sgi

DMF Administrator's Guide for SGI® InfiniteStorage

COPYRIGHT

© 1997, 1998, 2000, 2002, 2003, 2004, 2005, 2006, 2007, 2008 SGI. All Rights Reserved; provided portions may be copyright in third parties, as indicated elsewhere herein. No permission is granted to copy, distribute, or create derivative works from the contents of this electronic documentation in any manner, in whole or in part, without the prior written permission of SGI.

LIMITED RIGHTS LEGEND

The software described in this document is "commercial computer software" provided with restricted rights (except as to included open/free source) as specified in the FAR 52.227-19 and/or the DFAR 227.7202, or successive sections. Use beyond license provisions is a violation of worldwide intellectual property laws, treaties and conventions. This document is provided with limited rights as defined in 52.227-14.

TRADEMARKS AND ATTRIBUTIONS

SGI, Altix, the SGI cube and the SGI logo are registered trademarks and CXFS, SGI ProPack, and OpenVault are trademarks of Silicon Graphics, Inc., in the United States and/or other countries worldwide.

AMPEX is a trademark of Ampex Corporation. Atempo and Time Navigator are trademarks or registered trademarks of Atempo S.A. and Atempo, Inc. DLT is a trademark of Quantum Corporation. FLEXIm is a trademark of Macrovision Corporation. IBM is a trademark and MVS is a product of International Business Machines Corporation. Intel and Itanium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries. LEGATO and LEGATO Networker are trademarks or registered trademarks of LEGATO Systems, Inc. Linux is a registered trademark of Linux Torvalds. FLEXIm is a registered trademark of Macrovision Corporation. MIPSpro is a trademark of MIPS Technologies, Inc., used under license by Silicon Graphics, Inc., in the United States and/or other countries worldwide. RedWood, STK, and TimberLine are trademarks of Storage Technology Corporation. Red Hat and all Red Hat-based trademarks are trademarks or registered trademarks of Red Hat, Inc. in the United States and other countries. Solaris and Sun are trademarks or registered trademarks of Sun Microsystems, Inc. UltraSPARC is a registered trademark of SPARC International, Inc. in the United States and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc. Novell is a registered trademark, and SUSE is a trademark of Novell, Inc. in the United States and other countries. UNICOS and UNICOS/mk are federally registered trademarks of Cray, Inc. UNIX is a registered trademark of the Open Group in the United States and other countries. All other trademarks mentioned herein are the property of their respective owners.

New Features in this Guide

This update includes the following:

- Support for the Mac OS X platform as a DMF client. For a complete list of the supported platforms, see "Hardware and Software Requirements" on page 4.
- Support for the HP ULTRIUM 4 drives
- DMF is now released for SGI Altix and SGI Altix XE systems via the ISSP distribution for both server and client. The DMF installation includes the supporting software for OpenVault, Tape Migration Facility (TMF), and APD.
- Updated information about BLOCK_SIZE values in "Drive Group Objects" on page 78
- The configuration parameters have been added to the glossary. See Glossary on page 293.

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

007–3681–019 iii

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.
009	August 2004 Supports DMF 3.1 running under SGI IRIX and Linux systems.
010	October 2004 Supports DMF 3.1 running under SGI IRIX and Linux systems.
011	February 2005 Supports DMF 3.2 running under SGI IRIX and Linux systems.
012	April 2005 Supports DMF 3.3 running under SGI IRIX and Linux systems.

007–3681–019 v

013	October 2005 Supports DMF 3.4 running under SGI IRIX and SGI ProPack for Linux systems.
014	March 2006 Supports DMF 3.5 running under SGI IRIX and SGI ProPack for Linux systems.
015	November 2006 Supports DMF 3.6 running under SGI IRIX and SGI ProPack for Linux systems.
016	March 2007 Supports DMF 3.7 running under SGI IRIX and SGI ProPack for Linux systems.
017	May 2007 Supports DMF 3.8 running under SGI IRIX and SGI ProPack for Linux systems.
018	August 2007 Supports DMF 3.8 running under SGI IRIX and SGI ProPack for Linux systems.
019	March 2008 Supports DMF 3.9 running under SGI IRIX and SGI ProPack for Linux systems.

vi 007–3681–019

Contents

About This Guide	xxv
Related Publications	xxv
Obtaining Publications	xxvii
Conventions	xxvii
Reader Comments	xviii
1. Introduction	1
What Is DMF?	1
DMF Client and Server Subsystems	3
Hardware and Software Requirements	4
DMAPI Requirement	5
Licensing Requirement	6
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	18
Migrating a File	18
Recalling a Migrated File	19
Command Overview	20
Configuration Commands	20
DMF Daemon and Related Commands	20
Space Management Commands	22
007–3681–019	vii

MSP/LS Commands					22
Disk Cache Manager (DCM) Commands					24
Commands for Other Utilities					24
Customizing DMF					26
Partial-State Files		•		•	27
2. Configuring DMF	•	•	•		29
Overview of the Configuration Steps					29
Configuration Considerations					30
Configuration File Requirements					31
Filesystem Mount Options					31
Mounting Service				•	32
Inode Size Configuration					32
Configuring Daemon Database Record Length					33
Interprocess Communication Parameters					35
Automated Maintenance Tasks					36
Using dmmaint to Install the License and Configure DMF					38
Overview of dmmaint					38
Installing the License, Reading News, and Defining the Configuration File					40
Configuration Objects					41
Configuring the Base Object					42
Configuring the DMF Daemon					46
Configuring Daemon Maintenance Tasks					50
Configuring Device Objects					57
Configuring Filesystems					58
Configuring DMF Policies					62
Automated Space Management Parameters					63

viii 007–3681–019

File Weighting and MSP or Volume Group Selection Parameters		64
Configuring Policies		70
DCM Policies		75
Setting Up an LS	•	76
LS Objects	•	76
Drive Group Objects		78
Volume Group Objects		83
Resource Scheduler Objects		87
Resource Watcher Objects		89
Example		89
Using OpenVault for LS Drive Groups		92
Using TMF tapes with LS Drive Groups		96
Configuring Maintenance Tasks for the LS		97
LS Database Records	•	100
Setting Up an FTP MSP		101
Setting Up a Disk MSP	•	106
Setting Up a Disk MSP in DCM Mode	•	110
Verifying the Configuration		113
Initializing DMF		113
General Message Log File Format		114
Parameter Table	•	115
3. Automated Space Management		121
Generating the Candidate List		122
Selection of Migration Candidates		123
Space Management and the Disk Cache Manager	•	125
Automated Space Management Log File		125

007–3681–019 ix

4. The DMF Daemon	27
Daemon Processing	27
DMF Daemon Database and dmdadm	29
dmdadm Directives	30
dmdadm Field and Format Keywords	32
dmdadm Text Field Order	36
Daemon Logs and Journals	36
5. The DMF Lock Manager	39
dmlockmgr Communication and Log Files	39
dmlockmgr Individual Transaction Log Files	1 1
6. Media-Specific Processes and Library Servers	
LS Operations	14
LS Directories	15
Media Concepts	15
CAT Database Records	17
VOL Database Records	18
LS Journals	19
LS Logs	50
Volume Merging	53
dmcatadm Command	54
dmcatadm Directives	55
dmcatadm Keywords	57
dmcatadm Text Field Order	₅₂
dmvoladm Command	53
dmvoladm Directives	53
dmvoladm Field Keywords	56
dmvoladm Text Field Order	73

x 007–3681–019

χi

dmatread Command										174
dmatsnf Command										175
dmaudit verifymsp Command										175
FTP MSP				•	•		•	•		176
FTP MSP Processing of Requests										176
FTP MSP Activity Log										177
FTP MSP Messages					•					178
Disk MSP										178
Disk MSP Processing of Requests										179
Disk MSP Activity Log					•					180
Disk MSP and Disk Cache Manager (DCM)										180
dmdskvfy Command										181
Moving Migrated Data between MSPs and Volume Groups .										181
Converting from an IRIX DMF to a Linux DMF										182
LS Error Analysis and Avoidance										184
LS Drive Scheduling										186
LS Status Monitoring										187
7. DMF Maintenance and Recovery	•	•	•	•	•	•	•	•	•	189
Retaining Old DMF Daemon Log Files		•		•	•		•	•	•	189
Retaining Old DMF Daemon Journal Files					•		•	•		189
Soft- and Hard-Deletes										190
Backups and DMF										191
DMF-managed User Filesystems										192
Using SGI xfsdump and xfsrestore with Migrated Files										192
Ensuring Accuracy with xfsdump										194
Dumping and Restoring Files without the DMF Scripts										195
Filesystem Consistency with xfsrestore										195
	•									

