manage at most one copy of a migrated file. Each volume group has an associated pool of tapes. Data from more than one volume group is never mixed on a tape.

This chapter discusses the following:

- "Tape MSP and LS Operations"
- "FTP MSP" on page 168
- "Disk MSP" on page 171
- "Disk MSP and Disk Cache Manager (DCM)" on page 173
- "Moving Migrated Data between MSPs and Volume Groups" on page 173
- "Converting from an IRIX DMF to a Linux DMF" on page 174
- "Converting from a Tape MSP to a Library Server" on page 178
- "Library Server Error Analysis and Avoidance" on page 181
- "Library Server Drive Scheduling" on page 183
- "Library Server Status Monitoring" on page 184

Tape MSP and LS Operations

The tape MSP consists of the following programs:

- `dmatmsp`
- `dmatwc`
- `dmatrc`

The DMF daemon executes `dmatmsp` as a child process. The MSP communicates with the daemon through a pair of unnamed pipes. In turn, `dmatmsp` executes `dmatwc` (the write child) to write data to tape and `dmatrc` (the read child) to read data from tape.

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 `dmatmsp` or the `dmatls` program maintains the following types of records in its database:

- Catalog (CAT) records, which contain information about the files the MSP/LS maintains
- Volume (VOL) records, which contain information about the media the MSP/LS uses

The database is not a text file and cannot be updated by standard utility programs. Detailed information about the database and its associated utilities is provided in "CAT Database Records" on page 140, and "VOL Database Records" on page 140.

The tape MSP/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* or *compression*. Data on MSP/LS volumes becomes obsolete when users delete or modify their files. Volume merging can be configured to occur automatically (see "Configuring Maintenance Tasks for Tape MSP and LS" on page 91). It can also be triggered by marking MSP/LS volumes as sparse with the `dmvoladm`(8) command.

The tape MSP/LS provides two utilities that read MSP/LS volumes directly:

- `dmatread`(8), which copies all or part of a migrated file to disk
- `dmatsnf`(8), which audits and verifies MSP/LS volumes

Tape MSP/LS Directories

Each instance of the tape MSP/LS needs three types of directories, one for each of the following:

- Databases
- 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 tape MSP/LS creates a subdirectory named after itself in each of these three directories.

For example, if an instance of the tape MSP is called `cart1`, its database files reside in directory *HOME_DIR*`/cart1`. If another instance of the tape MSP 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, MSP `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 tape MSP/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, `dmatmsp` and `dmatls` use 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 a 3480/3490 device is 65,536 bytes.

- A *chunk* is as much or as little of a user file as fits on the remainder of the tape (see Figure 6-1 on page 139). 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 several physical blocks ending with a tape mark. A zone has a target size that is configurable by media type. The default zone target size is 50 MB.

  The MSP or volume group writes chunks into the zone until one of three conditions occurs:

  – The zone size is exceeded

  – The MSP or volume group 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 `dmatmsp/dmatls` to start a process to write files to tape.

  – The MSP/LS records the position of the beginning of each zone in its database so that it can use fast hardware positioning functions to return there to restore the chunks in that zone.

  – When a tape volume develops a defect, the data loss usually will be restricted to the 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 6-1 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.

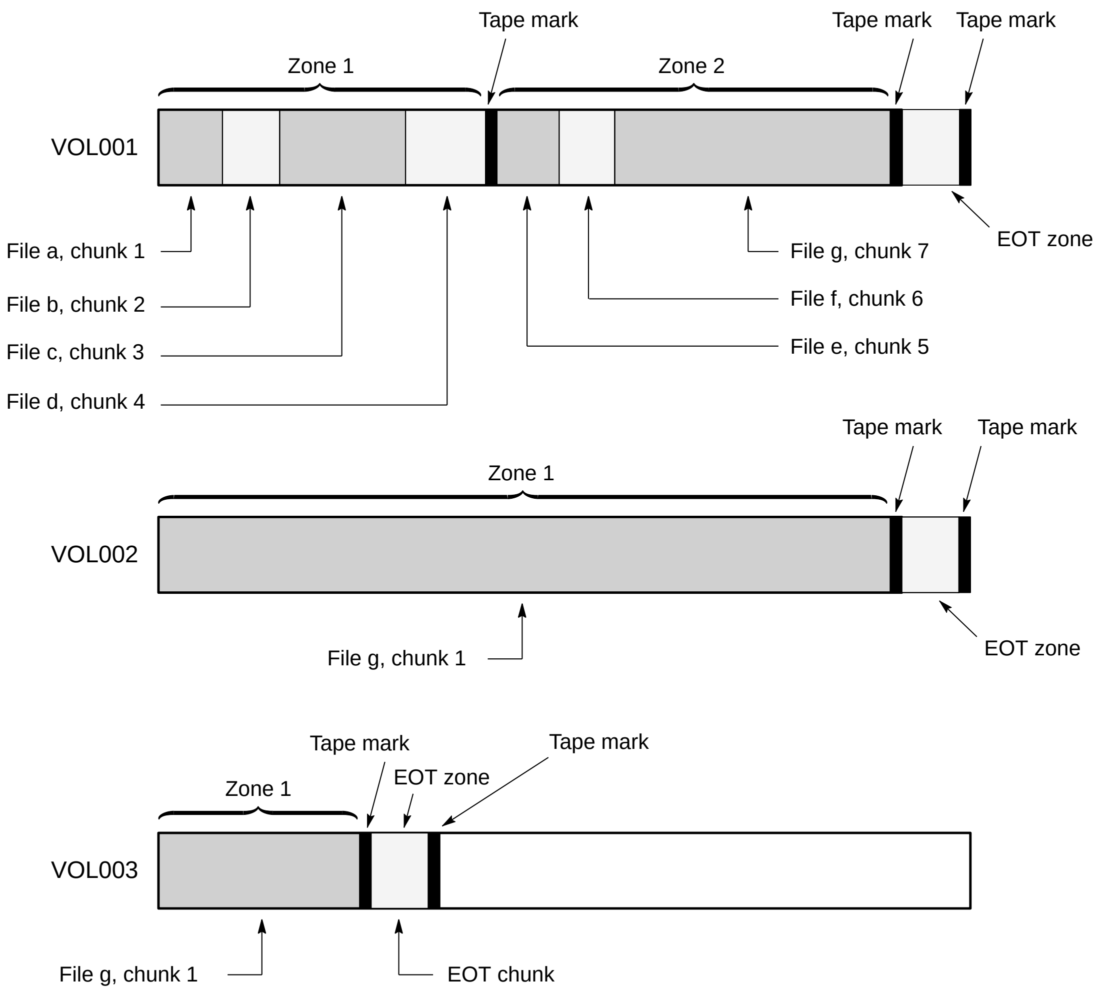

Figure 6-1 Media Concepts

CAT Database Records

Records in the tape catalog (CAT), `tpcrdm`, 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).

You do not explicitly create CAT records in the database. They are created when files migrate.

The CAT portion of the MSP/LS database consists of the following files:

- `tpcrdm.dat`, which contains the data records themselves
- `tpcrdm.key1.keys` and `tpcrdm.key2.keys`, which contain the indexes to those records

For MSPs, the database definition file (in the same directory) that describes these files and their record structure is named `atmsp_db.dbd`. For LSs, it is named `libsrv_db.dbd`.

All files are non-ASCII and cannot be maintained by standard utility programs. The `dmcatadm` command provides facilities to create, query, and modify CAT database records (see "`dmcatadm` Command" on page 147).

Note: The ability to create or modify CAT database records with `dmcatadm` is provided primarily for testing purposes. In the normal course of operations, you would never use this capability.

VOL Database Records

Records in the tape volume (VOL) portion of the MSP/LS database, `tpvrdm`, contain information about each volume that exists in the pool of tapes to be used by `dmatmsp` or `dmatls`. These records are indexed by the volume serial number (VSN) of each volume and contain such information as the volume's type, estimated capacity, label type, and a number of flags indicating the state of the volume. For LSs, the record also contains the volume's volume group or allocation group. Unlike the CAT records, you must create the VOL records in the database before using `dmatmsp` or `dmatls` for the first time.

The VOL portion of the MSP/LS database consists of two files:

- `tpvrdm.dat`, which contains the volume records themselves

- tpvrdm.vsn.keys, which contains the indexes to the records

For MSPs, the database definition file (in the same directory) that describes these files and their record structure is named atmsp_db.dbd. For LSs, it is named libsrv_db.dbd.

Both files contain binary data and require special maintenance utilities. The dmvoladm command, described in more detail in "dmvoladm Command" on page 155, provides facilities to create, query, and modify VOL records in the database. Additional database maintenance utilities are described in "Database Recovery" on page 192.

Note: If you have more than one instance of the tape MSP or volume group, you must ensure that the volume sets for each are mutually exclusive.

Tape MSP/LS Journals

Each instance of dmatmsp or dmatls protects its database by recording every transaction in a journal file. For MSPs, journal file pathnames have the following format:

JOURNAL_DIR/*msp_name*/atmsp_db.*yyyymmdd[.hhmmss]*

For LSs, journal file pathnames have the following format:

JOURNAL_DIR/*ls_name*/libsrv_db.*yyyymmdd[.hhmmss]*

The MSP/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*/*msp_name* directory on 15 June 1998:

```
atmsp_db.19980527.235959  atmsp_db.19980606.235959
atmsp_db.19980528.235959  atmsp_db.19980607.235959
atmsp_db.19980529.235959  atmsp_db.19980608.235959
atmsp_db.19980530.235959  atmsp_db.19980609.235959
atmsp_db.19980531.235959  atmsp_db.19980610.235959
atmsp_db.19980601.235959  atmsp_db.19980611.235959
atmsp_db.19980602.235959  atmsp_db.19980612.235959
atmsp_db.19980603.235959  atmsp_db.19980613.235959
atmsp_db.19980604.235959  atmsp_db.19980614.235959
atmsp_db.19980605.235959  atmsp_db.19980615
```

For every date on which 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 "Configuring Daemon Maintenance Tasks" on page 44.

If an MSP/LS database becomes corrupt, recovery consists of applying the journal files to a backup copy of the database.

Tape MSP/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 tape 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 tape MSP/LS and contains most of the messages. This file is written by `dmatmsp`, `dmatls`, `dmatrc`, and `dmatwc`. A new `msplog.`*yyyymmdd* is created for each day.

This section describes informational statistics provided by the tape log files. These messages appear in the *SPOOL_DIR*/*msp_name*/`msplog.`*yymmdd* 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 MSP/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 "Configuring Daemon Maintenance Tasks" on page 44.

Example 6-1 Tape MSP Statistics Messages

The following is an example of tape MSP statistics messages taken from an `msplog.`*yyyymmdd* file. These messages are automatically and periodically issued by the MSP.

```
09:06:00-I 18979429-dmatmsp stats: children=3/3/0/4, btp=92274688/130740224/250609687, wc=2/4, cwc=0
09:06:00-I 18979429-dmatmsp stats: data put=722.469 mb, data recalled= 45.089 mb
09:06:00-I 18979429-dmatmsp stats:    Put_File -      6     33      0      6
09:06:00-I 18979429-dmatmsp stats:    Get_File -      1     13      0      0
09:06:00-I 18979429-dmatmsp stats: Delete_File -      0      1      0      0
09:06:00-I 18979429-dmatmsp stats:  Cancel_Req -      0      6      0      0
09:06:00-I 18979429-dmatmsp stats:    Flushall -      0      8      3      0
09:06:00-I 18979429-dmatmsp stats:       Merge -     20     14      0      0
09:06:00-I 18979429-dmatmsp stats: mc=4, ms=0, mu=0, sm=1
```

The information provided by these entries is defined as follows:

- `children=3/3/0/4` represents the total child processes (3), the active child processes (3), the clean processes running (0), and the configured value of `CHILD_MAXIMUM` (4). Clean children are used when a `dmatrc` or `dmatwc` process dies without cleaning up.
- `btp=92274688/130740224/250609687` represents the bytes queued for putting (92274688), the threshold at which to start the next put child (130740224), and the bytes assigned to socket I/O (250609687)

- `wc=2/4` represents the active write child processes (2) and the configured value of `MAX_PUT_CHILDREN` (4)
- `cwc=0` represents the process ID of the current write child (that is, the write child that is accepting data to write). 0 represents none.

The next line gives the total amount of data put (722.469 megabytes) and recalled (45.089 megabytes).

The next six lines provide statistics for each type of MSP request. Statistics information is provided only for requests that have been issued since the MSP 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 line provides the following information:

- `mc` is the configured value for `MERGE_CUTOFF`, the cutoff to stop scheduling tapes for merging (4)
- `ms` is the configured value for `CACHE_SPACE`, the merge cache space available (0 bytes)
- `mu` is the merge cache space used (0 bytes)
- `sm` is the number of socket merge children (1)

Example 6-2 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.

```
00:02:00-I 13902144-dmatls  vg9a16.stats: children=1/0/0/7, btp=28098297/0/0, wc=0/7, cwc=0
00:02:00-I 13902144-dmatls  vg9a17.stats: children=1/0/0/7, btp=59032803/0/0, wc=0/7, cwc=0
00:02:00-I 13902144-dmatls  vg9a16.stats: data put=608.607 mb, data recalled=114.270 mb
00:02:00-I 13902144-dmatls  vg9a17.stats: data put=1068.423 mb, data recalled=210.575 mb
00:02:01-I 13902144-dmatls  vg9a16.stats:    Put_File -     10    172      0     12
00:02:01-I 13902144-dmatls  vg9a16.stats: Delete_File -      0    130      0      0
00:02:01-I 13902144-dmatls  vg9a16.stats:  Cancel_Req -      0     12      0      0
00:02:01-I 13902144-dmatls  vg9a16.stats:    Flushall -      0      2      0      0
```

```
00:02:01-I 13902144-dmatls  vg9a16.stats:       Merge -     45     25      0     16
00:02:01-I 13902144-dmatls  vg9a17.stats:    Put_File -     14    210      0      8
00:02:01-I 13902144-dmatls  vg9a17.stats:    Get_File -      0      1      0      0
00:02:01-I 13902144-dmatls  vg9a17.stats: Delete_File -      0    178      0      0
00:02:01-I 13902144-dmatls  vg9a17.stats:  Cancel_Req -      0      8      0      0
00:02:01-I 13902144-dmatls  vg9a17.stats:    Flushall -      0      2      0      0
00:02:01-I 13902144-dmatls  vg9a17.stats:       Merge -     18     28      0     22
00:02:01-I 13902144-dmatls  vg9a16.stats: mc=7, ms=500000000, mu=133107712, sm=0
00:02:01-I 13902144-dmatls  vg9a17.stats: mc=7, ms=500000000, mu=73105408, sm=0
```

The fields in each message have the same meanings as in the MSP statistics messages (see Example 6-1 on page 143), except that they are on a volume group basis.

The tape MSP/LS write child (`dmatwc`) and read child (`dmatrc`) also produce statistics messages in the MSP/LS log 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 simply *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` task (see "Configuring Maintenance Tasks for Tape MSP and LS" on page 91).

Volume merging can also be done manually. `dmatmsp/dmatls` perform merge operations whenever sparse volumes and the necessary resources exist at the same time. Use the `dmvoladm select` directive to mark MSP or volume group volumes as sparse. (The `select` directive is described in "`dmvoladm` Command" on page 155.) 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 `dmatmsp/dmatls` utilities can perform volume-to-volume merging. Volume-to-volume merging is accomplished by moving data across a socket connection between the MSP/LS tape read-child and the MSP/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 MSP/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 MSP or volume group 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 MSP or volume group simultaneously if they share a tape device. If you initiate a merge process on more than one MSP or volume group 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_tape_merge.sh` task, 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 MSPs:

```
tapes=$(dmvoladm -m msp1 -c "count hsparse")
if [[ -z "$tapes" ]]; then
        # start merge on msp2
        dmvoladm -m msp2 -c "select hfull threshold<=30"
fi
```

`dmcatadm` Command

The `dmcatadm`(8) command provides maintenance services for CAT records in the MSP/LS database.