007-3681-019

Using DMF-aware Third-Party Backup Packages	. 196
Using XVM Snapshots and DMF	. 198
Optimizing Backups of Filesystems	. 199
Storage Used by an FTP MSP or a Standard Disk MSP	. 200
Filesystems Used by a Disk MSP in DCM Mode	. 200
DMF's Private Filesystems	. 201
Using dmfill	. 202
Database Recovery	. 202
Database Backups	. 203
Database Recovery Procedures	. 203
Appendix A. Messages	. 207
Message Format	. 207
Message Format for Catalog (CAT) Database and Daemon Database Comparisons .	. 207
Message Format for Volume (VOL) Database and Catalog (CAT) Database and Daemon Database Comparisons	. 208
dmcatadm Message Interpretation	. 209
dmvoladm Message Interpretation	. 211
amvordam neessige interpretation	
Appendix B. DMF User Library libdmfusr.so	. 213
Overview of the Distributed Command Feature and libdmfusr.so	. 213
Considerations for IRIX	. 216
libdmfusr.so Library Versioning	. 216
libdmfusr.so.2 Data Types	. 217
DmuAllErrors_t	. 218
DmuAttr_t	. 219
DmuByteRange_t	. 220
DmuByteRanges_t	. 220

xii 007–3681–019

D	muCompletion_t																			224
D	muCopyRange_t																			224
D	muCopyRanges_t																			225
D	muErrHandler_f																			226
D	muErrInfo_t .																			226
D	muError_t											•								227
D	muEvents_t .									•										227
D	muFhandle_t .																			227
D	muFullRegbuf_t																			228
D	muFullstat_t																			228
D	muRegion_t .																			229
D	muRegionbuf_t																			229
D	muReplyOrder_t																			230
D	muReplyType_t									•										230
D	muSeverity_t																			231
D	muVolGroup_t																			231
D	muVolGroups_t																			231
Use	r-Accessible API Sub	rou	tine	es fo	or I	Lik	odm:	fus	sr.	so.	. 2									232
C	Context Manipulation	Su	bro	utir	nes		•			•		•	•			•			•	232
	DmuCreateContex	t() S	ubr	out	ine)													233
	DmuChangedDirec	to	ry	() 5	Sub	rou	ıtin	e												234
	DmuDestroyConte	ext	()	Sub	oroi	atir	ne													235
Ε	MF File Request Sub	rou	ıtin	es																235
	Copy File Requests																			236
	fullstat Request	s																		238
	put File Requests																			240
	get File Requests									•										243
	settag File Reque	sts																		245
		-										•			-				•	0

007–3681–019 xiii

Request Completion Subroutines	247
DmuAwaitReplies() Subroutine	248
DmuFullstatCompletion() Subroutine	249
DmuGetNextReply() Subroutine	250
DmuGetThisReply() Subroutine	251
Appendix C. Site-Defined Policy Subroutines and the sitelib.so Library	255
Overview of Site-Defined Policy Subroutines	255
Getting Started	256
Considerations	258
sitelib.so Data Types	259
DmaContext_t	259
DmaFrom_t	260
DmaIdentity_t	260
DmaLogLevel_t	262
DmaRealm_t	262
DmaRecallType_t	262
SiteFncMap_t	262
Site-Defined Policy Subroutines	263
SiteCreateContext()	263
SiteDestroyContext()	263
SiteKernRecall()	264
SitePutFile()	265
SiteWhen()	267
Helper Subroutines for sitelib.so	269
DmaConfigStanzaExists()	269
DmaGetConfigBool()	270
DmaGetConfigFloat()	271

xiv 007–3681–019

	DmaGetConfigInt() .																					272
	DmaGetConfigList()																					273
	DmaGetConfigStanza()																					274
	DmaGetConfigString()																					275
	DmaGetContextFlags()																					276
	DmaGetCookie()																					276
	DmaGetDaemonVolGroups	()																				277
	DmaGetProgramIdentity	()																				277
	DmaGetUserIdentity()						•															278
	DmaOpenByHandle() .						•															279
	DmaSendLogFmtMessage()					•															279
	DmaSendUserFmtMessage	()			•									•								280
	DmaSetCookie()																					281
	ppendix D. DMF Direc		•													nk	Di	МІ	•		•	283 285
A	ppendix F. Third-Party	Ва	ack	up	Pa	ıck	ag	e C	Con	ıfig	gur	ati	on							•	•	287
LI	EGATO NetWorker																					287
A	tempo Time Navigator .						•	•														288
Γi	me Navigator for NDMP					•	•															289
G	lossary		•				•	•	•	•	•			•	•		•	•		•	•	29 3
ſτ	ndex																					315

007–3681–019 xv

Figures

Figure 1-1	Application Data Flow														2
igure 1-2	DMF Mechanisms														8
igure 1-3	Library Server Architecture														ç
igure 1-4	DMF Architecture														13
igure 3-1	Relationship of Automated S	Бра	ce :	Ma	nag	gem	ent	Ta	rge	ts					124
igure 6-1	Media Concepts														147
Figure F-1	NDMP Three-way Architectu	ure													291

007–3681–019 xvii

Tables

Table 2-1	Automated Maintenance Task Summary	•	•			•	37
Table 2-2	Default Maximum File Regions for XFS Filesystems						60
Table 2-3	NAME_FORMAT Strings						104
Table 2-4	DMF Log File Message Types						115
Table 2-5	Parameters for the DMF Configuration File						116
Table 5-1	dmlockmgr Token Files						140
Гable Е-1	Differences From UNICOS and UNICOS/mk						285

007–3681–019 xix

Examples

Example 2-1	policy Object for Automated Space Management
Example 2-2	policy Object for Automated Space Management using Ranges
Example 6-1	LS Statistics Messages
Example 6-2	dmcatadm list Directive
Example 6-3	dmvoladm list Directives
Example 6-4	Restoring Hard-deleted Files Using dmatread
Example 6-5	IRIX to Linux Conversion (Single LS)
Example 7-1	Database Recovery Example

007–3681–019 xxi

Procedures

Procedure 2-1	Configuration Steps	29
Procedure 2-2	Daemon Database Record Length Configuration	34
Procedure 2-3	Running dmmaint	40
Procedure 2-4	Base Object Configuration	45
Procedure 2-5	Daemon Configuration	49
Procedure 2-6	Configuring the daemon_tasks Object	51
Procedure 2-7	Configuring the dump_tasks Object	5 4
Procedure 2-8	Configuring filesystem Objects	61
Procedure 2-9	Configuring Objects for Automated Space Management	70
Procedure 2-10	Configuring Objects for MSP or Volume Group Selection	73
Procedure 2-11	Configuring an LS and Its Components	90
Procedure 2-12	Configuring DMF to Use OpenVault	92
Procedure 2-13	Configuring the msp_tasks Object	98
Procedure 2-14	Creating LS Database Records)(
Procedure 2-15	Configuring the ftp Object)6
Procedure 2-16	Configuring the dsk Object)9
Procedure 6-1	IRIX to Linux Conversion	32
Procedure 7-1	Recovering the Databases)4

007–3681–019 xxiii

About This Guide

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

Related Publications

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

If you have an IRIX cluster, 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 the IRIX distribution, there is also a $\protect\operatorname{CDROM/platform/DMF}$. Install for each client platform.

For example, for the IRIX platform see:

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

If you have an SGI ProPack cluster, see the SGI InfiniteStorage Software Platform release notes and the DMF.Readme and DMF.News files included in the package that contains the DMF server RPMs. You can also read these files after installation by using the dmmaint(8) command.

See the following man pages:

```
dmatsnf(8)
dmattr(1)
dmaudit(8)
dmatvfy(8)
dmcheck(8)
dmcollect(8)
```

007–3681–019 xxv

```
dmconfig(8)
dmdadm(8)
dmdate(8)
dmdbcheck(8)
dmdbrecover(8)
dmdidle(8)
dmdskvfy(8)
dmdstat(8)
dmdstop(8)
dmdu(1)
dmdump(8)
dmdumpj(8)
dmfill(8)
dmfind(1)
dmfsfree(8)
dmfsmon(8)
dmfdaemon(8)
dmget(1)
dmhdelete(8)
dmlockmgr(8)
dmmaint(8)
dmmigrate(8)
dmmove(8)
dmov_loadtapes(8)
dmov_makecarts(8)
dmov_keyfile(8)
dmscanfs(8)
dmselect(8)
dmsnap(8)
dmtag(1)
dmversion(1)
dmvoladm(8)
dmxfsrestore(8)
dmxfsprune(8)
sgi_dmdu(1)
sgi_dmfind(1)
sgi_dmls(1)
vi(1)
xfsdump(1M)(IRIX)
xfsdump(8) (Linux)
xfsrestore(1M)(IRIX)
```

xxvi 007–3681–019

xfsrestore(8) (Linux)

For information about running DMF in a high-availability (HA) Linux cluster, see *SGI InfiniteStorage High Availability Using Linux-HA Heartbeat*.

For information about XVM, see the XVM Volume Manager Administrator's Guide.

Obtaining Publications

You can obtain SGI documentation as follows:

- See the SGI Technical Publications Library at http://docs.sgi.com. Various formats are available. This library contains the most recent and most comprehensive set of online books, release notes, man pages, and other information.
- On IRIX systems, you can use InfoSearch (if installed), an online tool that provides a more limited set of online books, release notes, and man pages. Enter infosearch at a command line or select **Help > InfoSearch** from the Toolchest.
- On IRIX systems, you can view release notes by entering either grelnotes or relnotes at a command line.
- On Linux systems, you can view release notes on your system by accessing the README file(s) for the product. This is usually located in the /usr/share/doc/productname directory, although file locations may vary.
- On IRIX and Linux systems, you can view man pages by typing man *title* at 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.

007–3681–019 xxvii

variable Italic typeface denotes variable entries and words or

concepts being defined.

user input This bold, fixed-space font denotes literal items that the

user enters in interactive sessions. (Output is shown in

nonbold, fixed-space font.)

[] Brackets enclose optional portions of a command or

directive line.

.. Ellipses indicate that a preceding element can be

repeated.

Reader Comments

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

You can contact SGI in any of the following ways:

Send e-mail to the following address:

techpubs@sgi.com

- Contact your customer service representative and ask that an incident be filed in the SGI incident tracking system.
- Send mail to the following address:

SGI Technical Publications 1140 East Arques Avenue Sunnyvale, CA 94085–4602

SGI values your comments and will respond to them promptly.

xxviii 007–3681–019

Introduction

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

- "What Is DMF?" on page 1
- "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 18
- "Command Overview" on page 20
- "Customizing DMF" on page 26
- "Partial-State Files" on page 27

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. You can also manually force a file to be migrated or recalled.

007–3681–019

DMF can migrate data to the following:

- Disk
- Tape
- Another server
- Disk cache on serial ATA disk and then to tape, providing multiple levels of migration using *n*-tier capability

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

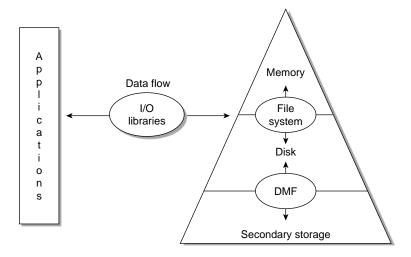


Figure 1-1 Application Data Flow

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.

2 007–3681–019

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 fileserver.

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 Subsystems

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 SGI ProPack for Linux systems on SGI Altix servers and to IRIX systems. You should install this subsystem on only those IRIX and SGI ProPack machines from which you will administer DMF, including NFS servers and potential CXFS metadata servers.
- 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 4). You should install this subsystem on machines from which you want to give users access to DMF user commands, such

007–3681–019

as dmput and dmget. IRIX and SGI ProPack machines that are CXFS client-only nodes or NFS clients should use this subsystem; for other supported operating systems, such as Solaris, the client subsystem is the only option.

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

Hardware and Software Requirements

The DMF server subsystem runs on the following operating systems:

Note: For the currently supported release levels, libraries, and tape devices, see the DMF.Readme file.

- IRIX 6.5.30
- SGI ProPack for Linux 5 Service Pack (SP) 5 for SGI Altix and SGI Altix XE

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

- IRIX 6.5.28
- IRIX 6.5.29
- IRIX 6.5.30
- Mac OS X Tiger (10.4.8 or later) for PPC and Intel
- Mac OS X Leopard (10.5) for PPC and Intel
- Red Hat Enterprise Linux 4 Update 5 (RHEL 4 U5) for x86_64 and ia64
- Red Hat Enterprise Linux 5 (RHEL 5) for x86_64 and ia64
- SGI ProPack 5 SP 2 for Altix and Altix XE
- SGI ProPack 5 SP 3 for Altix and Altix XE
- SGI ProPack 5 SP 4 for Altix and Altix XE
- SGI ProPack 5 SP 5 for Altix and Altix XE
- Solaris 10
- SUSE Linux Enterprise Server 10 Service Pack 1 (SLES 10 SP1) for x86_64 and ia64

4 007–3681–019

The client-only user commands are as follows:

```
dmattr
dmcopy
dmdu
dmget
dmfind
dmls
dmput
dmtag
```

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

Any DMF server subsystem that will be required to accept DMF requests from a DMF client-only platform will require the inetd(1M) (IRIX) or xinted(8) (SGI ProPack) daemon to be initiated via tcpmux (RFC 1078) for successful client-server communication.