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 and MSP/LS databases from updates.

Note: Most of these facilities, especially the ability to create and modify CAT database records, are intended primarily for testing 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 an existing CAT record.
`dump`	Prints the specified CAT records to standard out in ASCII; each database 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 MSP/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 an existing CAT record.
`verify`	Verifies the MSP/LS database against the `dmfdaemon` database.

The first parameter of most directives specifies the database 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` *key field*`...`
`delete` *selection* [*limit*]
`dump` *selection* [*limit*]
`help`
`list` *selection* [*limit*] [*format*]
`load` *filename*
`quit (or q or exit)`
`set` [*format*]
`update` *selection* [*limit*] `to` *fields*`...`

For MSPs:

`verify` *selection* [*entries*] [*mspname*] [*limit*]

For LSs:

`verify` *selection* [*entries*] [*vgnames*] [*limit*]

The value for *key* may be a bit file identifier (BFID) designator in the form of a hexadecimal number.

The value for *selection* can be one of the following:

- A *key* or range of *keys* in the form *key* [`-`] [*key*]. *key*`-` specifies all records starting with *key*, and `-`*key* specifies all records up to *key*.
- 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 < (less than), > (greater than), = (equal to), <= (less than or equal to), or >= (greater than or equal to) to compare a field keyword to an appropriate value.

The syntax for *selection* is as follows:

```
selection      ::=  or-expr
or-expr        ::=  and-expr [ or or-expr ]
and-expr       ::=  nested-expr [ and and-expr ]
nested-expr    ::=  comparison | ( expression )
comparison     ::=  key-range | field-keyword op field-value
op             ::=  < | > | = | >= | <=
key-range      ::=  key [ - key] | [key - [key]] | 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 key
```

Thus valid *selection*s could be any of the following:

```
1510-1514
10000000000-
-15138
```

```
all
chunkoffset>0
chunknumber>0 and writeage<5d
. recordorder data
vsn=S07638
```

`dmcatadm` Keywords

The *field* keywords listed below specify new values for fields. Some of the keywords are valid only if you also specify the `-u` option (*unsafe mode*), which allows changes to most database fields and supports the use of `dmcatadm` to repair database damage but must be used with care.

Keyword	Description
`chunkdata` (`cd`)	Specifies the actual number of bytes written to tape by the MSP or volume group 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. Valid only as part of *selection*.
`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 in LS databases.
`filesize` (`fs`)	The original file size in bytes, an integer. This is valid only in unsafe (`-u`) mode.
`flags` (`fl`)	For future use.

`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 time.
`volgrp` (`vg`)	The volume group name. This keyword is valid for LSs only.
`vsn` (`v`)	The volume serial numbers; a list of one or more 6-character alphanumeric volume serial numbers separated by colons (`:`).
`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 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*/*hexinteger*, designating a zone number and block ID. A value of zero is used for *hexinteger* if no block ID is known. *integer* is the same as `zonenumber`, and *hexinteger* 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 time (seconds since January 1, 1970). These keywords display their value as raw UNIX 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*, a string in a form such as `8w12d7h16m20s` (meaning 8 weeks, 12 days, 7 hours, 16 minutes, and 20 seconds old). The age keywords display their value as an integer followed by `w`, `d`, `h`, `m`, or `s` (weeks, days, hours, minutes, and seconds, respectively). The comparison `>` used with the age keywords means older than the value given (that is, `>5d` is older than 5 days).

The *limit* keywords limit the records acted upon:

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 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; a string.
`mspname` (`mn`)	Specifies the name of the MSP associated with the record; a string.
`vgnames` (`vn`)	Specifies the names of the volume groups associated with the record; a quoted, space-separated list of names.

The *format* keyword 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` keyword. Values for `format` can be `default`, `keyword`, or a list of field keywords enclosed in quotation marks.

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 192.

Example 6-3 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 volume groups within this LS. The output from the dmvoladm list directive that follows shows that VSN 000700 is assigned to the volume group named vg8a15, and VSN 00727 is assigned to the volume group 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; only for LS)

`dmvoladm` Command

The `dmvoladm`(8) command provides maintenance services for VOL records in the MSP/LS database. In addition to the creation and modification of volume records, `dmvoladm` has an important role in the recovery of VOL records from a 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 and MSP/LS databases 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 an existing VOL record.
`dump`	Prints the specified VOL records to standard output in ASCII. Each database 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 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.

`repair`	Causes `dmvoladm` to adjust the usage information for specified volumes based on CAT data in the 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 an existing VOL record.
`verify`	Verifies the MSP databases against the `dmfdaemon` databases.

The syntax for the `dmvoladm` directives is summarized as follows:

`count` *selection*
`create` *vsnlist* [*field*`...`]
`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` *field*
`verify` *selection*

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 may consists entirely of letters, entirely of digits, or may be 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 `hflags`, whose meanings are as follows:

Flag	Description
`all`	Specifies all volumes in the database
`empty`	Specifies all volumes in which data written is 0
`partial`	Specifies used volumes in which `hfull` is off
`used`	Specifies all volumes in which data written is not 0

The syntax for *selection* is as follows:

selection	`::=`	*or-expr*
or-expr	`::=`	*and-expr* `[ or` *or-expr* `]`
and-expr	`::=`	*nested-expr* `[ and` *and-expr* `]`
nested-expr	`::=`	*comparison* `| (` *expression* `)`
comparison	`::=`	*vsnlist* `|` *field-keyword op field-value*
op	`::=`	`< | > | = | >= | <=`
key-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 *selection*s could be any of the following:

```
tape01-tape02
tape50-
-vsn900
all
herr or hbadmnt
used and hfull=off
datawritten>0 and hfull=off
. and eotchunk>3000 and (eotchunk<3500 or hfree=on)
hfull and threshold<30
```

`dmvoladm` Keywords

The *field* keywords specify new values for fields:

Keyword	Description
`blocksize` (`bs`)	Specifies the data block size in bytes when the tape was first written; an integer. The default is 65,536. This keyword is used only when mounting tapes with

	existing valid data. When an empty tape is first written, the MSP or volume group uses the default value for the tape type, unless it is overridden by a value in the `BLOCK_SIZE` parameter for the tape device 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 hexinteger. 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*/*hexinteger*, designating a zone number and block ID. A value of zero is used for *hexinteger* if no block ID is known. *integer* the same as `eotzone`, and *hexinteger* 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.
`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 cannot be displayed or updated.
`upage` (`ua`)	(Display only.) 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 valid only in unsafe (`-u`) mode.
`update` (`ud`)	(Display only.) Specifies the date and time of the last update to the volume's database record, expressed as an integer that reflects raw UNIX time. This is valid only in unsafe (`-u`) mode.
`version` (`v`)	Specifies the DMF tape format version, an integer. This is valid only in unsafe (`-u`) mode.
`volgrp` (`vg`)	Specifies the volume group or allocation group. This field is valid only for LS databases.
`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 time. This is valid only in unsafe (`-u`) mode.

The date field keywords (`update` and `wfdate`) have a value of either `now` or raw UNIX time (seconds since January 1, 1970). These keywords display their value as raw UNIX 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*, a string in a form such as `8w12d7h16m20s` (meaning 8 weeks, 12 days, 7 hours, 16 minutes, and 20 seconds old). The age keywords display their value as an integer followed by `w`, `d`, `h`, `m`, or `s` (weeks, days, hours, minutes, and seconds, respectively). The comparison `>` used with the age keywords means older than the value given (that is, `>5d` is older than 5 days).

The *limit* keywords restrict the volumes acted upon:

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 database, which is fastest but essentially unordered.

The *format* keyword 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` keyword. Values for `format` can be `default`, `keyword`, or a list of field keywords enclosed in quotation marks.

The *flag* keywords change the settings of hold flags:

Keyword	Description
`hbadmnt` (`hb`)	Indicates that the volume could not be mounted. If the problem causing the mount to fail is transient, the MSP will clear the flag the next time it attempts to mount the tape and succeeds. Typically this flag indicates a permanent condition that should be investigated and corrected. It is displayed as `--------b`. Currently, this flag is used only by MSPs.
`herr` (`he`)	Indicates that an I/O error has occurred on the volume; displayed as `e--------`. Currently, this flag is used only by MSPs.
`hflags` (no abbreviation)	(Display only.) 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--. For LSs, this flag can also be set if error conditions indicate that no more data should be written to it.
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 a period of time has passed. Currently used only by the LS. Displayed as ----l----.
hoa (ho)	Indicates that the volume is not to be used for either input or output, displayed as --o------.
hro (hr)	Indicates that the volume is read-only, displayed as ---r-----; this inhibits the MSP from using the volume for output.
hrsv (h*)	Currently unused (reserved); displayed as ----*----. This is valid only in unsafe (-u) mode.
hsparse (hs)	Indicates that the volume is considered sparse and thus a candidate for a volume merge operation, displayed as -------s-.

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 database recovery, see "Database Recovery" on page 192. For details about dmvoladm syntax, see the man page.

Example 6-4 `dmvoladm list` directives

The following example illustrates the default format for the `list` directive when using an MSP. 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 the string `efor*lmusb`, representing `herr`, `hfree`, `hoa`, `hro`, `hrsv`, `hlock`, `hfull`, `hsparse`, and `hbadmnt`, respectively.

```
adm 1>list S03232-S03254
                                     DATA    EOT            WR/FR
VSN    LB      DATA LEFT        WRITTEN  CHUNK   HFLAGS      AGE
------------------------------------------------------------------
S03232 sl      185.105446     400.000000     10 ------u--   997d
S03233 sl      177.057792     400.000000      2 ------u--   495d
S03234 sl      253.573185     400.000000    598 ------u--   906d
S03235 sl      170.963133     400.000000     18 ------u--   497d
S03236 sl      194.456616     400.000000     38 ------u--   915d
S03237 sl      250.533926     400.000000     92 ------u--   803d
S03238 sl        0.000000       0.000000      1 ---------   114d
S03239 sl        0.000000       0.000000      1 ---------   114d
S03240 sl        0.000000       0.000000      1 ---------   114d
S03241 sl      252.162452     400.000000    325 ------u--   369d
S03242 sl      166.635861     400.000000     81 ------u--   631d
S03243 sl      202.468129     400.000000     26 ------u--   400d
S03244 sl        0.000000       0.000000      1 ---------    96d
S03245 sl      383.047890     400.000000     26 ------u--   212d
S03246 sl      288.721920     400.000000      5 ------u--   687d
S03247 sl      261.498716     400.000000    186 ------u--   691d
S03248 sl      255.480486     400.000000     17 ------u--   288d
S03249 sl      319.990661     400.000000    526 ------u--   253d
S03250 sl        0.000000       0.000000      1 ---------   114d
S03251 sl      241.785669     400.000000    533 ------u--   327d
S03252 sl     1223.947545    1223.947545    157 ------u--    44d
S03253 sl      386.038988     400.000000    636 ------u--   136d
S03254 sl      170.798521     400.000000     38 ------u--   228d
```

The following example illustrates using the `list` command to show only volumes meeting some criterion (in this case, those having their `hfree` flag set):

```
adm: 1>list hfree
                                     DATA     EOT             WR/FR
   VSN LB        DATA LEFT        WRITTEN  CHUNK   HFLAGS      AGE
-------------------------------------------------------------------
003249 sl         0.000000     115.000000      9 -f-r------     3h
003250 sl         0.000000     115.000000      9 -f-r------     3h
003251 sl         0.000000     115.000000     10 -f-r------     3h
003252 sl         0.000000     115.000000     11 -f-r------     3h
003255 sl         0.000000     115.000000     15 -f-r------     3h
003258 sl         0.000000     115.000000     13 -f-r------     3h
003263 sl         0.000000     115.000000     12 -f-r------     3h
003264 sl         0.000000       0.000000      1 -f--------     4h
003289 sl         0.000000       0.000000      1 -f-r------     3h
003290 sl         0.000000     215.000000     29 -f-r------     3h
003294 sl         0.000000       0.000000      1 -f--------     4h
```

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 "herr hfree hfull hlock hoa hro"
       herr hfree hfull hlock hoa hro
VSN
--------------------------------------
003232  off   off    on    off off off
003233  off   off   off    off off off
003234  off   off   off    off off off
003235  off   off   off    off off off
003236  off   off    on    off off off
003237  off   off    on    off off off
003238  off   off    on    off off off
003239  off   off    on    off off off
003240  off   off   off    off off off
003241  off   off    on    off off off
003242  off   off    on    off off off
003243  off   off   off    off off off
003244  off   off   off    off off off
003245  off   off    on    off off off
003246  off   off   off    off off off
003247  off   off    on    off off off
003248  off   off    on    off off  on
003249  off    on   off    off off  on
003250  off    on   off    off off  on
003251  off    on   off    off off  on
003252  off    on   off    off off  on
003253  off   off    on    off off  on
```

```
003254  off   off    on    off off  on
```

The following example shows how to display only those tapes assigned to the volume group named `vg9a00`. This example is valid with LSs only.

```
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`.

For MSP:

1. `V` (indicates the volume record type)
2. `vsn`
3. `lbtype`
4. `capacity`
5. `blocksize`
6. `hflags` (in octal)
7. `version`
8. `datawritten`
9. `eotchunk`
10. `eotposition` (`eotzone/eotblockid`) (in hexadecimal)
11. `dataleft`
12. `chunksleft`
13. `wfdate`

14. update
15. id (in octal). This field indicates the type of process that last updated the record.

For LS:

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 6-5 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 MSP/LS databases, use 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 MSP databases. 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 MSP/LS database. In this case, `dmatread` will issue messages indicating that the chunk is not found in the current 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 or to audit the contents of MSP/LS volumes. You may also generate text database records that can be applied to the MSP/LS databases (using the `load` directive in `dmcatadm` and `dmvoladm`, respectively), in order to add the contents of a volume to the MSP/LS database (although this is impractical for large numbers of volumes).

`dmatsnf` can be used to verify one or more tape volumes against the MSP/LS databases. It also can be used to generate journal entries, which can be added to the MSP/LS databases 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 DMF daemon and MSP/LS databases after an MSP, LS, DMF, or system failure. This command captures the database files and compares the contents of the daemon database with each MSP/LS database. Any problems are reported to standard output, but no attempt is made to repair them.

This function can also be done directly using dmatvfy(8) after a snapshot has been taken.

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.

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.

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 MSP 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 "Configuring Daemon Maintenance Tasks" on page 44.

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 MSP and Disk Cache Manager (DCM)" on page 173). 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.

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.

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 MSP 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 "Configuring Daemon Maintenance Tasks" on page 44.

Disk MSP and Disk Cache Manager (DCM)

The disk cache manager (DCM) lets you configure the disk MSP to manage data on secondary disk storage, allowing you to further migrate the data to tape as needed. The DCM provides an automated method of using secondary (slower and less-expensive) disk as a fast-access DMF cache 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), 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 volume group. 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.

Moving Migrated Data between MSPs and Volume Groups

DMF provides a mechanism to move copies of offline or dual-state files from one MSP or volume group to another. The `dmmove`(8) command takes a list of such files and moves them to a specified set of MSPs or volume groups. The list of MSPs or volume groups specified to the dmmove command indicates which MSPs or volume groups 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 volume groups. (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.

Converting from an IRIX DMF to a Linux DMF

You can convert IRIX DMF to Linux DMF and also convert Linux DMF to IRIX DMF . This section describes the necessary steps to convert an IRIX DMF to a Linux DMF.

DMF databases on IRIX machines cannot be copied to Linux machines because of binary incompatibility. Instead, they must be dumped to text on the IRIX machine, and the resulting text file must be loaded into the database on the Linux machine. DMF-managed filesystems, that is, filesystems containing user files that DMF has migrated, can be moved from an IRIX machine to a Linux machine.