DMAPI Requirement

For filesystems to be managed by DMF, they must be mounted on the DMF server in order to enable the Data Management API (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 dmi -o mtpt = mountpoint
```

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

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

007–3681–019 5

Licensing Requirement

You must have a separate license for each DMF server (no licensing is required on the client host):

- For IRIX systems, DMF servers use software licensing based on the FLEXIm product from Macrovision Corporation.
- For SGI ProPack 5 systems, DMF servers use software licensing based on License Keys (LK). LK is an SGI Linux software licensing product.

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. When requesting your permanent DMF license, you must submit the required information for your system type (this information may include information such as the unique system identifier, system serial number, hostname, and entitlement ID). To determine the required information, launch dmmaint and select **License Info**.

For more information about licensing, see the following webpage:

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

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.

How DMF Works

As a DMF administrator, you determine how disk space capacity is handled by selecting the filesystems that 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).

6 007–3681–019

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 *library server* (LS) and the *media-specific process* (MSP):

- *LS* (dmatls) transfers to and from magnetic tape in a tape library (also known as a *robotic library* or *silo*).
- FTP MSP (dmftpmsp) uses the file transfer protocol to transfer to and from disks of another system on the network.
- *Disk MSP* (dmdskmsp) uses a filesystem mounted on the DMF server itself. This can be a local filesystem or a remote one mounted through NFS or similar filesharing protocol.
- *Disk cache manager (DCM)* is the disk MSP configured 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. If the disk MSP is configured as a DCM, the filesystem used by the DCM must be a local XFS filesystem.

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

Figure 1-2 summarizes using DMF.

007–3681–019

Before DMF



After DMF

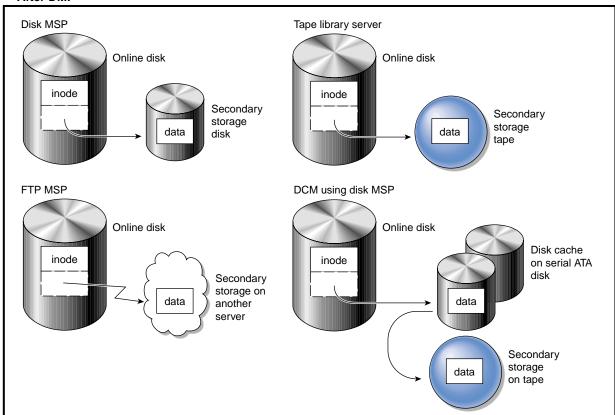


Figure 1-2 DMF Mechanisms

8 007–3681–019

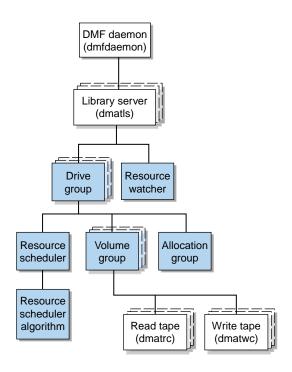


Figure 1-3 shows the architecture of the LS.

Figure 1-3 Library Server Architecture

There is one LS process (dmatls) per tape library, which maintains a database that all of its components share. The entities in the shaded boxes in Figure 1-3 are internal components of the dmatls process. Their functions are as follows:

Drive group

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

The main 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.

The volume group holds at most one copy of user files Volume group

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.

The dmatrc and dmatwc processes are called the read- and write-children. They are created by 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.

10 007-3681-019 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 4.

Ensuring Data Integrity

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

Much of the work done by DMF involves transaction processing that is recorded in databases. The DMF 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.

See "DMF Administration" on page 14 for more information

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 LS accrues requests until the volume of data justifies a volume mount. Requests for data retrieval are satisfied as they arrive. When multiple retrieval requests involve the same volume, all file data is retrieved in a single pass across the volume.

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

Figure 1-4 illustrates the DMF architecture.

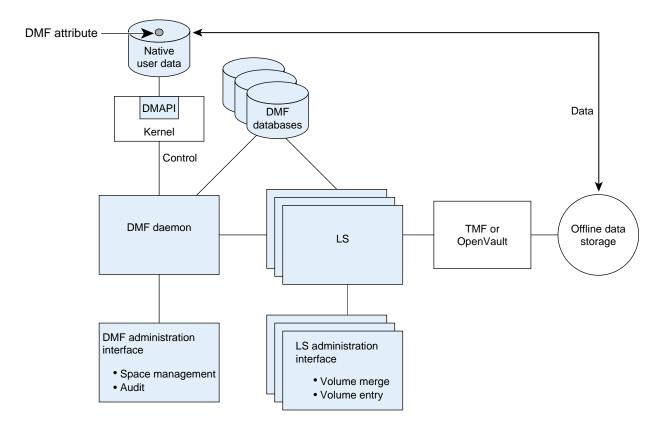


Figure 1-4 DMF Architecture

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 fileservers
- 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 50 and "Configuring Maintenance Tasks for the LS" on page 97.

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 122.
- 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 121.
- 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 volume group, you assign a pool of tapes for use by that volume group. The dmvoladm(8) utility provides management of the 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:

- dmatsnf(8) utility is used to scan a DMF volume for flaws
- dmatread(8) is used for recovering data.

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

Chapter 6, "Media-Specific Processes and Library Servers" on page 143, 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 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 19.

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.
- Determine the number of blocks contained in specified files and directories on a DMF-managed filesystem by using the dmdu(1) command.
- Allow a site-assigned 32-bit integer to be associated with a specific file (which can
 be tested in the when clause of particular configuration parameters and in
 site-defined policies) by using the dmtag(1) command. See "Customizing DMF" on
 page 26.

Note: The functionality of some of these commands can be modified by site–defined policies; see "Customizing DMF" on page 26.

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 one 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
 - Partial-state files are files with some combination of dual-state, offline, and/or unmigrating regions (see "Partial-State Files" on page 27)

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

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

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

Migrating a File

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

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 dmfsfree requested that the file's space be released, the daemon releases the data blocks and changes the user file state to offline.

For more information, see the dmput man page.

Recalling a Migrated File

When a migrated file must be recalled, a request is made to the DMF daemon. The daemon selects an MSP or 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*, meaning that it is logically deleted from the daemon database. This is accomplished by setting the delete date field in the database to the current date and time for each entry referring to the modified or removed file.

A file is *hard-deleted* when its BFID is 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 volume group, this involves volume merging.

Command Overview

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

Note: The functionality of some of these commands can be affected by site-defined policies; see "Customizing DMF" on page 26.

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 29. 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 127, 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 ${\tt dmfdaemon}$ 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 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 LS databases with journal entries.
<pre>dmdidle(8)</pre>	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 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(1)	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 121, for details.

MSP/LS Commands

The LS maintains a database that contains the following:

- Volume (VOL) records, which contain information about tape volumes
- Catalog (CAT) records, which 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 178 and "FTP MSP" on page 176.

The following commands manage the CAT and VOL records for the 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 merge operations.

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

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

There are also tools that check for LS database inconsistencies:

Command	Description
dmatvfy(8)	Verifies the 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.

 ${\tt dmdskvfy(8)} \qquad \qquad {\tt Verifies\ disk\ MSP\ file\ copies\ against\ the\ dmfdaemon}$

database.

Disk Cache Manager (DCM) Commands

The following commands support the DCM:

Command	Description
dmdskfree(8)	Manages file space within the disk cache and as needed migrates files to tape or removes them from the disk cache.

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.

dmmaint(8) Performs DMF maintenance and provides interfaces for

licensing and initial configuration.

dmmove(8) Moves copies of a migrated file's data to the specified

MSPs/volume groups.

dmmvtree(8) Moves files from one DMF-managed filesystem to

another without requiring file data to be recalled.

dmov_keyfile(8) Creates the file of DMF OpenVault keys, ensuring that

the contents of the file are semantically correct and have the correct file permissions. This command removes any DMF keys in the file for the OpenVault server

system and adds new keys at the front of the file.

dmov_loadtapes(8)

Scans a tape library for volumes not imported into the OpenVault database and allows the user to select a portion of them to be used by a volume group. The

selected tapes are imported into the OpenVault database, assigned to the DMF application, and added to the LS's database. This command can perform the equivalent actions for the filesystem backup scripts; just use the name of the associated task group instead of the

name of a volume group.

dmov_makecarts(8) Makes the tapes in one or more LS databases accessible

through OpenVault by importing into the OpenVault database any tapes unknown to it and by registering all volumes to the DMF application not yet so assigned. This command can perform the equivalent actions for the filesystem backup scripts; just use the name of the associated task group instead of the name of a volume

group.

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.

007–3681–019 25

Displays a variety of status information about DMF, including details about the requests currently being processed by the daemon, statistics about requests that have been processed since the daemon last started, and details of current tape drive usage by volume groups.

Calls the xfsrestore(1M) command to restore files dumped to tape volumes that were produced by DMF

administrative maintenance scripts.

Customizing DMF

You can modify the default behavior of DMF as follows:

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

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

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

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

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

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

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

AGE_WEIGHT CACHE_AGE_WEIGHT CACHE_SPACE_WEIGHT SELECT_LOWER_VG SELECT_MSP

SELECT_VG SPACE_WEIGHT

For example:

SELECT_VG fasttape when sitetag = 42

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

For more information, see the dmtag(1) man page and the DMF.Readme file.

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

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

For information about the aspects of DMF that may be modified, see the /usr/share/doc/dmf-*/info/sample/sitelib.readme file.

Partial-State Files

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

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

Note: You should use MAX_MANAGED_REGIONS cautiously. If set capriciously, filesystem scan times can increase greatly. For details about using MAX_MANAGED_REGIONS, see "Configuring Filesystems" on page 58.

Partial-state files provide the following capabilities:

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

You turn off the partial-state file feature by setting the PARTIAL_STATE_FILES daemon configuration parameter to off. For details, see the information in the DMF. News file.

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 30
- "Using dmmaint to Install the License and Configure DMF" on page 38
- "Configuration Objects" on page 41
- "Verifying the Configuration" on page 113
- "Initializing DMF" on page 113
- "General Message Log File Format" on page 114
- "Parameter Table" on page 115

Overview of the Configuration Steps

To configure DMF, you will perform the following steps. Before starting, read "Configuration Considerations" on page 30.

Procedure 2-1 Configuration Steps

- 1. Install DMF:
 - For IRIX clusters, according to the platform-specific instructions in the /CDROM/platform/DMF.Install file.
 - For SGI ProPack clusters, according to the instructions in the SGI InfiniteStorage Software Platform release note
- 2. Determine how you want to complete periodic maintenance tasks. See "Automated Maintenance Tasks" on page 36.
- 3. Invoke dmmaint(8) (see "Overview of dmmaint" on page 38) to do the following:
 - a. Install the license on each DMF server. (DMF clients do not require a license.)

007-3681-019

- 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 41.

4. Verify the configuration by clicking the **Inspect** button, which runs the dmcheck(8) script. See "Verifying the Configuration" on page 113.

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 113.

Configuration Considerations

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

- "Configuration File Requirements" on page 31
- "Filesystem Mount Options" on page 31
- "Mounting Service" on page 32
- "Inode Size Configuration" on page 32
- "Configuring Daemon Database Record Length" on page 33
- "Interprocess Communication Parameters" on page 35
- "Automated Maintenance Tasks" on page 36

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 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 API (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.

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

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 with the DMAPI interface enabled. Failure to enable DMAPI for DMF-managed user 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 LS checks the availability of the mounting service when it is started and after each occurrence in which an LS write child or read child was unable to reserve its drive. If the mounting service is found to be unavailable, the LS does not start any new child processes until the mounting service is once again available.

If the unavailable mounting service is OpenVault, the LS sends an e-mail message to the administrator, asking that OpenVault be started, 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 LS not only attempts to initiate tmdaemon if it is not up (based on the exit status of tmstat), but it waits until a TMF device in the configuration pending state is configured up before it resumes processing. If TMF cannot be started or if no devices are configured up, the LS sends e-mail to the administrator and polls TMF until a drive becomes available. For 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/LS supplies a path value that is longer than the path field, DMF creates one or more pathseg records to accommodate the extra space.

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

The default size of the path field in the pathseg record is 64. For MSP path values that are just slightly over the size of the dbrec path field, this will result in a large amount of wasted space for each record that overflows into the pathseg record. The ideal situation would be to have as few pathseg records as possible.

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. Change to the rdm directory:

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

- Back up the dmd_db.dbd and dmd_db.ddl files that reside in /usr/lib/dmf/rdm. This will aid in disaster recovery should something go wrong.
- 5. Edit dmd_db.ddl to set the new path field lengths for the dbrec and/or pathseg records.
- 6. Regenerate the new database definition, as follows:

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

7. Back up the new versions of dmd_db.dbd and dmd_db.ddl for future reference or disaster recovery.

8. If the daemon database was dumped to text in step 2, enter the following commands:

```
cd HOME_DIR/daemon
dmdadm -u -c "load textfile"
(textfile was created in step 2)
```

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

```
/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 sysct1(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 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 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 50. Table 2-1 provides 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_FILE_SYSTEMS DUMP_FLUSH_DCM_FIRST DUMP_INVENTORY_COPY DUMP_MAX_FILESPACE DUMP_MIGRATE_FIRST DUMP_RETENTION DUMP_TAPES DUMP_VSNS_USED DUMP_XFSDUMP_PARAMS
	run_hard_deletes	Hard-delete files	Uses DUMP_RETENTION
	run_partial_dump	Partial backup of filesystems	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	
LS	run_compact_tape_report	Create tape reports	
	run_merge_stop	Stop volume merges	
	run_tape_merge	Merge sparse tapes	DATA_LIMIT THRESHOLD VOLUME_LIMIT
	run_tape_report	Create tape reports	
	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 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 MSP or the 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 29, and the dmf.conf(5) man page (available from the **Help** button).

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.

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

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

Inspect

Dependencies

News

• Linux:

/usr/share/doc/dmf-version number/News

License Info

Displays the hostname and unique system identifier (which you need to obtain a DMF server license), the name of the license file, and a short description of the state of any DMF license within the file.

Update License

Lets you make changes to the license file. An editing session is started in a new window displaying a copy of the contents of the license file. You can add or delete licenses as desired. After you exit the editor, positive confirmation is requested before the original license file is replaced by the modified copy. For more information, see "Licensing Requirement" on page 6.

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. For more information about this button, see "Overview of dmmaint" on page 38.
- 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 "Initializing DMF" on page 113 and 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 *device objects*, which define parameters necessary for automatic use of tape devices. Normally, the backup scripts would refer to a DMF drive group to define parameters necessary for accessing tape drives. But if they are to use drives not in use by DMF, a device object may be used to define these parameters.

Device objects are not used by tape LSs; instead, LSs reference drive group objects.

- 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 *task group 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.

007–3681–019 41

- 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 2-5 on page 116.

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

For filesystems, <code>object_name</code> is the mount point. Otherwise, it is chosen by the administrator. <code>object_type</code> 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 <code>dmcheck(8)</code>.

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	Specifies the 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.

DIRECT_IO_MAXIMUM_SIZE Specifies the maximum size of I/O requests when using

O_DIRECT I/O to read from any primary filesystem or when migrating files down the hierarchy from the STORE_DIRECTORY of a disk MSP running in DCM mode. DIRECT_IO_MAXIMUM_SIZE is ignored for a

particular filesystem or DCM store when

DIRECT_IO_SIZE is specified in the configuration stanza for that filesystem or DCM. The legal range of values is 262144 through 18446744073709551615.

The default value is 1048576.

HOME_DIR Specifies the 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 Specifies the base pathname used to construct directory

names for DMF directories in which the daemon and 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 Specifies the maximum size (in bytes) of the database

journal file before DMF closes it and starts a new file.

LICENSE_FILE Specifies the full pathname of the file containing the license used by DMF. The default is as follows:

• For IRIX systems: /var/flexlm/license.dat

For SGI ProPack 5 systems: /etc/lk/keys.dat

There is no need to use this parameter if the default is

being used.

OV_KEY_FILE Specifies the 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 Specifies the name returned by the hostname(1) command on the machine on which the OpenVault

007–3681–019 43

server is running. This parameter only applies when OpenVault is used as the mounting service. The default value is the hostname of the machine on which you are running.

SPOOL_DIR Specifies the 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 Specifies the 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 LS 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 LS Drive Groups" on page 92.

Procedure 2-4 Base Object Configuration

The following example defines a base object:

```
define base
       TYPE
                      base
       ADMIN_EMAIL
                      root@dmfserver
                     /dmf/home
       HOME_DIR
       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 50, 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 36.
- 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 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 2-12, page 92.

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.

EXPORT_QUEUE

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

MESSAGE LEVEL

Specifies the highest message level number that will be written to the daemon log. It must be an integer in the range 0–6; the higher the number, the more messages written to the log file. The default is 2. For more information on message levels, see "General Message Log File Format" on page 114.

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. The filesystem specified must have been mounted with DMAPI enabled. There is no default. Necessary only if you wish to use dmmove.

LS_NAMES or MSP_NAMES

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

The order of the values specified for this parameter is integral to the determination of the MSP 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 drive group object's VOLUME_GROUPS parameter.

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

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 50.

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

RECALL_NOTIFICATION_RATE

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

PARTIAL_STATE_FILES

Enables or disables the DMF daemon's ability to produce partial-state files. If this parameter is set to on or yes, the daemon will correctly process put and get requests that would result in a partial-state file. If this parameter is set to off or no, all put and get requests that require a change to the online status of the file will result in a file that is completely online or offline. That is, any put request that makes any part of the file offline will result in the entire file being made offline. Any get request that would result in any part of the file being brought back online will result in the entire file being brought back online. The default is on.

MAX_VIRTUAL_MEMORY

Note: This parameter is used on IRIX systems only. It is ignored on SGI ProPack systems.

Specifies the maximum number of bytes to which the DMF daemon's virtual memory size is allowed to grow. This parameter is meant to prevent the DMF daemon's memory from growing so large that it overflows its virtual memory addressing limits and aborts. When the daemon has grown to this maximum size, the daemon will disable reading new requests from its existing connections that have already issued more than 50 requests. This throttling will remain in effect until the overall memory usage decreases by 10% as the result of request completions, at which time all connections will be reinstated for input.

While the throttling is active, new commands are free to connect to the daemon and issue requests up to the point where they have

Procedure 2-5 Daemon Configuration

The following example defines a daemon object:

```
define daemon
TYPE dmdaemon
MOVE_FS /move_fs
MIGRATION_LEVEL auto
LS_NAMES lib1 ftp2
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 "Configuring DMF Policies" on page 62, for information about configuring automated space management.

4. Configure MSP_NAMES or LS_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. 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:

- Running dmscanfs(8) on filesystems to collect file information for subsequent use by other scripts and programs (the run_filesystem_scan.sh task)
- Creating a regular report (the run_daily_report.sh task)
- 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 37 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_filesystem_scan.sh at 2:00
        SCAN FAST yes
        RUN_TASK $ADMINDIR/run_daily_report.sh at 3:00
        RUN_TASK
                             $ADMINDIR/run_audit.sh every day \
                                at 23:00
        RUN_TASK
                             $ADMINDIR/run_scan_logs.sh at 00:01
        RUN_TASK
                             $ADMINDIR/run_remove_logs.sh every \
                                day at 1:00
        LOG_RETENTION
        RUN_TASK
                             $ADMINDIR/run_remove_journals.sh every \
                                day at 1:00
        JOURNAL_RETENTION
        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. You must provide the following:

 The run_filesystem_scan.sh task runs dmscanfs(8) on filesystems specified by SCAN_FILESYSTEMS (by default, all DMF-managed filesystems) writing the output to a file specified by SCAN_OUTPUT (by default /tmp/dmscanfs.output).

This file, if it exists, is used by run_daily_report.sh and dmstat(8) and may be of use to site-written scripts or programs. Although DMF does not require this file, the output from run_daily_report.sh and dmstat will be incomplete if it is unavailable.

If SCAN_FAST is set to no or off, dmscanfs will use its recursive option, which is much slower but results in pathnames being included in the output file. The default is yes.

By default, another output file is written to the bfid2path file in the daemon's *SPOOL_DIR* directory, optimized for use by dmstat; setting SCAN_FAST or SCAN_FOR_DMSTAT to no will suppress this.

- The run_daily_report.sh task reports on DCM MSPs and managed filesystems (if run_filesystem_scan.sh has been run recently) and on all LSs, and has superseded run_tape_report.sh and run_compact_tape_report.sh.
- 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 42).
- 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.
- 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.

• 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, spreading 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 \setminus
                                 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 51, 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 190.
- d. Manage the data from the filesystem dumps by configuring the parameters listed below.

The parameters specified in the task group include:

DUMP_DATABASE_COPY	Specifies the path to a directory where a snapshot of the DMF databases will be placed when do_predump.sh is run. The third-party backup application should be configured to backup this directory. If not specified, a snapshot will not be taken. (Third-party backup applications only).
DUMP_DEVICE	Specifies the name of the drive group in the

DUMP_FILE_SYSTEMS Specifies one or more filesystems to dump. If not

that the dump tasks will use.

specified, the tasks will dump all the DMF-managed user filesystems configured in the configuration file. Use this parameter if your site needs different dump policies (such as different dump times) for different

configuration file that defines how to mount the tapes

007-3681-019 55

filesystems or wishes to back up filesystems that are not managed by DMF. It is safest not to specify a value for this parameter and therefore dump all filesystems configured for management by DMF.

DUMP_FLUSH_DCM_FIRST

If set to YES, specifies that the dmmigrate command is run before the dumps are done to ensure that all non-dual-resident files in the DCM caches are migrated to tape. If DUMP_MIGRATE_FIRST is also enabled, that is processed first.

DUMP_INVENTORY_COPY

Specifies the pathnames of one or more directories into which are copied the XFS inventory files for the backed-up filesystems. If you specify multiple directories, spreading the directories among multiple disk devices minimizes the chance of losing all the copies of the inventory. The dump scripts choose the directory with the oldest inventory copy and copy the current one to it.

DUMP_MAX_FILESPACE

Specifies a maximum disk space used for files to be dumped, which may be larger or smaller than the length of the file. Regular files using more than this space are silently left out of the dump. This limit is not applied to offline or dual-state files. Note that this value applies to all filesystems being dumped except for the backup of the DMF databases. The value, which is in bytes, may have a suffix of k, m or g, meaning a multiplier of 1000, 1000000, or 1000000000 respectively. If this parameter is not provided, there is no limit.

DUMP_MIGRATE_FIRST

If set to YES, specifies that the dmmigrate command is run before the dumps are done to ensure that all migratable files in the DMF-managed user filesystems are migrated, thus reducing the number of tapes needed for the dump and making it run much faster.

DUMP_RETENTION

Specifies how long the backups of the filesystem will be kept before the tapes are reused. Valid values are a number followed by one of m[inutes], h[ours], d[ays] or w[eeks]. On some older versions of IRIX, a backup cannot be retired on the same day it was created, so short retention periods may not be very effective on those systems.

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

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

DUMP_VSNS_USED

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

DUMP_XFSDUMP_PARAMS

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

Configuring Device Objects

Normally, a drive group object is used to define the tape devices to be used by a dump_tasks task, with the DMF library server and the backup scripts sharing the same devices. However, if backups are to use different drives from those in use by DMF, they should be defined by a device object. The parameters you define are based on the mounting service you intend to use.

The following parameters are common to all device objects:

Option	Description
TYPE	device (required name for this type of object)
MOUNT_SERVICE	Specifies the mounting service to use. Supported values are openvault and tmf. This parameter is required; there is no default.

MOUNT_SERVICE_GROUP	Specifies the name by which the object'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 OpenVault drivegroup name that would be listed by the ov_drivegroup command. If this parameter is not specified, the device object's name is used.
OV_ACCESS_MODES	Specifies 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.
TMF_TMMNT_OPTIONS	Specifies command options that should be added to the tmmnt command when mounting a tape. DMF uses the -z option to tmmnt, so options controlling block size and label parameters are ignored. Use -g if the group name is different from the device object's name. Use -i to request compression.

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)
BUFFERED_IO_SIZE	Specifies the size of I/O requests when reading from this filesystem using buffered I/O. The legal range of values is 4096 through 16777216. The default is 262144.
DIRECT_IO_SIZE	Specifies the size of I/O requests when reading from this filesystem using direct I/O (see O_DIRECT in the open(2) man page for a description of direct I/O). The

legal range of values is 65536 through 18446744073709551615. The default value depends on the filesystem's configuration, but will not exceed the value of DIRECT_IO_MAXIMUM_SIZE defined in the base object.

MAX MANAGED REGIONS

Sets the maximum number of managed regions that DMF will assign to a file on a per-filesystem basis. You can set MAX_MANAGED_REGIONS to any number that is less than the actual number of regions that will fit in a filesystem attribute. For XFS filesystems, that number is 3275.

By default, DMF allows a DMF attribute to contain the maximum number of managed regions that will still allow the attribute to fit completely inside the inode, based on inode size and attr type. The default value for a filesystem that does not have a MAX_MANAGED_REGIONS parameter in its configuration file stanza is calculated at filesystem mount time. This value is chosen to ensure that the DMF attribute will fit inside the inode, assuming that no other attribute (such as an ACL or a parent inode pointer) is already occupying the inode's attribute space. Table 2-2 lists the default maximum file regions for XFS filesystems.