It is assumed that sites converting DMF from an IRIX to a Linux machine (or vice versa) will obtain the help of SGI customer support; the following documentation is offered to familiarize you with the necessary steps. This procedure assumes the filesystems will be moved, and that this is done before the last step. It does not describe the steps required to move a filesystem.

Procedure 6-1 IRIX to Linux Conversion

1. Use dmaudit to verify that the DMF databases are valid. For more information, see the dmaudit(8) man page and the *DMF Administrator's Guide for SGI*

InfiniteStorage and *DMF Recovery and Troubleshooting Guide for SGI InfiniteStorage*. To verify the databases that will actually be moved, you should change the filesystem migration levels in the `dmf.conf` file to `none`, run `dmdidle`, and then ascertain that all DMF activity has stopped before beginning this step. You should also use `dmsnap` to back up your databases.

2. Stop DMF on the IRIX system. If DMF is started again on the IRIX system during or after this procedure, the databases captured during step 3 might not reflect reality, and loss of data might result if you use them. To verify the consistency of the DMF databases, use the `dmdbcheck`(8) command.

3. Dump all of the DMF databases to text on the IRIX system. This should include the daemon database and the CAT and VOL databases for all tape MSPs and/or tape LSs. For more information, see the `dmdump`(8) man page.

4. Set up the `/etc/dmf/dmf.conf` file on the Linux system. Note that tape MSPs are not supported on Linux. If you do not have any tape MSPs, the conversion will be simpler if you name all of the FTP and DISK MSPs and the tape volume groups and LSs with the same names used on IRIX. This assumes that you do not already have MSPs or volume groups with these names on your Linux system.

   If you do change the name of an MSP or volume group, you must convert the daemon database. For more information on how to perform this conversion, see the documentation in the `dmconvertdaemon` script.

   If you do have a tape MSP, it must be converted to a volume group in an LS. Perhaps the easiest way to do this is to make the volume group name the same as the name of the MSP being converted. This method avoids making changes to the daemon database. For more information, see step 6. Use `dmcheck` to ensure that your new `/etc/dmf/dmf.conf` file is valid on the Linux system.

   Copy the text versions of the databases that you created in step 3 to the Linux machine.

5. If you have a tape MSP on the IRIX system, run the `dmconvertvol` script to convert the text version of its VOL database to the format required for a volume group in an LS. Also run the `dmconvertcat` script to convert the text version of its CAT database to the format required for a volume group in an LS. If the name of the volume group is different from the name of the tape MSP being converted, run the `dmconvertdaemon` script to convert the text version of the DMF daemon's database. The `dmconvertcat`, `dmconvertvol`, and `dmconvertdaemon` scripts reside in the `/usr/lib/dmf/support` file. Man pages do not exist for them, each script contains documentation.

6. Load the database files from the text files on the Linux machine. Use dmdadm to load the daemon database file. Use dmcatadm to load the CAT database for each of the tape LSs. Use dmvoladm to load the VOL database for each of the tape LSs. If you are converting multiple tape MSPs to multiple volume groups within a single LS, you must load each of their databases. See Example 6-6 on page 176.

7. Use dmdbcheck to check the consistency of databases on the Linux machine.

8. Make sure all DMF filesystems are resident on the Linux machine.

9. Start DMF on the Linux machine and run dmaudit.

Example 6-6 IRIX to Linux Conversion (Single Tape LS)

In the following example, the IRIX machine has a single tape LS, named ls1, and no tape MSPs. 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 1 and 2 of Procedure 6-1 on page 174, the daemon database and the LS databases 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
```

Example 6-7 IRIX to Linux Conversion (Two Tape MSPs)

In the following example, the IRIX machine has two tape MSPs. Their names are msp1 and msp2. 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. This example also assumes that the HOME_DIR configuration parameter is set to /dmf/home on both systems. After completing steps 1 and 2 of Procedure 6-1 on page 174, the daemon database and the tape MSP databases are dumped to text, as follows:

```
$ dmdump -c /dmf/home/daemon > /tmp/dmfdatabases/daemon_txt
$ dmdump /dmf/home/msp1/tpcrdm.dat > /tmp/dmfdatabases/msp1_cat_txt
$ dmdump /dmf/home/msp1/tpvrdm.dat > /tmp/dmfdatabases/msp1_vol_txt
$ dmdump /dmf/home/msp2/tpcrdm.dat > /tmp/dmfdatabases/msp2_cat_txt
$ dmdump /dmf/home/msp2/tpvrdm.dat > /tmp/dmfdatabases/msp2_vol_txt
```

Next, we copy the files in /tmp/dmfdatabases on the IRIX system to /tmp/dmftxtdb on the Linux system. In this example, we assume that msp1 will be converted to a volume group by the name of vgpri in LS ls1. Similarly, msp2 will be converted to a volume group by the name of vgsec in LS ls1. Note that both of these volume groups will be in the same LS. To do this, you must ensure that the VSNs in each of the volume groups are unique.

After creating the /etc/dmf/dmf.conf file on the Linux system, the text versions of the database files are converted. First, the text versions of msp1's files are converted to vgpri:

```
$ dmconvertcat /tmp/dmftxtdb/msp1_cat_txt > /tmp/dmftxtdb/vgpri_cat_txt
$ dmconvertvol /tmp/dmftxtdb/msp1_vol_txt vgpri > /tmp/dmftxtdb/vgpri_vol_txt
```

Next, the text versions of msp2's files are converted to vgsec:

```
$ dmconvertcat /tmp/dmftxtdb/msp2_cat_txt > /tmp/dmftxtdb/vgsec_cat_txt
$ dmconvertvol /tmp/dmftxtdb/msp2_vol_txt vgsec > /tmp/dmftxtdb/vgsec_vol_txt
```

Since the name for the volume group has not been chosen to be the same as for the MSP being converted, the daemon database must be converted. For each tape MSP being converted to a volume group with a different name, dmconvertdaemon is run. In the following two steps, the first step handles the conversion of msp1 to vgpri in the daemon database. The output of that command is then used as a parameter to dmconvertdaemon in the second step. The second step handles the conversion from msp2 to vgsec in the daemon database.

```
$ dmconvertdaemon msp1 vgpri /tmp/dmftxtdb/daemon_txt >\
/tmp/dmftxtdb/daemon_pri_txt

$ dmconvertdaemon msp2 vgsec /tmp/dmftxtdb/daemon_pri_txt > \
/tmp/dmftxtdb/daemon_cnvt_txt
```

Next, the databases are loaded on the Linux system. Since two tape MSPs are being converted to volume groups within the same LS, two files are loaded into the `ls1` `CAT` database, and into the `ls1` `VOL` database:

```
$ dmdadm -u -c "load /tmp/dmftxtdb/daemon_cnvt_txt"
$ dmcatadm -m ls1 -u -c "load /tmp/dmftxtdb/vgpri_cat_txt"
$ dmvoldadm -m ls1 -u -c "load /tmp/dmftxtdb/vgpri_vol_txt"
$ dmcatadm -m ls1 -u -c "load /tmp/dmftxtdb/vgsec_cat_txt"
$ dmvoldadm -m ls1 -u -c "load /tmp/dmftxtdb/vgsec_vol_txt"
```

Now `dmdbcheck` is run to verify the consistency of the databases.

```
$ cd /dmf/home/daemon; dmdbcheck -a dmd_db
$ cd /dmf/home/ls1; dmdbcheck -a libsrv_db
```

Converting from a Tape MSP to a Library Server

For an existing MSP-based configuration to take advantage of the additional features of the LS, the existing databases must be converted. Several databases can be converted at the same time, or the conversion can be done in stages over a period of time. You can perform any of the following conversions:

- Convert just one MSP's databases to a new volume group in a new LS
- Convert an additional MSP to a new volume group within an existing LS
- Convert all databases at once

You can run a mixture of MSPs and LSs, with multiple copies of user files being held simultaneously by a volume group and an MSP. Procedure 6-2 on page 178 provides the steps for conversion from tape MSP to LS databases.

Procedure 6-2 Tape MSP/LS Conversion

1. Run `dmcheck`(8)to check the existing configuration.

2. Copy the production configuration file (`/etc/dmf/dmf.conf`(2.8 or later) or `/etc/dmf/dmbase/host/`*hostname*`/dmf_config`(2.7 or earlier)) and replace the definition of the MSPs to be converted with the stanzas defining the equivalent LS components to a new file, `/tmp/dmf.conf.new`. You might find it useful to examine the sample configuration to be found in `/usr/share/doc/dmf-version_number/info/sample/dmf.conf.ls`. Over time, many small changes have been made to benefit existing installations as well as new ones.

To replace the definition of the MSPs, you must do the following:

a. Delete the stanza for the MSP object.

b. If there are no other references to the device object, remove it.

c. Create an LS stanza and include the following parameters if they were specified in the MSP definition:

   - CACHE_DIR
   - CACHE_SPACE
   - MAX_CACHE_FILE
   - MESSAGE_LEVEL
   - TASK_GROUPS

d. Replace the MSP's name in the MSP_NAMES (or LS_NAMES) directive in the daemon stanza with the name of this LS.

e. Create a drive group stanza and include the following parameters if they were specified in the possibly-deleted device object:

   - BLOCK_SIZE
   - LABEL_TYPE
   - MOUNT_SERVICE
   - MSG_DELAY
   - OV_ACCESS_MODES
   - OV_INTERCHANGE_MODES
   - POSITIONING
   - POSITION_RETRY
   - TMF_TMMNT_OPTIONS
   - VERIFY_POSITION
   - WRITE_CHECKSUM

Add a MOUNT_SERVICE_GROUP parameter to specify the TMF device group or OpenVault drive group. If TMF_TMMNT_OPTIONS contained a -g specification to provide this information, remove that part of it.

The DRIVE_GROUPS parameter in the LS stanza should refer to this drive group.

f. Create one volume group stanza per MSP being converted, possibly with the same names as the MSPs they are replacing, and include the following parameters if they were specified in the MSP definitions:

   - HFREE_TIME
   - MAX_CHUNK_SIZE
   - MAX_PUT_CHILDREN
   - MERGE_CUTOFF
   - TIMEOUT_FLUSH

   Include the ZONE_SIZE parameter from the possibly-deleted device object.

   The VOLUME_GROUPS parameter in the drive group stanza should refer to these volume groups. If their names differ from those of the MSPs they are replacing, update the SELECT_MSP/SELECT_VG policy parameters.

g. In the task-group that controls filesystem backups with run_full_dump.sh and run_partial_dump.sh, change the DUMP_DEVICE parameter to refer to the drive group rather than to the possibly-deleted device object.

To check this new configuration before placing it into production, before running dmcheck(8), set the DMF_CONFIG environment variable to the absolute path of the file, as follows:

```
setenv DMF_CONFIG /tmp/dmf.conf.new
dmcheck
```

3. Run dmaudit(8) and dmdbcheck(8) to confirm that there are no problems with the current databases.

4. Stop DMF and put the new configuration in place. You can run dmcheck(8) again, if you wish.

5. Run /usr/lib/dmf/support/dmmsptols, as described in the man page. This process might take some time (even several hours for a large configuration). The

selection of MSPs that must be converted at this point is determined by the changes made to the configuration in the previous step.

If there are any problems, the `dmmsptols` process will instruct you how to back out of the conversion by using the backups it created.

6. Start DMF and run `dmaudit`(8) and `dmdbcheck`(8).
7. At a later time, you can make additional changes to the configuration to enable or configure new features, such as allocation groups, error recovery, or resource schedulers and watchers. The parameters controlling these are described in Chapter 2, "Configuring DMF" on page 25, or elsewhere in this chapter.

When this procedure is followed, the resulting configuration will schedule tape merges for all volume groups to be done at once. This does not cause problems for the LS as it would for the MSP-based configuration, but you might wish to have finer control over the process. You can do this by creating new task-group objects just to control tape merging, and invoking them with a `TASK_GROUPS` parameter inside the volume groups’ stanzas. The other parameters inside the original task-group should still be invoked only from the LS’s stanza.

Alternatively, the `RUN_TASK` parameters can be placed directly in the volume group stanza; they can be specified outside a task-group’s stanza.

The improved tape positioning code specified by the drive group’s `POSITIONING` parameter to `data` will be activated only for data written by the volume group (that is, newly migrated files and files that have been merged from a tape written by the MSP to a new volume group one). The improvement in performance will become more noticeable over time, as a greater proportion of data fits into these categories.

Library Server Error Analysis and Avoidance

Unlike the MSP, the drive group component of the LS monitors tape use, analyzing any failures, and using 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 for a while, forcing DMF to migrate to a different tape in that volume group, or to recall the file from another tape held by a different volume group. This suspension is usually done by setting the `HLOCK` flag in the tape's entry in the volume database. This makes the tape inaccessible to the volume group 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 HFULL, which results in the tape being used in a read-only mode until eventually emptied by merges or hard deletion of its files. HFREE_TIME seconds after it becomes empty, it may be placed back into use unless the administrator has decided, possibly as a result of testing it, that it should be deleted or replaced.

In all of these situations, the administrator is notified by e-mail.

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.

Library Server Drive Scheduling

When multiple volume groups are requesting the use of more tape drives than exist in the drive group, the resource scheduler is used to decide which volume groups 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 MSPs, XFS dumps, and any direct tape use by the system's users, and does not prevent the LS from working properly, though it might be less than optimal.

By default, the resource scheduler uses a round-robin based algorithm, but a site can assign different weightings to different volume groups to meet local requirements. (For more information, see "Resource Scheduler Objects" on page 80).

Some sites will have requirements that cannot be met by a general purpose algorithm. Such sites can write their own resource scheduler algorithms in C++, to be used in place of the supplied one. Instructions can be found in the `/usr/share/doc/dmf-`*version_number*`/info/sample/RSA.readme` file.

Library Server Status Monitoring

You can observe the performance of the LS in two ways. You can monitor its log file with a tool like `tail -f`, which allows an experienced administrator to follow the flow of events as they happen. You can also 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, which 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 (`lsname` and `rwname` in the following example). The easiest way to find the precise path is to look in the LS log file for messages like the following:

```
dmatls rwname.config_changed:
Resource Watcher output files will be placed in /dmf/spool/lsname/_rwname at DMF startup or
whenever the configuration file is altered or "touch"ed.
```

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 volume group pages, if the volume groups 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 DMF version shown by `dmversion`(1) and the IRIX version from `uname`(1).

DMF Maintenance and Recovery

This chapter contains information for the administrative maintenance of DMF:

- "Retaining Old DMF Daemon Log Files"
- "Retaining Old DMF Daemon Journal Files"
- "Soft- and Hard-deletes" on page 188
- "Using `xfsdump` and `xfsrestore` with Migrated Files" on page 189
- "Using `dmfill`" on page 191
- "Database Recovery" on page 192

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_name*/`msplog.`*yyyymmdd* log file, which also contains sometimes 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 "Configuring Daemon Maintenance Tasks" on page 44, for more information.

Retaining Old DMF Daemon Journal Files

The daemon, the tape MSP, and the LS all 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 "Configuring Daemon Maintenance Tasks" on page 44, 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 which 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 "Configuring Daemon Maintenance Tasks" on page 44, 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.

Using `xfsdump` and `xfsrestore` with Migrated Files

Filesystem backup is a vital operational procedure and DMF-managed filesystems should be backed up regularly. Running DMF affords a high degree of protection for user data. Because DMF only migrates user data and not inodes, directories, or other filesystem structures, you must backup filesystems that hold important data.

The `xfsdump`(1M) and `xfsrestore`(1M) 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.
- Because DMF provides safe, reliable management of offline data, it can be viewed as a data backup service. The `dmmigrate`(8) command lets you implement a 100% migration policy that does not interfere with customary management of space thresholds. The `-a` option of the `xfsdump` command causes `xfsdump` to skip the data associated with any dual-state file. Whenever `xfsdump` detects a file that is backed up by DMF, it retains only the inode for that file, since DMF already has a copy of the data itself.

  When you run `xfsdump -a` in concert with `dmmigrate`, the volume of backup data produced by `xfsdump` can be significantly reduced, thereby reducing the amount of time spent performing backups.

You can also use dmmigrate to force data copies held only in a DCM cache to be copied to tapes in the underlying volume groups. This removes the need to back up the cache filesystem. However, if you do wish to back up the cache instead of of flushing it to tape, you can use any 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.

You can configure tasks in the dump_tasks object to automatically do full and incremental dumps of the DMF-managed filesystems. See "Configuring Daemon Maintenance Tasks" on page 44, for more information.

The dump_tasks object employs scripts that call the xfsdump(1m) command in conjunction with the dmtape DMF support program. This mechanism gives you flexible and efficient use of a predetermined set of backup volumes that are automatically allocated to the xfsdump program as needed during the backup. In order to allow you an equally flexible and efficient method for restoring files backed up by the dump_tasks object, the dmxfsrestore(8) command should be used any time a restore is required for a dump_tasks-managed filesystem. Please see the dmxfsrestore(8) man page for more information on running the command.

Dumping and Restoring Files without the dump_tasks Object

If you choose to dump and restore DMF filesystems without using the provided dump_tasks object, there are several items that you must remember:

- The dump_tasks object uses 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 restored files will be recallable by DMF.

- 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 188.) `dmaudit`(8) will report this as an inconsistency between the filesystem and the 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 any user data loss. 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 in that case.

Sites should be aware that inconsistencies between a filesystem and the DMF database can occur as a result of restoring migrated files, and that it is good practice to run `dmaudit` after a restore to correct those inconsistencies.

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 the 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

You configure tasks in the `run_copy_databases.sh` task in the `dump_tasks` object to automatically generate DMF database backups. See "Configuring Daemon Maintenance Tasks" on page 44, for more information.

There are several databases in the DMF package. The daemon database consists of the following files:

- *HOME_DIR*/*daemon_name*/`dbrec.dat`
- *HOME_DIR*/*daemon_name*/`dbrec.keys`
- *HOME_DIR*/*daemon_name*/`pathseg.dat`
- *HOME_DIR*/*daemon_name*/`pathseg.keys`

The database definition file (in the same directory) that describes these files and their record structure is named `dmd_db.dbd`.

Each tape MSP/LS has two databases in the *HOME_DIR*/*msp_or_ls_name* directory:

- The CAT database (files `tpcrdm.dat`, `tpcrdm.key1.keys`, and `tpcrdm.key2.keys`)
- The VOL database (files `tpvrdm.dat` and `tpvrdm.vsn.keys`)

The database definition file (in the same directory) that describes these files and their record structure is named `atmsp_db.dbd` (for MSPs) or `libsrv.db.dbd` (for LSs).

Database Recovery Procedures

The DMF daemon and the tape MSP/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 multiple MSPs or 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 7-1 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.
2. If you have configured the `run_copy_databases` task, copy the files from the directory with the most recent copy of the databases that were in *HOME_DIR*.
3. If you have **not** configured the `run_copy_databases` task, reload an old version of the daemon or tape MSP/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/msp_or_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 tape MSP, the journal files are named `atmsp_db.`*yyyymmdd[.hhmmss]*.

   For the LS, the journal files are named `libsrv_db.`*yyyymmdd[.hhmmss]*.
5. Note the time of the last database update from step 2.
6. Use `dmdbrecover` to update the old database with the journal entries from journal files identified in step 3.

Example 7-1 Database Recovery Example

Suppose that the filesystem containing *HOME_DIR* was destroyed on February 1, 1997, and that your most recent backup copy of the daemon and tape MSP databases is from January 28, 1997. To recover the database, you would do the following:

1. Stop DMF.
2. Ensure that *JOURNAL_DIR/daemon_name* (or *JOURNAL_DIR/msp_or_ls_name*) contains the following journal files (one or more for each day):

   JOURNAL_DIR/daemon_name

   ```
   dmd_db.19970128.235959
   dmd_db.19970129.235959
   dmd_db.19970130.235959
   dmd_db.19970131.235959
   dmd_db.19970201
   ```

   JOURNAL_DIR/msp_name (If a tape MSP is configured)

   ```
   atmsp_db.19970128.235959
   atmsp_db.19970129.235959
   atmsp_db.19970130.235959
   atmsp_db.19970131.235959
   atmsp_db.1997020
   ```

   JOURNAL_DIR/ls_name (If an LS is configured)

   ```
   libsrv_db.19970128.235959
   libsrv_db.19970129.235959
   libsrv_db_db.19970130.235959
   libsrv_db_db.19970131.235959
   libsrv_db_db.1997020
   ```

3. Restore databases from January 28, to *HOME_DIR/daemon_name* and/or *HOME_DIR/msp_or_ls_name*. The following files should be present:

   HOME_DIR/daemon_name

   ```
   dbrec.dat
   dbrec.keys
   pathseg.dat
   pathseg.keys
   ```

HOME_DIR/msp_or_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 msp_name atmsp_db (If a tape MSP is configured)
dmdbrecover -n ls_name libsrv_db (If an LS is configured)
```

Messages

This appendix describes the format and interpretation of messages reported by `dmcatadm`(8) and `dmvoladm`(8). If you are uncertain about how to correct these errors, contact your customer service representative.

Message Format

Messages in this section are divided into the format used for `dmcatadm` and `dmvoladm`.

Message Format for Catalog (CAT) Database and Daemon Database Comparisons

Error messages generated when comparing the CAT 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
```

Message Format for Volume (VOL) Database and Catalog (CAT) Database and Daemon Database Comparisons

Error messages generated when comparing the VOL database to the CAT 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
hrsv 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 `dmcatadm` database.

nnn `bytes duplicated`	Two or more chunks in the database contain data from the same region of the file (MSPs only).
nnn `bytes duplicated in volume group name`	Two or more chunks in the database, which belong to volume group name, contain data from the same region of the file (LSs only).
`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`	The daemon entry was not found in the CAT database.
`missing from daemon db`	No daemon entry was found for the entry in the CAT database (MSP only).
`entry for volume group name missing from daemon db`	No daemon entry was found for the entry in the CAT database (LS only).
`no chunk for bytes nnn - nnn`	There is no chunk that contains the specified bytes of the file (MSP only).
`for volgrp name; no chunk for bytes nnn - nnn`	There is no chunk that contains the specified bytes of the file (LS only).

`dmvoladm` Message Interpretation

The following lists the meaning of messages associated with the `dmvoladm` database.

`blocksize is bad`	The `blocksize` field for the tape is <=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 database 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 database 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 database but is not in the VOL database.

`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`
`volume is empty but hrsv is on`
	When a volume is empty, the hfull, hsparse, and hrsv 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 volume contains new chunks`	One or more of the chunks associated with this volume were written by the advanced tape MSP, but the version value does not match.
`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 MSP/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 DMF distributed command feature is available with DMF version 2.7 and later. This appendix presents an overview of the feature, a summary of data types, and a summary of user-accessible API subroutines.

Overview

The distributed command feature allows DMF commands to execute on a host other than the host on which the DMF daemon is running. A host that imports DMF-managed filesystems from the DMF daemon host machine can execute the dmput, dmget, dmls, dmfind, dmattr, and dmcopy commands locally.

As part of the distributed command feature, the DMF user commands listed above were radically re-designed to communicate with a process named dmusrcmd instead of directly with the DMF daemon. The DMF user commands are no longer installed as setuid root processes. Rather, the dmusrcmd process is executed as setuid root and performs all of the validity checks and communicates, ultimately, with the DMF daemon.

For the DMF user commands to communicate in an efficient and consistent manner with the dmusrcmd process, the DMF user library, libdmfusr.so, must be accessed. This is a shared object library (DSO) that is installed in /usr/{lib|lib32|lib64} and to which each of the DMF user commands is linked for its protocol-based communications.

As a feature of this re-design, the subroutines that comprise the DMF user command application program interface (API) are now available to user-written programs simply by linking to libdmfusr.so. Sites can now design and write their own custom DMF user commands, which eliminates the need to use wrapper scripts around the DMF user commands.

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 (put, get, fullstat, or copy) API call. The context is used by each API routine 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 and dmu_err.h header files are provided, which are required for a site to effectively create their own commands.

Both header files are installed in /usr/include/dmf. The libdmfusr.h file contains 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 definitions 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 DmuErrorInfo_t object described below (see "DmuErrorInfo_t" on page 208). 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. The 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 are obviously going to perform less efficiently than their asynchronous counterparts because of the serial nature of their activity.

The asynchronous subroutine calls return immediately to the caller. The return code of these asynchronous routines 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 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 206) to determine the eventual result of the original request.

There are several different API subroutine calls for processing asynchronous request completion objects. The user can choose to be simply notified when all requests have completed, without doing any processing of the return status of each request. The user can also choose to process the return status of each request, one at a time, in the order in which they complete, or in the order in which they were sent (request ID order), or the user can, by request ID, synchronously wait on an individual asynchronous request's completion.

The API includes well-defined protocols that it uses to communicate with the dmusrcmd process. These protocols make use of the pthreads(5) mechanism and as such, any user application program making use of the API via libdmfusr.so will also need to link to the shared object library, libpthread.so, via the -lpthread compiler option (cc(1)or CC(1)) or loader option (ld(1) or rld(1)) option.

The API can return different types of objects to the callers of many of the API subroutines by passing the addresses of the objects in subroutine parameters. Many of these objects have been created by allocating new memory for them through the use of the `malloc` command. The API includes several subroutines that will free the memory used by these objects when the caller is through with them, and they are defined below in "Memory Management Subroutines" on page 224. It is up to the caller to make use of these subroutines, however, if memory leakage is a concern.

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 to avoid causing a `SIGSEGV` fault in the application program.

Data Types

The data types described in this section are defined in `libdmfusr.h`. For the most up-to-date definitions of each of these types, see the `libdmfusr.h` file. The following information is provided as a general description and overall usage outline.

DmuAllErrors_t

This 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.

This object may contain 0 or more `DmuErrorInfo_t` objects (see "`DmuErrorInfo_t`" on page 208).

DmuByteRange_t

This object defines a range of bytes that are to be associated with a put or get request. The fields and their definitions are as follows:

`offset`	Starting offset in bytes of the range in the file.
`size`	Size in bytes of the range.

Currently, only offset 0 and size 0 (indicating the whole file) are supported as valid definitions.

DmuByteRanges_t

This object defines a set of DmuByteRange_t objects that are to be associated with a put or get request. The fields and their definitions are as follows:

rounding	Rounding method to be used to validate range addresses. Only DMU_RND_NONE is currently defined.
num_ranges	Number of DmuByteRange_t objects in the ranges field. Currently, only a single range is allowed.
ranges	A pointer to an array of DmuByteRange_t objects. Currently, only a single element array is allowed.

Example: In the current API, define a DmuByteRanges_t as follows:

```
DmuByteRanges_t ranges = {DMU_RND_NONE, 0, NULL};
```

or

```
DmuByteRange_t range = {0, 0};
DmuByteRanges_t ranges = {DMU_RND_NONE, 1, &range);
```

DmuCompletion_t

This object is returned by one of the API request completion routines (see "Request Completion Subroutines" on page 219) with the results of an asynchronous request.

The request_id field can be used to associate the completion object with an asynchronous request that was previously issued. This value coincides with the request ID value that any of the asynchronous routines return to the user.

The ureq_data field is request-type specific, and API routines are defined below (see "Fullstat Requests" on page 213) to help the application process the object (that is, to extract the DmuFullstat_t information from a fullstat completion). This field has no meaning for put, get, or copy requests.

The reply_code field has 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.

The allerrors field (type DmuAllErrors_t, defined previously) contains the error information for a failed request.

DmuCopyRange_t

This object defines a range of bytes that are to be associated with a copy request. The fields and their definitions are as follows:

src_offset	Starting offset in bytes of the range in the source file to be copied.
src_length	Length in bytes of the range to be copied.
dst_offset	Starting offset in bytes in the destination file to which the copy is sent.

DmuCopyRanges_t

This object defines a set of DmuCopyRange_t objects that are to be associated with a put or get request. The fields and their definitions are as follows:

rounding	Rounding method to be used to validate range addresses. Only DMU_RND_NONE is currently defined.
num_ranges	Number of DmuCopyRange_t objects in the ranges field. Currently, only a single range is allowed.
ranges	A pointer to an array of DmuCopyRange_t objects. Currently, only a single element array is supported.

DmuErrHandler_f

This type 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 DmuErrorInfo_t object (see "DmuErrorInfo_t" on page 208) 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 DmuErrorInfo_t object can also be passed into the DmuErrHandler_f, which 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 routine, or the caller can use one of the following API-supplied routines:

DmuDefErrHandler	Outputs the severity of error and the error message associated with the error to stderr.
DmuNullErrHandler	Does nothing with the error.

DmuError_t

This is the type that most of the API subroutines pass as a return code. The definition DmuNoError is the general success return code.

DmuErrorInfo_t

This object contains the information about a single error occurrence. Included are the error code, which might or might not be meaningful to an application, the originator of the error (API, dmusrcmd, daemon), a severity code, and perhaps most importantly, an ACSII message that can be displayed.

DmuFhandle_t

This object contains the ASCII representation of the file fhandle as it is known on the host on which the file's filesystem is native.

DmuFullstat_t

This rather lengthy 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.

DmuReplyOrder_t

This type is used to select the order in which asynchronous replies are to be returned by the API reply processing subroutines defined in the following list.

DmuAnyOrder	Return in the order the replies are received.

`DmuReqOrder`	Return in the order the requests were issued.

DmuReplyType_t

This type 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.

The valid definitions are:

`DmuIntermed`	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 library server for processing and that the completion reply is deferred until that operation is complete.
`DmuFinal`	Final reply for the request.

This definition is used to specify the types of replies that some of the reply processing routines defined below are to consider.

DmuReqid_t

This type is used to describe the request identifier returned to the caller for a successful asynchronous function call.

DmuRounding_t

This is an enum that specifies the kind of address manipulation that the caller would like performed on his DMF put/get/copy file access requests:

`DMU_RND_NONE`	Do none.
`DMU_RND_IN`	Not yet supported.
`DMU_RND_OUT`	Not yet supported.

DMU_RND_MAX	Not yet supported.

User-Accessible API Subroutines

This section describes the following types of user-accessible API subroutines:

- Context manipulation
- DMF daemon request
- Request completion
- Memory management

Context Manipulation Routines

This section describes context manipulation routines.

DmuCreateContext Subroutine

The DmuCreateContext subroutine creates an opaque context for the API to use to correctly communicate with the dmusrcmd process. This routine 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. The code is as follows:

```
extern DmuError_t
DmuCreateContext(
                void                **dmuctxt,
        const   DmuErrHandler_f     err_handler,
                pid_t               *child_pid,
                DmuAllErrors_t      **errs)
```

The parameters of the DmuCreateContext() call are as follows:

dmuctxt — This parameter is returned with the address of the newly created API context. This parameter is passed to the API on all subsequent subroutine calls that require the program's API context.

err_handler — This parameter can be used to specify a user-defined error handling routine. The DmuErrHandler_f type 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 207.

`child_pid`	This parameter specifies the 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`	This parameter is set with a pointer to a `DmuAllErrors_t` object if errors occur.

If the `DmuCreateContext` call completes successfully, it returns `DmuNoError`.

`DmuDestroyContext` Subroutine

The `DmuDestroyContext` subroutine destroys the API context to which that `dmuctxt` points. The memory that had been allocated for its use is freed. The code is as follows:

```
extern DmuError_t
DmuDestroyContext(
                 void           *dmuctxt,
                 DmuAllErrors_t  **errs)
```

The parameters of the `DmuDestroyContext()` call are as follows:

`dmuctxt`	This parameter is pointer to an API context that was previously created via `DmuCreateContext()`.
`errs`	This parameter is set with a pointer to a `DmuAllErrors_t` object if errors occur.

DMF File Request Subroutines

Each of the following subroutines makes a DMF file request. The context parameter that is included in each of these calls 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 code is as follows:

```
extern DmuError_t
DmuCopyAsync(
              void             *dmuctxt,
        const char             *srcfile_path,
        const char             *dstfile_path,
        const int              copy_flags,
        const DmuCopyRanges_t *copyranges,
              DmuReqid_t       *request_id,
              DmuAllErrors_t   **errs)

extern DmuError_t
DmuCopySync(
              void             *dmuctxt,
        const char             *srcfile_path,
        const char             *dstfile_path,
        const int              copy_flags,
        const DmuCopyRanges_t *copyranges,
              DmuAllErrors_t   **errs)
```

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.

The DmuCopySync subroutine does not return until the requested copy has either completed successfully or been aborted due to an error condition.

This request manipulates the destination file in exactly the same manner as that of the to_file argument of the dmcopy command.

The parameters of these routines are as follows:

dmuctxt — This parameter is a pointer to an API context that was previously created by DmuCreateContext().

srcfile_path — This parameter specifies the pathname of the source (input) file for the copy operation. It must be an offline or dual state DMF file.

dstfile_path — This parameter 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 in a DMF-managed filesystem that is native on the same host as that of the source file's filesystem.
`copy_flags`	This parameter specifies the OR'd value of the following copy operation flags as defined in `libdmfusr.h`: `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`	This parameter specifies a pointer to a `DmuCopyRanges_t` object, as defined in "`DmuCopyRanges_t`" on page 207 and in `libdmfusr.h`. Currently, this object can have only one `DmuCopyRange_t` as defined in "`DmuCopyRange_t`" on page 207 and in `libdmfusr.h`.
`request_id`	This parameter specifies a unique request ID. This value can be used when processing `DmuCompletion_t` objects to find the completion status.
`errs`	This parameter is set with a pointer to a `DmuAllErrors_t` object if errors occur.

If the routine succeeds, it returns `DmuNoError`.

Fullstat Requests

The following routines send a `fullstat` request to the `dmusrcmd` process. The ultimate result of this request is the transfer of a `DmuFullstat_t` object to the caller. Code for the routines is as follows:

```
extern DmuError_t
DmuFullstatByPathAsync(
                         void             *dmuctxt,
                const    char             *path,
                         DmuReqid_t       *request_id,
                         DmuAllErrors_t   **errs)
```

```
extern DmuError_t
DmuFullstatByPathSync(
                        void           *dmuctxt,
                const   char           *path,
                        DmuFullstat_t  **fullstatb,
                        DmuFhandle_t   **fhandle,
                        DmuAllErrors_t **errs)

extern DmuError_t
DmuFullstatByFhandleAsync(
                        void           *dmuctxt,
                const   DmuFhandle_t   *client_fhandle,
                        DmuReqid_t     *request_id,
                        DmuAllErrors_t **errs)

extern DmuError_t
DmuFullstatByFhandleSync(
                        void           *dmuctxt,
                const   DmuFhandle_t   *client_fhandle,
                        DmuFullstat_t  **fullstatb,
                        DmuAllErrors_t **errs)
```

The ’Sync’ versions of these calls do not return until the DmuFullstat_t has been received or the request has been aborted due to errors.

The ’Async’ versions of these routines 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 . That is the only way to actually receive the DmuFullstat_t object, however. The DmuFullstatCompletion subroutine has been supplied to extract the fullstat information from a fullstat completion object.

The ’ByPath’versions of these calls allow the target file to be defined by its pathname.

The ’ByFhandle’ versions of these calls allow the target file to be defined by its filesystem handle, the fhandle. These routines 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 routines can return a successful completion (DmuNoError), but might still not return valid DmuFullstat_t information. The routines are designed to return the normal stat type information regardless of whether a DMAPI fullstat could be successfully completed. Upon return from these routines, the caller can use a macro defined in the `libdmfusr.h` file named `DMU_NO_FULLSTAT_INFO` with the address of the `DmuFullstat_t` block as the parameter and it will verify the validity of the DMAPI information in the `DmuFullstat_t` block.

The parameters of these routines are as follows:

`dmuctxt` — This parameter is a pointer to an API context that was previously created by `DmuCreateContext()`.

`path` — This parameter specifies the relative or absolute pathname of the target file.

`client_fhandle` — This parameter specifies the DMF filesystem `fhandle` of the target file.

`fullstatb` — This parameter specifies the pointer that will be returned with the `DmuFullstat_t fullstat` block.

`fhandle` — This parameter specifies a pointer that will be returned with the `DmuFhandle_t` value.

`request_id` — This parameter is set with the unique request ID of the `fullstat` request. You can use this value when processing `DmuCompletion_t` objects to find the request's completion status.

`errs` — This parameter is set with a pointer to a `DmuAllErrors_t` object if errors occur.

If the routine succeeds, it returns `DmuNoError`.

Put File Requests

The following routines perform the put DMF request.

```
extern DmuError_t
DmuPutByFhandleAsync(
                       void            *dmuctxt,
                const  DmuFhandle_t    *client_fhandle,
                const  int             flags,
                const  DmuByteRanges_t *byteranges,
                       DmuReqid_t      *request_id,
                       DmuAllErrors_t  **errs)
```

```
extern DmuError_t
DmuPutByFhandleSync(
                        void            *dmuctxt,
                const   DmuFhandle_t    *client_fhandle,
                const   DmuMigFlags_t   flags,
                const   DmuByteRanges_t *byteranges,
                        DmuAllErrors_t  **errs)

extern DmuError_t
DmuPutByPathAsync(
                        void            *dmuctxt,
                const   char            *path,
                const   DmuMigFlags_t   flags,
                const   DmuByteRanges_t *byteranges,
                        DmuReqid_t      *request_id,
                        DmuAllErrors_t  **errs)

extern DmuError_t
DmuPutByPathSync(
                        void            *dmuctxt,
                const   char            *path,
                const   DmuMigFlags_t   flags,
                const   DmuByteRanges_t *byteranges,
                        DmuAllErrors_t  **errs)
```

The ’Sync’ versions of these calls do not return until the put request has either completed successfully, or been aborted due to errors.

The ’Async’ versions of these routines 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.

The ’ByPath’ versions of these calls allow the target file to be defined by its pathname.

The ’ByFhandle’ versions of these calls allow the target file to be defined by its filesystem handle, the `fhandle`. These routines 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 parameters of these routines are as follows:

`dmuctxt`	This parameter is a pointer to an API context that was previously created by `DmuCreateContext()`.
`client_fhandle`	This parameter specifies the DMF filesystem `fhandle` of the target file. Valid for use only by a privileged (root) user.
`path`	This parameter specifies the relative or full path name of the target file.
`flags`	These parameters specify the following migration flags as defined in `libdmfusr.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`	This parameter specifies a pointer to a `DmuByteRanges_t` object, as defined in `libdmfusr.h`. Currently, this object can have only one `DmuByteRange_t` as defined in `libdmfusr.h`.
`request_id`	This parameter specifies a unique request ID of the `put` request. This value can be used when processing `DmuCompletion_t` objects to find the request's completion status.
`errs`	This parameter is set with a pointer to a `DmuAllErrors_t` object if errors occur.

If the routine succeeds, it returns `DmuNoError`.

Get File Requests

The following routines perform the get DMF request.

```
extern DmuError_t
DmuGetByFhandleAsync(
                          void              *dmuctxt,
```

```
                const   DmuFhandle_t     *client_fhandle,
                const   DmuRecallFlags_t flags,
                const   DmuByteRanges_t  *byteranges,
                        DmuReqid_t       *request_id,
                        DmuAllErrors_t   **errs)

extern DmuError_t
DmuGetByFhandleSync(
                        void             *dmuctxt,
                const   DmuFhandle_t     *client_fhandle,
                const   DmuRecallFlags_t flags,
                const   DmuByteRanges_t  *byteranges,
                        DmuAllErrors_t   **errs)

extern DmuError_t
DmuGetByPathAsync(
                        void             *dmuctxt,
                const   char             *path,
                const   DmuRecallFlags_t flags,
                const   DmuByteRanges_t  *byteranges,
                        DmuReqid_t       *request_id,
                        DmuAllErrors_t   **errs)

extern DmuError_t
DmuGetByPathSync(
                        void             *dmuctxt,
                const   char             *path,
                const   DmuRecallFlags_t flags,
                const   DmuByteRanges_t  *byteranges,
                        DmuAllErrors_t   **errs)
```

The ’Sync’ versions of these calls do not return until the get request has either completed successfully, or has been aborted due to errors.

The ’Async’ versions of these routines 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.

The ’ByPath’versions of these calls allow the target file to be defined by its path name.

The ’ByFhandle’ versions of these calls allow the target file to be defined by its filesystem handle, the fhandle. These routines 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 parameters of these routines are as follows:

`dmuctxt` — This parameter is a pointer to an API context that was previously created by `DmuCreateContext()`.

`client_fhandle` — This parameter specifies the DMF filesystem `fhandle` of the target file. Valid for use only by a privileged (root) user.

`path` — This parameter specifies the relative or full path name of the target file.

`flags` — These parameters specify the following recall flags as defined in `libdmfusr.h`:

- `RECALL_NONE` – No flags specified.
- `RECALL_NOWAIT` – If the daemon is not available to process the request, do not wait. Return immediately.

`byteranges` — This parameter specifies pointer to a `DmuByteRanges_t` object, as defined in `libdmfusr.h`. Currently, this object can have only one `DmuByteRange_t`, as defined in `libdmfusr.h`.

`request_id` — This parameter specifies a unique request ID of the `get` request. This value can be used when processing `DmuCompletion_t` objects to find the completion status.

`errs` — This parameter is set with a pointer to a `DmuAllErrors_t` object if errors occur.

If the routine 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 simply 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 type *type* 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. Code for the routine is as follows:

```
extern DmuError_t
DmuAwaitReplies(
                void            *dmuctxt,
                DmuReplyType_t  type,
                int             max_outstanding,
                DmuAllErrors_t  **errs)
```

The parameters of this routine are as follows:

dmuctxt
: This parameter is a pointer to an API context that was previously created by DmuCreateContext().

type
: This parameter 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 209 or in libdmfusr.h.

max_outstanding
: This parameter 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 routine returns.

errs
: This parameter is set with a pointer to a DmuAllErrors_t object if errors occur. Note that this

error object refers to errors that occur while waiting and receiving the next reply.

If no errors occurred getting the next reply, this routine returns `DmuNoError`.

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 routine returns. An intermediate reply has no completion object associated with it, and 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.

Code is as follows:

```
extern DmuError_t
DmuGetNextReply(
                void             *dmuctxt,
                DmuReplyOrder_t  order,
                DmuReplyType_t   type,
                DmuCompletion_t  **comp,
                DmuAllErrors_t   **errs)
```

The parameters of this routine are as follows:

`dmuctxt` — This parameter is a pointer to an API context that was previously created by `DmuCreateContext()`.

`order` — This parameter 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 208 or in `libdmfusr.h`.

`type`
: This parameter 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`
: This parameter is set upon receipt of a final (completion) reply to the address of a completion object. 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 `DmuNoError`, the `comp->allerrors` object will contain the error information needed to determine the cause of the error. Note that the `errs` parameter on the subroutine call does not contain the error information for the failed request.

`errs`
: This parameter is set with a pointer to a `DmuAllErrors_t` object if errors occur. Note that this error object refers to errors that occur while waiting and receiving the next reply. It does not refer to the errors that occurred during the request processing that is referenced by `comp`.

If no errors occurred getting the next reply, this routine 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 request reply specified on the call is received.

Code for this routine is as follows:

```
extern DmuError_t
DmuGetThisReply(
                void            *dmuctxt,
                DmuReqid_t      request_id,
```

```
DmuCompletion_t **comp,
DmuAllErrors_t  **errs)
```

The parameters of this routine are:

`dmuctxt` — This parameter is a pointer to an API context that was previously created by `DmuCreateContext()`.

`request_id` — This parameter is the unique request ID of the request for which the caller wants to wait.

`comp` — This parameter is set upon receipt of the final (completion) reply to the address of a completion object. 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 that the `errs` parameter on the subroutine call does not contain the error information for the failed request.

`errs` — This parameter is set with a pointer to a `DmuAllErrors_t` object if errors occur. Note that this error object refers to errors that occur while waiting and receiving this reply. It does not refer to the errors that occurred during the request processing that is referenced by `comp`.

If no errors occurred getting the next reply, this routine returns `DmuNoError`. Any other error return code indicates an error and the `errs` parameter can be processed for the error information.

`DmuFullstatCompletion` Subroutine

The `DmuFullstatCompletion` subroutine is supplied in the API to allow a user to make asynchronous `fullstat` requests and to ease the processing of the completion objects of those requests. When a `DmuCompletion_t` is returned to the caller via `DmuGetNextReply()` or `DmuGetThisReply()`, the user can extract the `DmuFullstat_t` and `DmuFhandle_t` information by calling this subroutine.

Code for the routine is as follows:

```
extern DmuError_t
DmuFullstatCompletion(
                DmuCompletion_t  *comp;
                DmuFullstat_t    **fullstatb,
                DmuFhandle_t     **fhandle)
```

The parameters on this call are as follows:

comp	This parameter specifies the DmuCompletion_t object from an asynchronous fullstat request.
fullstatb	This parameter is returned with the fullstat information returned by the original request.
fhandle	This parameter is returned with the fhandle returned by the original request.

Memory Management Subroutines

Memory management subroutines are available so that API users can efficiently manage their use of memory. Each subroutine defined in this section frees all of the memory associated with the object being deleted. It is safe to call all of these subroutines with a null object pointer.

The user should feel free to call any of these subroutines, using a parameter of the appropriate type that was used as input to one of the function or completion processing routines described previously.

- The following subroutine frees all memory associated with a DmuAllErrors_t object:

  ```
  extern void
  DmuDeleteAllErrors( DmuAllErrors_t *errs )
  ```

- The following subroutine frees all memory associated with a DmuCompletion_t object:

  ```
  extern void
  DmuDeleteCompletion( DmuCompletion_t *comp )
  ```

- The following subroutine frees all memory associated with a `DmuFullstat_t` object:

```
extern void
DmuDeleteFullstat( DmuFullstat_t *fullstat )
```

- The following subroutine frees all memory associated with a `DmuFhandle_t` object:

```
extern void
DmuDeleteFhandle( DmuFhandle_t *fhandle )
```

DMF Directory Structure Prior to 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. Since 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.

Differences from UNICOS DMF and UNICOS/mk DMF

If you are upgrading from a UNICOS or UNICOS/mk operating system to an IRIX or Linux operating system, you will need to be aware of the differences between IRIX/Linux DMF functionality and UNICOS or UNICOS/mk DMF functionality. The basic structure of DMF is the same for IRIX or Linux environments as for UNICOS and UNICOS/mk environments. However, the differences occur in areas affected by operating system dependencies. The DMF administrator interface differs in the areas of product installation, database administration utilities, and automatic space management. There are also differences in basic terminology. Table D-1 on page 229 provides a summary of key differences between the two operating systems as they relate to DMF.

Table D-1 Differences From UNICOS and UNICOS/mk

Functionality	UNICOS and UNICOS/mk	IRIX and Linux
Kernel interface that supports file state transitions	`dmofrq`(2) command.	DMAPI 2.3
Use of `HOME_DIR`, `SPOOL_DIR`, `JOURNAL_DIR` directories	No separate daemon subdirectory (daemon files in root of `HOME`, `SPOOL`, or `JOURNAL` directory).	Separate daemon subdirectory.
Protected files feature.	Supported as a part of the user database feature (UDB).	Not supported.
`dmmode`(2) command	Supported.	Not supported. Offline files are always processed when accessed.
Client/server configuration option	Supported.	Not supported.
Reporting	`dmhit`(blank) command.	`dmscanfs`(8) command.
DMF database administration	`dmdalter` and `dndbase` commands.	`dmdadm`(8) command, which has an administrator interface similar to that of the `dmcatadm`(8) and `dmvoladm`(8) commands.

Functionality	UNICOS and UNICOS/mk	IRIX and Linux
File migration and conversion to dual state	`dmmigall`(8) command.	`dmmigrate`(8) command.
Information reporting on DMF managed files	`ls`(1) and `find`(1) commands	`dmls`(1) and `dmfind`(1) commands (based on the IRIX commands, `ls` and `find`).
Structure of directory written by the `dmsnap`(8) command	Daemon database in `snap` directory, MSP databases in `snap` directory subdirectories named for *mspname*	Separate daemon and MSP/LS subdirectories in `snap` directory
File handle terminology	File handle.	Bit-file identifier (`bfid`).
File handle terminology.	`dev/inode`.	`fhandle`.

Glossary

active database entry

A valid daemon database entry. See also *soft-deleted database entry* and *hard-deleted database entry*.

allocation group

A source of additional volumes for a volume group that runs out of media. An allocation group defines a logical pool of volumes, and is different from an actual operational volume group. Normally, one allocation group is configured to serve multiple volume groups. If a volume group has an associated allocation group, when the volume group runs out of empty volumes, the library server assigns one from the allocation group to it, subject to configuration restrictions. Similarly, when a volume's `hfree` flag is cleared in a volume group, 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`).

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.

base object

The configuration object that defines pathname and file size parameters necessary for DMF operation.

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.

BFID set

The collection of database entries and the user file associated with a particular BFID.

BFID-set state

The sum of the states of the components that comprise a BFID set: the file state of any user file and the state of any database entries (incomplete, complete, soft-deleted, or active).

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.

candidate list

A list that contains an entry for each file in a filesystem eligible for migration, 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.

CAT records

The catalog (CAT) records in the tape MSP or LS database that track which migrated files reside on which tape volumes.

chunk

That portion of a user file that fits on the current media (tape) 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.

complete MSP or volume group daemon-database entry

An entry in the daemon database whose `path` field contains a key returned by its MSP or volume group, indicating that the MSP or volume group maintains a valid copy of the user file.

compression

The mechanism provided by the tape MSP/LS for copying active data from volumes that contain largely obsolete data to volumes that contain mostly active data. This process is also known as volume merging or tape merging.

configuration object

A series of parameter definitions in the DMF configuration file that controls the way 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.

daemon database

A database maintained by the DMF daemon. This database contains such information as the BFID, the MSP volume group name, and MSP or volume group key for each copy of a migrated file.

daemon object

The configuration object that defines parameters necessary for `dmfdaemon`(8) operation

data-pointer area

The portion of the inode that points to the file's data blocks.

device object

The configuration objects that define parameters for DMF's use of tape devices.

direct-access storage device (DASD)

An IBM disk drive.

disk cache manager (DCM)

The feature that lets you configure the disk media-specific process (MSP) to manage data on secondary storage, allowing you to further migrate the data to tape as needed.

DMF state

See *file state*.

dual-state file

A file whose data resides both online and offline.

dual-state filesystems

Those filesystems that have the necessary inode space to support dual-state files.

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, unmigrating, never-migrated, or have an invalid DMF state.

freed file

A user file that has been migrated and whose data blocks have been released.

fully migrated file

A file that has one or more complete offline copies and no pending or incomplete offline copies.

hard-deleted database entry

An MSP or volume group database entry that has been removed from the daemon database and whose MSP or volume group copy has been discarded. See also *active database entry* and *soft-deleted database entry*.

inode

The portion of a file that contains the BFID, the state field, and the data pointers.

incomplete MSP or volume group daemon-database entry

An entry in the daemon database for an MSP or volume group 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.

library server (LS)

The daemon-like process that provides much of the same functionality as one or more tape MSPs. Each LS has an associated catalog (CAT) and volume (VOL) database. An LS can be configured to contain one or more drive groups. Each drive group defines a pool of volume groups. A volume group is responsible for copying data blocks onto alternate media.

LS

See *library server*

media-specific process (MSP)

The daemon-like process by which data blocks are copied onto alternate media, and which assigns keys to identify the location of the migrated data.

merging

The mechanism provided by the tape MSP/LS for copying active data from volumes that contain largely obsolete data to volumes that contain mostly active data. This process is also known as *volume merging* or *tape merging*.

migrated file

A file that has a BFID and whose offline copies (or copy) are completed. Migrated files can be *dual-state* or *offline*.

migrating file

A file that has a BFID but whose offline copies (or copy) are in progress.

MSP

See *media-specific process (MSP)*.

MSP or volume group database entry

The daemon database entry for a file that contains the path or key that is used to inform a particular MSP or volume group where to locate the copy of the file's data.

MSP objects

The configuration objects that define parameters necessary for that MSP's operation

nonmigrated file

A file that does not have a BFID or any offline copies. See *regular file*.

offline file

A file whose inode contains a BFID but whose disk blocks have been removed. The file's data exists elsewhere in copies on alternate media.

offline pointer

In tape MSP/LS processing, a character string that the MSP/LS returns to the daemon to indicate how a file is to be retrieved. For the tape MSP/LS , the offline pointer is the character key into the MSP/LS catalog (CAT) records of the database.

orphan chunks

Unused chunks in the tape MSP/LS catalog (CAT) database entries resulting from the removal of migrated files.

orphan database entries

Unused database entries resulting from the removal of migrated files during a period in which the DMF daemon is not running.

parameter

See *configuration parameter*.

policy objects

The configuration objects that specify parameters to determine MSP or volume group selection, automated space management policies, and/or file weight calculations in automatic space management.

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.

regular file

DMF considers a regular file to be one with no BFID and no offline copies.

snapshot

The information about all BFID 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 volume group 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*.

sparse tape

A tape containing only a small amount of active information.

special file

UNIX special files are never migrated by DMF.

state field

The field in the inode that shows the current migration state of a file.

tape block

See *block*.

tape chunk

See *chunk*.

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.

unmigratable file

A file that the daemon will never select as a migration candidate.

unmigrate

See *recall*.

VG

See *volume group*

voided BFID-set state

A 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 BFID from the user file inode and soft-deleting all associated database entries.

VOL records

The volume (VOL) records in the tape MSP/LS database that contain information about each tape volume that exists in the pool of tapes used by the tape MSP/LS.

volume group

One of the components of a library server. A volume group is responsible for copying data blocks onto alternate media. Each volume group 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 volume groups. See also *library server*.

volume merging

The mechanism provided by the tape MSP/LS for copying active data from volumes that contain largely obsolete data to volumes that contain mostly active data.

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.

Index

A

absolute block positioning
  definition, 11
ADMIN_EMAIL configuration parameter
  base object
    definition, 39
$ADMINDIR directory
  daemon maintenance tasks, 46
  MSP maintenance tasks, 92
administration
  overview, 14
administrative tasks
  daemon configuration, 44
    TASK_GROUPS parameter, 43
  filesystem backups, 16
    configuring automated tasks, 48
  overview, 14
  overview of automated maintenance tasks, 31
  tape management
    configuring automation, 91
administrative tips, 187
age expression
  configuration file
    definition, 56
AGE_WEIGHT configuration parameter
  definition, 55
all keyword
  dmvoladm command, 157
allocation group, 10
application data flow, 2
architecture
  overview, 12
atmsp_db journal file
  dmatmsp, 141
atmsp_db.dbd
  database definition file, 141, 193
atmsp_db.dbd database definition file, 140
autolog log file, 119
  message format, 109
Automated maintenance tasks
  daemon configuration
    TASK_GROUPS parameter, 43
automated maintenance tasks
  overview, 31
Automated space management
  configuration parameters
    definitions, 51
  filesystem configuration
    MIGRATION_LEVEL parameter, 51
automated space management
  candidate list generation, 116
  configuration parameters
    definitions, 53
  daemon configuration
    MIGRATION_LEVEL parameter, 42
  log file, 119
    message format, 109
  relationship of targets, 118
  selection of migration candidates
    configuration parameters, 116
    file exclusion, 116
    FREE_SPACE_DECREMENT configuration parameter, 118
    FREE_SPACE_MINIMUM configuration parameter, 117
    FREE_SPACE_TARGET configuration parameter, 117
    MIGRATION_TARGET configuration parameter, 117
automated space management commands
  overview, 21
automounters
  supported, 11

B

backups
  of daemon database
    configuring automated task, 47
base object
  configuration, 38
  configuration file
    definition, 37
  configuration parameters
    definitions, 38
BFID
  definition, 12
BFID record
  dmcatadm text field order, 154
  dmdadm text field order, 128
bit file identifier
  See "BFID", 12
BLOCK_SIZE configuration parameter
  device object
    definition, 67
blocks
  DMF tape concepts, 137
blocksize keyword
  dmvoladm command, 157
blocksize record
  dmvoladm text field order, 165, 166
bolume merging
  configuration of automated task, 93

C

CACHE_DIR configuration parameter
  dmatmsp, 146
    definition, 62
CACHE_SPACE configuration parameter
  dmatmsp, 146
    definition, 63
CANCEL message
  FTP MSP, 170
candidate list, 55
  creation, 115
  definition, 15
  generation, 116
candidates for migration
  file exclusion, 116
  file selection, 116
    FREE_SPACE_DECREMENT configuration parameter, 118
    FREE_SPACE_MINIMUM configuration parameter, 117
    FREE_SPACE_TARGET configuration parameter, 117
    MIGRATION_TARGET configuration parameter, 117
  relationship of space management targets, 118
capacity
  of DMF, 13
capacity record
  dmvoladm text field order, 165, 166
CAT database
  backup, 192
  message format comparison, 197, 198
  message interpretation, 199
CAT records
  dmatmsp/dmatls database, 136
  tape MSP/LS database directories, 140
checkage keyword
  dmdadm command, 125
checktime keyword
  dmdadm command, 125
  dmdadm text field order, 128
CHILD_MAXIMUM configuration parameter
  dmatmsp
    definition, 63
  dmdskmsp
    definition, 102
  dmftpmsp
    definition, 96
chkconfig command
  initial configuration, 37
chunkdata keyword

  dmcatadm command, 150
chunkdata record
  dmcatadm text field order, 154
chunklength keyword
  dmcatadm command, 150
chunklength record
  dmcatadm text field order, 154
chunknumber keyword
  dmcatadm command, 150
chunknumber record
  dmcatadm text field order, 154
chunkoffset keyword
  dmcatadm command, 150
chunkoffset record
  dmcatadm text field order, 154
chunkpos keyword
  dmcatadm command, 150
chunks
  DMF tape concepts, 138
chunksleft keyword
  dmvoladm command, 158
chunksleft record
  dmvoladm text field order, 165, 166
client-only user commands, 4
COMMAND configuration parameter
  dmatmsp
    definition, 63
  dmdskmsp
    definition, 102
  dmftpmsp
    definition, 96
configuration
  command overview, 19
  installing binary files, 26
  overview, 25
  tape MSPs/LSs
    setting up, 94
  verifying, 108
configuration file
  automated space management configuration, 53
  base object configuration, 38
  daemon object configuration, 42
  daemon_tasks object, 45
  DCM configuration, 105
  device object configuration, 66
    OpenVault mounting service, 68
    TMF mounting service, 69
  disk MSP configuration, 101
  dump_tasks object, 48
  file weighting parameters, 55
  filesystem object configuration, 51
  FREE_SPACE_DECREMENT configuration
      parameter, 118
  FREE_SPACE_MINIMUM configuration
      parameter, 117
  FREE_SPACE_TARGET configuration
      parameter, 117
  FTP MSP configuration, 96
  MIGRATION_TARGET configuration
      parameter, 117
  MSP or volume group selection parameters, 55
  msp_tasks object, 91
  OpenVault mounting service configuration, 85
  policy object configuration, 52
  space management parameters, 116
  tape MSP configuration, 62
configuration objects
  configuration file, 37
  definition, 19
configuration parameters
  automated space management
    definitions, 53
  base object
    definitions, 38
  daemon object
    definitions, 42
  DCM
    definitions, 105
  definition, 19
  device object
    definitions, 66
    OpenVault mounting service, 68
    TMF mounting service, 69

disk MSP
definitions, 101
file weighting