Caution: You should use MAX_MANAGED_REGIONS cautiously. If you set this parameter to a value that is larger than the default maximum (see Table 2-2 on page 60), the DMF attribute may not fit inside the inode. If there are many files with DMF attributes outside of the inode, filesystem scan times can increase greatly. To avoid this problem, SGI recommends that a file that has exceeded the maximum default file regions be made offline (that is, having a single region) as soon as possible after the online data has been accessed.

 Table 2-2 Default Maximum File Regions for XFS Filesystems

Size of inode	attr Type	Default Maximum Number of File Regions
256	1	(Not recommended)
256	2	2
512 or greater	1	8
512 or greater	2	11
	For information about setting mkfs(1M) or mount(1M) m	0 ,1
MESSAGE_LEVEL	Specifies the highest message level number that written to the automated space management log (autolog). It must be an integer in the range 0-higher the number, the more messages written to log file. The default is 2. For more information of message levels, see "General Message Log File Foon page 114.	
MIGRATION_LEVEL	Sets the level of migration s Valid values are:	service for the filesystem.
	• none (no migration)	
	• user (only user-initiated	d migration)
	• auto (automated space	management)
	The migration level actually the lesser of the MIGRATIO object and this value. The control of the migration of the migration of the migration level actually the lesser of the migration level actually the migr	N_LEVEL of the daemon
MIN_DIRECT_SIZE	Determines whether direct when reading from this file the open(2) man page for a If the number of bytes to be value specified, buffered I/I/O is used. The legal rang is always used) through 18 (direct I/O is never used).	system (see O_DIRECT in a description of direct I/O). The read is smaller than the O is used, otherwise direct the of values is 0 (direct I/O 446744073709551615

Note: Buffered I/O is always used when writing to a filesystem.

For real-time filesystems, this parameter is ignored.

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

Procedure 2-8 Configuring filesystem Objects

enddef

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 "Configuring DMF Policies" and Procedure 2-9, page 70, 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 "Configuring 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.

Configuring 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 or volume group selection
- Disk cache manager (DCM) use (see "DCM Policies" on page 75)

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 or volume group 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 121, 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, specifies that dmfsmon will first free dual-state and partial-state files before freeing files it must migrate. The default is off.
FREE_SPACE_DECREMENT	Specifies the percentage of filesystem space by which dmfsmon will decrement FREE_SPACE_MINIMUM if it cannot find enough files to migrate so that the value is reached. The decrement is applied until a value is found that dmfsmon can achieve. If space later frees up, the FREE_SPACE_MINIMUM is reset to its original value. Valid values are in the range 1 through the value of FREE_SPACE_TARGET. The default is 2.
FREE_SPACE_MINIMUM	Specifies the minimum percentage of free filesystem space that dmfsmon maintains. dmfsmon will begin to migrate files when the available free space for the filesystem falls below this percentage value. This parameter is required; there is no default.

FREE_SPACE_TARGET Specifies the percentage of free filesystem space that

dmfsmon will try to achieve if free space reaches or falls below FREE_SPACE_MINIMUM. This parameter is

required; there is no default.

MIGRATION_TARGET Specifies the percentage of filesystem capacity that DMF

maintains as a reserve of dual-state files whose online space can be freed if free space reaches or falls below FREE_SPACE_MINIMUM. dmfsmon tries to make sure that this percentage of the filesystem is migrated, migrating, or free after it runs to make space available.

This parameter is required; there is no default.

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 follows:
	constant + (multiplier * file_age_in_days)
	If DMF cannot locate values for this parameter, it uses a floating point constant of 1 and a floating point multiplier of 1.

SPACE_WEIGHT

This parameter accepts an optional when clause, which contains a conditional expression. This parameter also accepts an optional ranges clause, which specifies the ranges of a file for which the parameter applies.

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 follows:

constant + (multiplier * file_disk_space_in_bytes)

If DMF cannot locate values for this parameter, it uses a floating point constant of 0 and a floating point multiplier of 0.

For a partial-state (PAR) file, *file_disk_space_in_bytes* is the amount of space occupied by the file at the time of evaluation.

This parameter accepts an optional when clause, which contains a conditional expression. This parameter also accepts an optional ranges clause, which specifies the ranges of a file for which the parameter applies.

The parameter for MSP or volume group selection follows:

Parameter

SELECT_MSP or SELECT_VG

Description

Specifies the MSPs and volume groups to use for migrating a file. (It is not used for defining which MSP or volume group to use for recalls; for that, see the definitions of the LS_NAMES, MSP_NAMES, DRIVE_GROUPS, and VOLUME_GROUPS parameters.)

You can list as many 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; the names are equivalent. Volume groups, MSPs, or a mixture of both may be specified by either parameter.

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.

This parameter allows conditional expressions based on the value of a file tag. See "Customizing DMF" on page 26.

The root user on the DMF server can override the selection specified in this parameter through the use of the -V option on dmput, or with libdmfusr.so calls. If site-defined policies are in place, they may also override this parameter.

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	Specifies the number of days since last modification or last access of the file, whichever is more recent.
gid	Specifies the group ID or group name of the file.
sitefn	Invokes a site-defined policy function once for each file being considered, and is replaced by the return code of the function. This is only applicable to the AGE_WEIGHT, SPACE_WEIGHT, SELECT_MSP, and SELECT_VG parameters in a filesystem's policy stanza. For more information, see Appendix C, "Site-Defined Policy Subroutines and the sitelib.so Library" on page 255.

Specifies a site-determined number associated with a sitetag file by the dmtag(1) command, in the range 0 -4294967295. For example: sitetag = 27sitetag in (20-40, 5000, 4000000000) Specifies the logical size of the file, as shown by 1s -1. size softdeleted Specifies whether or not the file corresponding to a cached copy has been soft deleted; only applicable to the CACHE_AGE_WEIGHT, CACHE_SPACE_WEIGHT, and SELECT_LOWER_VG parameters in a DCM-mode MSP's policy stanza. Values are false and true. Specifies the number of bytes the file occupies on disk space (always a multiple of the blocksize, which may be larger or smaller than the length of the file). For a partial-state (PAR) file, the value used is the space that the file occupies on disk at the time of evaluation. uid Specifies the user ID or user name of the file.

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

Use the following operators to specify values:

= ! = > < < > < > = < = in

The following are examples of valid expressions:

The AGE_WEIGHT and SPACE_WEIGHT parameters accept an optional ranges clause to restrict the ranges of a file for which a parameter applies. The clause has the following form, where *byteranges* is one or more byte ranges:

```
ranges byteranges
```

Each byte range consists of a set of numbers that indicate byte positions. (You can also use BOF or bof to indicate the first byte in the file and EOF or eof to indicate the last byte in the file.) Each byte range is separated by a comma and can have one of the following forms:

• A specification of two byte positions, where *first* specifies the first byte in the range and *last* specifies the last byte in the range:

```
first:last
```

If unsigned, *first* and *last* count from the beginning of the file; if preceded by a minus sign (-), they count backwards from the end of the file.

The first byte in the file is byte 0 or BOF and the last byte is -0 or EOF. Therefore, BOF: EOF and 0:-0 both define a range covering the entire file.