definitions, 55
procedure for configuring, 57
filesystem object
definitions, 51
FTP MSP
definitions, 96
HOME_DIR, 137
JOURNAL_DIR, 137, 141
dmfdaemon and, 129
JOURNAL_SIZE
dmfdaemon and, 130
tape MSP/LS and, 142
MSP or volume group selection
definitions, 55
procedure for configuring, 59
policy object
definitions, 52
SPOOL_DIR, 119, 129, 137
tape MSP
definitions, 62
procedure for configuring, 65
configuration requirements, 27
configure button
dmmaint utility, 34
conversion
tape MSP to LS, 178
count directive
dmcatadm command, 148
dmdadm command, 123
dmvoladm command, 155
cpio command
file recall, 189
create directive
dmcatadm command, 148
dmvoladm command, 155

D

Daemon
database
configuring automated verification task, 47
daemon
commands
overview, 20
configuration parameters
definitions, 42
configuring automated maintenance tasks, 44
database, 122
automating copying for reliability, 47
backup, 192
directory location, 122
selection, 192
database record length, 29
procedure for configuring, 29
database recovery, 193
dmdadm command, 123
log file
message format, 109
logs and journals, 129
processing, 121
shutdown, 122
daemon database
message format comparison, 198, 197
recovery example, 194
daemon object
configuration, 42
configuration file
definition, 37
daemon startup, 121
daemon_tasks object
configuration, 45
parameters
definitions, 45
data integrity
administrative tasks and, 16
copying filesystem data
configuring automated tasks, 48

overview, 11
data reliability
administrative tasks and, 16
copying daemon database
configuring automated task, 47
copying filesystem data
configuring automated tasks, 48
data types
distributed commands, 205
DATA_LIMIT parameter
msp_tasks object
configuration, 93
database definition file
atmsp_db.dbd, 140, 141, 193
dmd_db.dbd, 108, 192
libsrv_db.dbd, 141, 193
database journal files
dmlockmgr process, 131
DATABASE_COPIES parameter
daemon_tasks object
configuration, 47
databases
CAT
backup, 192
daemon, 122, 193
backup, 192
configuring record length, 29
database record length, 29
directory location, 122
dmcatadm message interpretation, 199
dmvoladm message interpretation, 201
example of recovery, 194
message format for comparisons, 198, 197
tape MSP/LS recovery, 193
VOL
backup, 192
dataleft keyword
dmvoladm command, 158
dataleft record
dmvoladm text field order, 165, 166
datalimit keyword
dmvoladm command, 160
datawritten keyword
dmvoladm command, 158
datawritten record
dmvoladm text field order, 165, 166
dbrec.dat file, 192
dbrec.keys file, 192
DCM
configuration parameters
definitions, 105
delete directive
dmcatadm command, 148
dmdadm command, 123
dmvoladm command, 155
deleteage keyword
dmdadm command, 125
deletetime keyword
dmdadm command, 126
dmdadm text field order, 128
Dependencies button
dmmaint utility, 35
device object
configuration parameters
definitions, 66
OpenVault mounting service, 68
TMF mounting service, 69
device objects
configuration file
definition, 37
directories
daemon database, 122
directory structure prior to DMF 2.8, 227
disk cache manager (DCM), 7
disk MSP, 171
configuration parameters
definitions, 101
log files, 172
request processing, 171
disk space capacity
handling, 6
DISK_IO_SIZE configuration parameter
dmatmsp

definition, 63
dmdskmsp
definition, 102
dmftpmsp
definition, 97
distributed commands
data types, 205
overview, 203
user-accessible API routines, 210
DMAPI requirement, 5
dmatls, 136
journal files, 141
log files, 142
dmatmsp, 136
CAT database records, 140
configuration parameters
definitions, 62
procedure for configuring, 65
directories, 137
dmvoladm command, 155
journal files, 141
log files, 142
merging tape volumes, 145
setup, 94
VOL database records, 140
dmatread command, 166
definition, 22
reading MSP/LS volumes, 137
dmatsnf command, 167
definition, 22
reading MSP/LS volumes, 137
dmattr command
definition, 17
dmatvfy command
definition, 23
dmaudit command
definition, 20
dmaudit verifymsp command, 168
dmcatadm command, 147
chunkdata keyword, 150
chunklength keyword, 150
chunknumber keyword, 150
chunkoffset keyword, 150
chunkpos keyword, 150
count directive, 148
create directive, 148
definition, 22
delete directive, 148
directives
syntax, 148
dump directive, 148
entry keyword, 152
example of list directive, 153
filesize keyword, 150
flags keyword, 150
format keyword, 152
help directive, 148
limit keywords, 152
list directive, 148
load directive, 148
mspname keyword, 152
quit directive, 148
readage keyword, 151
readcount keyword, 151
readdate keyword, 151
recordlimit keyword, 152
recordorder keyword, 152
set directive, 148
text field order, 154
update directive, 148
verify directive, 148
volgrp keyword, 151
vsn keyword, 151
writeage keyword, 151
writedate keyword, 151
zoneblockid keyword, 151
zonenumber keyword, 151
zonepos keyword, 151
dmcatadm directives, 147
field keywords, 150
dmcheck command
definition, 20
dmclripc command

definition, 23
dmcollect command
definition, 23
dmconfig command
definition, 20
dmcopy command
definition, 17
dmd_db journal file, 129
dmd_db.dbd
database definition file, 108, 192
dmdadm command, 122
checkage keyword, 125
checktime keyword, 125
count directive, 123
create directive, 123
definition, 20
deleteage keyword, 125
deletetime keyword, 126
directives, 123
field keywords, 125
format keywords, 125
syntax, 124
dmdump
text field order, 128
dump directive, 124
example of list directive, 127
format keyword, 127
help directive, 124
limit keywords, 127
list directive, 124
load directive, 124
mspkey keyword, 126
mspname keyword, 126
origage keyword, 126
origdevice keyword, 126
originode keyword, 126
origname keyword, 126
origsize keyword, 126
origtime keyword, 126
origuid keyword, 126
quit directive, 124
recordlimit keyword, 127
recordorder keyword, 127
selection expression, 124
set directive, 124
text field order, 128
update directive, 124
updateage keyword, 126
updatetime keyword, 126
dmdate command
definition, 23
dmdbcheck command
definition, 20, 23
dmdbrecover command
database recovery, 193
definition, 21
dmdidle command
definition, 21
dmdlog log file, 121, 129
message format, 109
dmdskfree command
definition, 23
dmdskmsp, 7, 171
dmdstat command
overview, 21
dmdstop command, 108
daemon shutdown, 122
definition, 21
dmdump command
definition, 23
text field order, 154
dmdump directive
text field order, 165
dmdumpj command
definition, 23
DMF
shutdown, 108
DMF initialization, 108
DMF state information
extended attribute structure, 28
DMF user library, 203
dmf.conf man page
definition, 20

dmfdaemon command, 121
  definition, 20
dmfill command
  definition, 24
  file restoration, 191
dmfind command
  definition, 17
dmfsfree command
  candidate list creation, 115
  definition, 21
  migration target and, 115
dmfsmon command, 53
  candidate list creation, 115
  candidate list generation, 116
  candidate selection, 116
  configuration parameters, 116
  definition, 22
  file exclusion, 116
dmftpmsp, 168
  configuration parameters
    definitions, 96
dmget command
  definition, 17
dmhdelete command
  definition, 21
dmlocklog log file
  message format, 109
dmlockmgr command
  definition, 24
dmlockmgr process, 131
  abort, 133
  communication and log files, 131
  database journal files, 131
  interprocess communication, 132
  log file
    message format, 109
  shutdown, 133
  token files, 132
  transaction log files, 131, 133
dmls command
  definition, 17
dmmaint command
  definition, 24
dmmaint utility, 34
  configure button, 34
  dependencies button, 35
  inspect button, 35
  License Info button, 36
  licensing DMF within, 5
  News button, 35
  Update license button, 36
dmmigrate command
  definition, 21
  file backup, 189
dmmove command
  definition, 24
  moving data between MSPs, 174
  scratch filesystem location
    MOVE_FS configuration parameter, 42
dmov_keyfile command, 87
  definition, 24
dmov_loadtapes command, 90
  definition, 24
dmov_makecarts command, 90
  definition, 24
dmput command
  definition, 17
dmscanfs command
  definition, 22
  uses, 116
dmselect command
  definition, 24
  moving data between MSPs, 174
dmsnap command
  definition, 21
dmsort command
  definition, 24
dmversion command
  definition, 21
dmvoladm command, 155
  all keyword, 157
  blocksize keyword, 157
  chunksleft keyword, 158

count directive, 155
create directive, 155
dataleft keyword, 158
datalimit keyword, 160
datawritten keyword, 158
definition, 22
delete directive, 155
directives, 155, 157
  syntax, 156
dump directive, 155
empty keyword, 157
eotblockid keyword, 158
eotchunk keyword, 158
eotpos keyword, 158
eotzone keyword, 158
examples of list directive, 162
field keywords, 157
flag keywords, 160
format keyword, 160
hbadmnt flag, 160
help directive, 155
herr flag, 160
hflags flag, 160
hfree flag, 161
hfull flag, 161
hlock flag, 161
hoa flag, 161
hro flag, 161
hrsv flag, 161
hsparse flag, 161
label keyword, 158
limit keywords, 160
list directive, 155
load directive, 155
partial keyword, 157
quit directive, 155
recordlimit keyword, 160
recordorder keyword, 160
repair directive, 156
select directive, 145, 156
selection expression, 156
set directive, 156
tapesize keyword, 158
text field order, 165
threshold keyword, 159
upage keyword, 159
update directive, 156
update keyword, 159
used keyword, 157
verify directive, 156
version keyword, 159
volgrp keyword, 159
vsnlist expression, 156
wfage keyword, 159
wfdate keyword, 159
dmxfsrestore command
  definition, 24
drive group, 9
  BLOCK_SIZE option, 72
  DISK_IO_SIZE option, 72
  DRIVE_MAXIMUM option, 72
  DRIVE_SCHEDULER option, 73
  DRIVES_TO_DOWN option, 73
  LABEL_TYPE option, 73
  MAX_MS_RESTARTS option, 73
  MOUNT_SERVICE option, 73
  MOUNT_SERVICE_GROUP option, 73
  MOUNT_TIMEOUT option, 74
  MSG_DELAY option, 74
  OV_ACCESS_MODES option, 74
  OV_INTERCHANGE_MODES option, 74
  POSITION_RETRY option, 75
  POSITIONING option, 74
  REINSTATE_DRIVE_DELAY option, 75
  REINSTATE_VOLUME_DELAY option, 75
  RUN_TASK option, 75
  TASK_GROUPS option, 76
  TMF_TMMNT_OPTIONS option, 76
  TYPE option, 72
  VERIFY_POSITION option, 76
  VOLUME_GROUPS option, 76
  WRITE_CHECKSUM option, 77
drive group object

configuration file
definition, 37
drive groups
with OpenVault, 85
with TMF tapes, 91
Dual-state file
definition, 17
dual-state file
definition, 6
xfsdump and, 190
dump and restore
migrated files, 189
dump directive
dmcatadm command, 148
dmdadm command, 124
dmvoladm command, 155
dump utilities
administrative tasks and, 16
DUMP_DEVICE parameter
dump_tasks object
configuration, 49
DUMP_FILE_SYSTEMS parameter
dump_tasks object
configuration, 50
DUMP_INVENTORY_COPY parameter
dump_tasks object
configuration, 50
DUMP_MIGRATE_FIRST parameter
dump_tasks object
configuration, 50
DUMP_RETENTION parameter
dump_tasks object
configuration, 49
DUMP_TAPES parameter
dump_tasks object
configuration, 49
dump_tasks object
configuration, 48
parameters
definition, 45
DUMP_VSNS_USED parameter
dump_tasks object
configuration, 50

E

empty keyword
dmvoladm command, 157
entries keyword
dmcatadm command, 152
eotblockid keyword
dmvoladm command, 158
eotchunk keyword
dmvoladm command, 158
eotchunk record
dmvoladm text field order, 165, 166
eotpos keyword
dmvoladm command, 158
eotposition record
dmvoladm text field order, 165, 166
eotzone keyword
dmvoladm command, 158
error reports
tapes
configuring automated tasks, 91
extended attribute structure
and DMF states, 28

F

field keywords
dmcatadm command, 150
dmdadm command, 125
dmvoladm command, 157
File concepts
definition, 17
file migration
automated selection of candidates, 116
FREE_SPACE_DECREMENT configuration
parameter, 118

FREE_SPACE_MINIMUM configuration parameter, 117
FREE_SPACE_TARGET configuration parameter, 117
MIGRATION_TARGET configuration parameter, 117
excluding files from, 116
MSP or volume group selection for files
configuration parameter definition, 55
procedure for configuring, 59
overview, 6, 18
real-time partitions and, 119
relationship of space management targets, 118
weighting of files
configuration parameter definition, 55
procedure for configuring, 57
file recall
overview, 18
file weighting configuration parameters
definitions, 55
procedure for configuring, 57
filesize keyword
dmcatadm command, 150
filesize record
dmcatadm text field order, 154
filesystem
backups
configuring automated tasks for retaining, 49
configuration parameters
definitions, 51
conversion
dmdskmsp configuration parameters, 102
dmftpmsp configuration parameters, 97
mount options, 27
filesystem object
configuration, 51
configuration file
definition, 37
FINISH message
FTP MSP, 170
flag keywords
dmvoladm command, 160
flags keyword
dmcatadm command, 150
flags record
dmcatadm text field order, 154
FLEXlm
licensing requirements, 5
FLEXlm license configuration
LICENSE_FILE base object parameter definition, 39
FLUSHALL message
FTP MSP, 171
format keyword
dmcatadm command, 152
dmdadm command, 127
dmvoladm command, 160
format keywords
dmdadm command, 125
free space
managing
overview, 6
FREE_DUALSTATE_FIRST configuration parameter
policy object
definition, 54
FREE_SPACE_DECREMENT configuration parameter
and automated space management, 118
policy object
definition, 54
FREE_SPACE_MINIMUM configuration parameter
and automated space management, 117
policy object
definition, 54
FREE_SPACE_TARGET configuration parameter
and automated space management, 117
policy object
definition, 54
FTP
DMF interoperability, 2
FTP MSP, 168

configuration parameters
definitions, 96
log files, 169
messages, 170
request processing, 168
FTP_ACCOUNT configuration parameter
dmftpmsp
definition, 97
FTP_COMMAND configuration parameter
dmftpmsp
definition, 97
FTP_DIRECTORY configuration parameter
dmftpmsp
definition, 97
FTP_HOST configuration parameter
dmftpmsp
definition, 97
FTP_PASSWORD configuration parameter
dmftpmsp
definition, 97
FTP_PORT configuration parameter
dmftpmsp
definition, 97
FTP_USER configuration parameter
dmftpmsp
definition, 97
fully migrated file
definition, 6

G

gid expression
configuration file
definition, 56
GUARANTEED_DELETES configuration parameter
dmdskmsp
definition, 102
dmftpmsp
definition, 97
GUARANTEED_GETS configuration parameter
dmdskmsp
definition, 102
dmftpmsp
definition, 97

H

hard-deleted files
defined, 189
definition, 19
maintenance/recovery, 188
hardware and software requirements
operating system, 3
hbadmnt keyword
dmvoladm command, 160
help directive
dmcatadm command, 148
dmdadm command, 124
dmvoladm command, 155
herr keyword
dmvoladm command, 160
hflags keyword
dmvoladm command, 160
hflags record
dmvoladm text field order, 165, 166
hfree keyword
dmvoladm command, 161
HFREE_TIME configuration parameter
dmatmsp
definition, 63
hfull keyword
dmvoladm command, 161
hierarchical storage management, 1
hlock keyword
dmvoladm command, 161
hoa keyword
dmvoladm command, 161
HOME_DIR, 27
HOME_DIR configuration parameter
definition, 39

dmatmsp and, 137
HOME_DIR directory
location of, 41
hro keyword
dmvoladm command, 161
hrsv keyword
dmvoladm command, 161
hsparse keyword
dmvoladm command, 161

I

IBM AIX version, 4
id record
dmvoladm text field order, 166
IMPORT_DELETE configuration parameter
dmdskmsp
definition, 102
dmftpmsp
definition, 97
IMPORT_ONLY configuration parameter
dmdskmsp
definition, 102
dmftpmsp
definition, 98
initial configuration
dmmaint utility, 36
initialization
of DMF, 108
inode size
configuration, 28
inspect button
dmmaint utility, 35
inst utility, 108
installation
binary files, 26
interprocess communication (IPC)
configuring operating system parameters, 31
dmlockmgr process, 131, 132
exit cleanup, 132
IRIX version, 3
IRIX/Linux and UNICOS/UNICOS/mk
differences, 229

J

Journal files
configuring automated task for retaining, 47
journal files
dmfdaemon, 129
dmlockmgr process, 131
retaining, 187
tape MSP/LS, 141
JOURNAL_DIR, 27
JOURNAL_DIR configuration parameter
definition, 39
dmatls, 141
dmatmsp, 137, 141
dmfdaemon and, 129
JOURNAL_DIR directory
location of, 41
JOURNAL_RETENTION parameter
daemon_tasks object
configuration, 47
JOURNAL_SIZE configuration parameter
definition, 39
dmfdaemon and, 130
tape MSP/LS and, 142

L

label keyword
dmvoladm command, 158
LABEL_TYPE configuration parameter
device object
definition, 67
lbtype record
dmvoladm text field order, 165, 166
library server
CACHE_DIR option, 71

CACHE_SPACE option, 71
COMMAND option, 71
conversion from tape MSP, 178
drive scheduling, 183
DRIVE_GROUPS option, 71
error analysis and avoidance, 181
MAX_CACHE_FILE option, 71
MESSAGE_LEVEL option, 72
objects, 71
RUN_TASK option, 72
setup, 70
status monitoring, 184
TASK_GROUPS option, 72
TYPE option, 71
WATCHER option, 72
library server object
configuration file
definition, 37
library servers
See "LS", 6
libsrv_db journal file
dmatls, 141
libsrv_db.dbd
database definition file, 141, 193
License Info button
dmmaint utility, 36
LICENSE_FILE configuration parameter
definition, 39
licensing
overview, 6
requirements, 5
limit keywords
dmcatadm command, 152
dmdadm command, 127
dmvoladm command, 160
Linux version, 4
list directive
dmcatadm command, 148
dmdadm command, 124
dmvoladm command, 155
list keyword
dmdadm command
example, 127
load directive
dmcatadm command, 148
dmdadm command, 124
dmvoladm command, 155
lock manager
aborts, 133
communication and log files, 131
database journal files, 131
interprocess communication, 132
RDM, 131
shutdown, 133
token files, 132
transaction log files, 131, 133
Log files
tape MSP/LS, 142
log files
automated space management, 119
configuring automated task for retaining, 47
disk MSP, 172
dmfdaemon, 129
dmlockmgr process, 131, 133
FTP MSP, 169
general format, 108
retaining, 187
LOG_RETENTION parameter
daemon_tasks object
configuration, 47
LS, 136
definition, 6
description, 135
operations, 136
LS configuration example, 82
LS process, 9
LS_NAMES configuration parameter
daemon object
definition, 43

M

maintenance and recovery
  cleaning up journal files, 187
  cleaning up log files, 187
  database backup, 192–194
  dmfill command, 191
  dumping migrated files, 189
  example, 194
  hard-deletes, 188
  restoring migrated files, 189
  soft-deletes, 188
  tape MSP/LS database, 193, 194
maintenance tasks
  automated
    overview, 31
  daemon configuration, 44
maintenance utility, 34
MAX_CACHE_FILE configuration parameter
  dmatmsp, 146
    definition, 63
MAX_CHUNK_SIZE configuration parameter
  dmatmsp
    definition, 63
MAX_PUT_CHILDREN configuration parameter
  dmatmsp
    definition, 64
media concepts, 137
media transports
  supported, 11
media-specific processes
  See "MSP", 6
MERGE_CUTOFF configuration option
  dmatmsp, 146
MERGE_CUTOFF configuration parameter
  dmatmsp
    definition, 64
merging tapes
  configuration of automated task, 93
    stopping automatically, 94
MESSAGE_LEVEL configuration parameter
  daemon object
    definition, 42
  dmatmsp
    definition, 64
  dmdskmsp
    definition, 102
  dmftpmsp
    definition, 98
  filesystem object
    definition, 51
messages
  CAT database, 198, 197
  daemon database, 198, 197
  FTP MSP, 170
  interpretation for dmcatadm, 199
  interpretation for dmvoladm, 201
  log file
    general format, 108
  VOL database, 198
Microsoft Windows Version, 4
migrated data
  moving between MSPs, 173
migrated file
  definition, 17
  recalling, 18
migrating file
  definition, 17
migration
  MSP or volume group selection for files
    configuration parameter definition, 55
    procedure for configuring, 59
  weighting of files
    configuration parameter definition, 55
    procedure for configuring, 57
Migration candidates
  file selection
    FREE_SPACE_MINIMUM configuration parameter, 117
    MIGRATION_TARGET configuration parameter, 117
  relationship of space management targets, 118
migration candidates

file exclusion, 116
file selection, 116
FREE_SPACE_DECREMENT configuration parameter, 118
FREE_SPACE_TARGET configuration parameter, 117
migration of files
overview, 18
migration target
definition, 115
MIGRATION_LEVEL configuration parameter
daemon object
definition, 42
filesystem object
definition, 51
MIGRATION_TARGET configuration parameter
and automated space management, 117
policy object
definition, 54
MIN_TAPES configuration parameter
dmatmsp
definition, 64
mount command
DMF-managed filesystems, 27
MOUNT_SERVICE configuration parameter
device object
definition, 67
mounting services
support for, 28
MOVE_FS configuration parameter
daemon object
definition, 42
MOVE_RS, 27
MSG_DELAY configuration parameter
device object
definition, 67
MSGMAX operating system parameter
configuring, 31
MSGSEG operating system parameter
configuring, 31
MSGSSZ operating system parameter
configuring, 31
MSP
commands, 22
definition, 6
description, 135
disk, 171
dmcatadm message interpretation, 199
dmfdaemon, 136
dmvoladm message interpretation, 201
FTP, 168
log files
and automated maintenance tasks, 47
message format, 109
message format, 198, 197
moving migrated data between MSPs, 173
tape pool
configuring automated task to report status, 93
MSP log files
and automated maintenance tasks, 91
MSP objects, 62
configuration file
definition, 37
MSP or volume group
selection for migrating files
configuration parameter definition, 55
procedure for configuring, 59
MSP or volume group selection configuration parameters
definitions, 55
procedure for configuring, 59
MSP types, 8
MSP/LS
CAT database tape records, 140
dmatread command, 166
dmatsnf command, 167
dmaudit verifymsp command, 168
dmcatadm command, 147
journals, 141
tape
log files, 142
setup, 94

tape operations, 136
tape volume merging, 145
VOL database records for tape, 140
MSP/LS database
CAT records, 136, 140
VOL records, 136, 140
files, 140
MSP_NAMES configuration parameter
daemon object
definition, 43
msp_tasks object
configuration, 91
parameters
definitions, 91
mspkey keyword
dmdadm command, 126
dmdadm text field order, 129
msplog file, 172
dmatls, 142, 144
dmatmsp, 142, 143
LS statistics messages, 144
message format, 109
MSP statistics messages, 143
mspname keyword
dmcatadm command, 152
dmdadm command, 126
dmdadm text field order, 129
MVS_UNIT configuration parameter
dmftpmsp
definition, 98

N

NAME_FORMAT configuration parameter
dmdskmsp
definition, 103
dmftpmsp
definition, 98
News button
dmmaint utility, 35
NFS
DMF interoperability, 2

O

objects
configuration file, 37
offline data management
overview, 15
offline file
definition, 6, 17
OpenVault
enhancements, 28
OpenVault for tape MSPs and drive groups, 85
OpenVault mounting service
configuration, 85
device object configuration parameters, 68
OV_ACCESS_MODES base object parameter
definition, 68
OV_INTERCHANGE_MODES base object parameter
definition, 69
OV_KEY_FILE base object parameter
definition, 39
OV_SERVER base object parameter
definition, 39
origage keyword
dmdadm command, 126
origdevice field
dmdadm text field order, 128
origdevice keyword
dmdadm command, 126
originode keyword
dmdadm command, 126
dmdadm text field order, 128
origname keyword
dmdadm command, 126
dmdadm text field order, 129
origsize keyword
dmdadm command, 126
dmdadm text field order, 128

origtime keyword
dmdadm command, 126
dmdadm text field order, 128
origuid keyword
dmdadm command, 126
dmdadm text field order, 129
OV_ACCESS_MODES configuration parameter
device object
definition, 68
OV_INTERCHANGE_MODES configuration parameter
device object
definition, 69
OV_KEY_FILE configuration parameter
definition, 39
OV_SERVER configuration parameter
definition, 39
overhead
of DMF, 13

P

parameter table, 110
partial keyword
dmvoladm command, 157
pathseg.dat file, 192
pathseg.keys file, 192
periodic maintenance tasks
daemon configuration, 44
POLICIES configuration parameter
dmdskmsp
definition, 103
filesystem object
definition, 51
policy configuration parameters
definitions, 52
policy object
configuration, 52
configuration file
definition, 37
POSITION_RETRY configuration parameter
device object
definition, 67
POSITIONING configuration parameter
device object
definition, 67
ProPack version, 3

Q

quit directive
dmcatadm command, 148
dmdadm command, 124
dmvoladm command, 155

R

RDM
lock manager, 131
aborts, 133
communication and log files, 131
database journal files, 131
interprocess communication, 132
shutdown, 133
token files, 132
transaction log files, 131, 133
readage keyword
dmcatadm command, 151
readcount keyword
dmcatadm command, 151
readcount record
dmcatadm text field order, 154
readdate keyword
dmcatadm command, 151
readdate record
dmcatadm text field order, 154
Readme file
viewing with dmmaint, 35
recall
migrated files, 18

record length
daemon database, 29
procedure for configuring, 29
recordlimit keyword
dmcatadm command, 152
dmdadm command, 127
dmvoladm command, 160
recordorder keyword
dmcatadm command, 152
dmdadm command, 127
dmvoladm command, 160
recovery
daemon database, 193, 194
tape MSP/LS database, 193, 194
Red Hat Linux version, 4
regular file
definition, 17
reliability
copying daemon database
configuring automated tasks, 47
repair directive
dmvoladm command, 156
request processing
disk MSP, 171
FTP MSP, 168
requirements, 3
resource scheduler , 10
ALGORITHM option, 80
MODULE_PATH option, 81
PENALTY option, 81
TYPE option, 80
WEIGHT option, 81
resource scheduler algorithm, 10
resource scheduler object
configuration file
definition, 38
resource watcher, 10
HTML_REFRESH option, 82
TYPE option, 82
resource watcher object
configuration file
definition, 38
restore utilities
migrated files, 189
retention of journal files
configuration of automated task, 47
retention of log files
configuration of automated task, 47
rpm utility, 108
run_audit.sh task
configuration, 47
definition, 45
run_copy_databases.sh task
configuration, 47
definition, 45
run_full_dump.sh task
configuration, 49
definition, 45
run_hard_deletes.sh task
configuration, 49
definition, 45
run_merge_stop.sh task
configuration, 94
run_partial_dump.sh task
configuration, 49
definition, 45
run_remove_journals.sh task
and MSP logs, 47, 91
configuration, 47
definition, 45
run_remove_logs.sh task
and MSP logs, 47, 91
configuration, 47
definition, 45
run_scan_logs.sh task
configuration, 47
definition, 45
run_tape_merge.sh task
configuration, 93
definition, 91
run_tape_report.sh task
configuration, 93
definition, 91

run_tape_stop.sh task
  definition, 91

S

select directive
  dmvoladm command, 156
select system call
  dmfdaemon, 122
SELECT_MSP configuration parameter
  definition, 55
selection expression
  dmvoladm command, 156
set directive
  dmcatadm command, 148
  dmdadm command, 124
  dmvoladm command, 156
shutdown
  DMF, 108
  dmlockmgr process, 133
  inst and rpm, 108
soft-deleted files
  definition, 18, 189
  maintenance/recovery, 188
Solaris Version, 4
space expression
  configuration file
    definition, 56
space management
  commands
    overview, 21
SPACE_WEIGHT configuration parameter
  definition, 55
Sparse tapes
  merging
    configuring automated tasks, 91
sparse tapes
  configuration of automated merging, 93
    stopping automatically, 94
  definition, 15
  merging, 145
SPOOL_DIR, 27
SPOOL_DIR configuration parameter, 119
  definition, 39
  dmatmsp and, 137
  dmfdaemon and, 129
STORE_DIRECTORY configuration parameter
  dmdskmsp
    definition, 103
subsystems
  client and server, 3
support
  mounting services, 28
sutomated maintenance tasks
  daemon configuration, 44

T

tape activity
  configuration of automated task, 93
tape maintenance task configuration, 91
tape management
  error reports
    configuring automated tasks, 91
  merging sparse tapes, 145
    configuring automated tasks, 91
  msp_tasks object
    configuration of automated tasks, 93
tape merging
  configuration of automated task, 93
    stopping automatically, 94
  tape MSP/LS, 145
tape mounting, 28
tape MSP, 136
  configuration parameters
    definitions, 62
    procedure for configuring, 65
  conversion to LS, 178
  setup, 94
tape MSP/LS
  CAT database records, 140

database recovery, 193
database recovery example, 194
directories, 137
dmatread command, 166
dmatsnf command, 167
dmaudit command, 168
dmcatadm command, 147
dmvoladm command, 155
journals, 141
log files, 142
merging tape volumes, 145
VOL database records, 140
tape MSPs
with OpenVault, 85
with TMF tapes, 91
tape reports
configuration of automated task, 93
TAPE_TYPE configuration parameter
dmatmsp
definition, 64
tapesize keyword
dmvoladm command, 158
tar command
file recall, 189
task
automated maintenance tasks
overview, 31
definition, 14
TASK_GROUPS configuration parameter
daemon object
definition, 43
dmatmsp object
definition, 65
dmdskmsp object
definition, 103
dmftpmsp object
definition, 99
filesystem object
definition, 51
taskgroup objects
configuration file
definition, 37
Text field order
dmcatadm command, 154
text field order
dmdadm command, 128
dmvoladm command, 165
threshold keyword
dmvoladm command, 159
THRESHOLD parameter
msp_tasks object
configuration, 93
time_expression configuration
daemon maintenance tasks, 46
MSP maintenance tasks, 92
TIMEOUT_FLUSH configuration parameter
dmatmsp
definition, 65
TMF
enhancements, 28
TMF mounting service
device object configuration, 69
TMF tapes, 91
TMF_TMMNT_OPTIONS configuration parameter
dmatmsp
definition, 69
TMP_DIR, 27
TMP_DIR configuration parameter
definition, 40
token files
dmlockmgr process, 132
tpcrdm.dat file, 192
definition, 140
tpcrdm.key1.keys file, 192
definition, 140
tpcrdm.key2.keys file, 192
definition, 140
tpvrdm.dat file, 192
definition, 141
tpvrdm.vsn.keys file, 192
definition, 141
transaction processing, 12

transports
  supported, 11
TYPE configuration parameter
  base object
    definition, 39
  daemon object
    definition, 42
  device object
    definition, 66
  filesystem object
    definition, 51
  msp object
    definition, 62
  policy object
    definition, 54

U

uid expression
  configuration file
    definition, 56
UNICOS differences, 229
upage keyword
  dmvoladm command, 159
update directive
  dmcatadm command, 148
  dmdadm command, 124
  dmvoladm command, 156
update keyword
  dmvoladm command, 159
Update License button
  dmmaint utility, 36
update record
  dmvoladm text field order, 166
updateage keyword
  dmdadm command, 126
updatetime keyword
  dmdadm command, 126
  dmdadm text field order, 128
used keyword
  dmvoladm command, 157
user interface
  commands, 17
/usr/dmf/dmbase, 227

V

V record
  dmvoladm text field order, 165, 166
verification
  of configuration, 108
  of daemon database integrity
    configuration of automated task, 47
verify directive
  dmcatadm command, 148
  dmvoladm command, 156
VERIFY_POSITION configuration parameter
  dmatmsp
    definition, 68
version keyword
  dmvoladm command, 159
version record
  dmvoladm text field order, 165, 166
vista.taf file
  dmlockmgr process, 134
VOL database
  backup, 192
  message format comparison, 198
  message interpretation, 201
VOL database records, 136
  tape MSP/LS, 140
    files, 140
volgrp keyword
  dmcatadm command, 151
  dmvoladm command, 159
volgrp record
  dmvoladm text field order, 166
volume group, 10
  ALLOCATION_GROUP option, 77
  ALLOCATION_MAXIMUM option, 77
  ALLOCATION_MINIMUM option, 78

DRIVE_MAXIMUM option, 78
HFREE_TIME option, 78
MAX_CHUNK_SIZE option, 78
MAX_PUT_CHILDREN option, 79
MERGE_CUTOFF option, 79
MIN_VOLUMES option, 79
PUTS_TIME option, 79
READ_TIME option, 80
RUN_TASK option, 80
TASK_GROUP option, 80
TIMEOUT_FLUSH option, 80
TYPE option, 77
ZONE_SIZE option, 80
volume group object
configuration file
definition, 38
volume merging
configuration of automated task
stopping automatically, 94
definition, 11
tape MSP/LS, 145
volume-to-volume merging
tape MSP/LS, 145
VOLUME_LIMIT parameter
msp_tasks object
configuration, 93
vsn keyword
dmcatadm command, 151
vsn record
dmvoladm text field order, 165, 166
vsnlist expression
dmvoladm command, 156

W

weighting
of files for migration
configuration parameter definition, 55
procedure for configuring, 57
wfage keyword
dmvoladm command, 159
wfdate keyword
dmvoladm command, 159
wfdate record
dmvoladm text field order, 165, 166
when clause
configuration file
definition, 56
Windows version, 4
WRITE_CHECKSUM configuration parameter
device object
definition, 68
writeage keyword
dmcatadm command, 151
writedata record
dmcatadm text field order, 154
writedate keyword
dmcatadm command, 151

X

xfsdump command, 189
xfsrestore command, 189
xinetd, 5

Z

ZONE_SIZE configuration parameter
dmatmsp
definition, 68
zoneblockid keyword
dmcatadm command, 151
zonenumber keyword
dmcatadm command, 151
zonepos keyword
dmcatadm command, 151
zoneposition record
dmcatadm text field order, 154
zones
DMF tape concepts, 138