For example:

- ranges 0:4095 specifies the first 4096 bytes of the file
- ranges -4095:EOF specifies the last 4096 bytes of the file
- A specification of the size of the range, starting at a given point, where *first* is a byte position as above and *size* is the number of bytes in the range, starting at *first*:

first+size

For example, the following indicates bytes 20 through 29:

```
ranges 20+10
```

If *size* is preceded by a minus sign, it specifies a range of *size* bytes ending at *first*. For example, the following indicates bytes 11 through 20:

```
ranges 20+-10
```

 A specification of the size of the range only (without a colon or plus symbol), assumed to start at the end of file (when preceded by a minus sign) or beginning of file:

```
-size
size
```

For example, the following specifies the last 20 bytes in the file:

```
ranges -20
```

The *first*, *last*, or *size* values can be of the following forms:

- A hexadecimal number: 0xn
- A based number: base#n
- A decimal number with an optional trailing scaling character. The decimal number may include a decimal point (.) and exponent. The trailing scaling character may be one of the following (all of which are powers of 1000, not 1024):

```
k or K for 1 thousand
m or M for 1 million
g or G for 1 billion
t or T for 1 trillion
p or P for 1 quadrillion
```

Note: DMF may round byte ranges and join nearby ranges if necessary. If a range is given a negative weight, rounding may cause additional bytes to be ineligible for automatic space management.

Do not use a ranges clause when partial-state files are disabled in DMF. Specifying many ranges for a file is discouraged, as it can cause the time and memory used by automatic space management to grow. DMF has an upper limit on the number of

007–3681–019

regions that can exist within a file; this can sometimes cause a range to be given an effective lower weight than what was specified in the configuration file. This might happen if the file is already partial-state and the range with largest weight cannot be made offline (OFL) because that would create too many regions. If the file has too many regions to make the range offline, but it could be made offline at the same time as a range with lower weight, it will be given the lower weight. If more than one range in the middle of a file is not a candidate for automatic space migration, the limit on the number of regions may make it impossible to automatically free other regions of the file.

Configuring Policies

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

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 or partial-state files before it migrates and frees regular files. The default is off.

- 4. Configure the age and size weighting factors associated with a file when it is evaluated for migration as follows:
 - a. The syntax of the 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 and no ranges clause is present, DMF determines whether the file matches the criteria in the clause. If no when clause is present, a match is assumed. If the file matches the criteria, the file weight is calculated from the parameter values. If they do not match, the next instance of that parameter is examined.

An AGE_WEIGHT of 1 1.0 is used if no AGE_WEIGHT applies for a file.

Example 2-1 policy Object for Automated Space Management

```
define fs_space
                                 policy
        TYPE
                                 50
       MIGRATION_TARGET
        FREE_SPACE_TARGET
                                 10
        FREE_SPACE_MINIMUM
        FREE_DUALSTATE_FIRST
                                 off
        AGE_WEIGHT 0
                       0.00
                                 when age < 10
        AGE_WEIGHT 1
                       0.01
                                 when age < 30
                                when age < 120
        AGE_WEIGHT 10 0.05
        AGE_WEIGHT 50
                      0.1
        SPACE_WEIGHT 0 0
enddef
```

In Example 2-1, 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.

007–3681–019

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 Example 2-1, 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 \leq 1
```

The following parameter specifies that small files are never automatically migrated:

```
SPACE_WEIGHT -1 0 when space <= 4k
```

d. If partial-state files are enabled on your machine (meaning that you have the PARTIAL_STATE_FILES configuration file parameter set to on and have the appropriate kernel installed, according to the information in the DMF.News file), you can use the ranges clause to select ranges of a file.

DMF checks each AGE_WEIGHT parameter in turn, in the order they occur in the configuration file. As described in step 4a above, DMF checks the when clause, if present, to see if the file matches the criteria. If the file matches and a ranges clause is present, DMF determines if that range has already been weighted. If it has not been weighted, the specified range is given the weight calculated from the parameter values. DMF examines the next instance of the parameter until all ranges in the file have been assigned a weight.

Example 2-2 policy Object for Automated Space Management using Ranges

```
define fs2_space
TYPE policy
MIGRATION_TARGET 50
FREE_SPACE_TARGET 10
FREE_SPACE_MINIMUM 5
FREE_DUALSTATE_FIRST off
```

```
AGE_WEIGHT -1. 0.00 ranges 0:4095 when uid=624
AGE_WEIGHT -1 0 ranges 0:4095,-4095:EOF when uid=321
AGE_WEIGHT 1 0.01 when age < 30
AGE_WEIGHT 10 0.05 when age < 120
AGE_WEIGHT 50 0.1

SPACE_WEIGHT 0 0
```

enddef

In Example 2-2, if a file is owned by UID 624 and is 1004096 byes long, the first 4096 bytes are given an AGE_WEIGHT of -1. The remaining 1000000 bytes are given an AGE_WEIGHT based on the age of the file. Therefore, the first 4096 bytes of the file would not be eligible for being put offline by automatic space management. If a file is owned by UID 321, the first and last 4096 bytes of it are not eligible for being put offline by automatic space migration. If a file is owned by UID 956, the policy in Example 2-2 would give the entire file an AGE_WEIGHT based on its age. SPACE_WEIGHT parameters are evaluated similarly.

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

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 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:

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

The order of the SELECT statements is important. The first SELECT statement that applies to the file is honored. For example, if the statements above were reversed, a 50m file would be migrated to cart2, because the check for greater than or equal to (>=) 65536 would be done first, and it would be true.

b. If you want certain files to be copied to more than one MSP 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_VG none when space <= 4096

(You could use either SELECT_VG or SELECT_MSP in these examples.)

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 DMF-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

007–3681–019

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 dual-residence, which means that a
 cache-resident copy of a migrated file has already been copied to tape, and can
 therefore be released quickly in order to prevent the cache filling, without any
 need to first copy it to tape. It is analogous to a dual-state file 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. (A ranges clause is not valid with the CACHE_AGE_WEIGHT or CACHE_SPACE_WEIGHT parameters.)
- SELECT_LOWER_VG defines which volume groups should maintain tape-based copies of files in the cache, and under what conditions that would define dual-residence.

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 an LS

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 2-5 on page 116.

LS 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	Specifies the 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.

Specifies the amount of disk space (in bytes) that dmatls can use when merging chunks from sparse tapes. During merging, small chunks from sparse tapes are cached on disk before being written to a tape. The default is 0, which causes all files to be merged via sockets.

Specifies the binary file to execute to initiate the LS. This value must be ${\tt dmatls}$.

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 drive group object's VOLUME_GROUPS parameter.

Note: Do not change this parameter while DMF is running.

Specifies the largest chunk (in bytes) that will be merged using the merge disk cache. Larger files are transferred directly via a socket from the read child to the write child. The default is 25% of the CACHE_SPACE value. Valid values are 0 through the value of CACHE_SPACE.

Specifies the highest message level number that will be written to the LS log, which includes messages from the LS's components. It must be an integer in the range

CACHE_SPACE

COMMAND

DRIVE_GROUPS

MAX_CACHE_FILE

MESSAGE_LEVEL

007-3681-019

77

	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 36.
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	Specifies the 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	
	HP ULTRIUM 4	2097152	
	HP ULTRIUM 3	524288	
	HP ULTRIUM 2	262144	
	IBM 03590B1A	16384	
	IBM 03590E1A	32768	
	IBM 03590H1A	16384	
	IBM 03592E05	65536	
	IBM ULTRIUM-TD1	131072	
	IBM ULT3580-TD1	131072	
	IBM ULTRIUM-TD2	262144	
	IBM ULT3580-TD2	262144	
	IBM ULTRIUM-TD3	262144	
	IBM ULT3580-TD3	262144	
	IBM ULTRIUM-TD4	2097152	

IBM ULT3580-TD4	2097152
QUANTUM SDLT600	131072
QUANTUM SDLT320	131072
QUANTUM SuperDLT1	131072
SEAGATE ULTRIUM	262144
SONY SDX-700C	131072
SONY SDZ-100	131072
SONY SDZ-130	262144
SONY SDZ-200	2097152
SONY SDZ-230	2097152
STK 9840	126976
STK T9840B	126976
STK T9840C	262144
STK T9940A	262144
STK T9940B	262144
STK T10000A	524288
(Other devices)	65536

DRIVE_MAXIMUM

Specifies the maximum number of drives within this drive group that the LS is allowed to attempt to use simultaneously. This can be more or less than the number of drives the LS can physically detect. The maximum is 100; the default is 100 for drive groups. If a negative value is specified for DRIVE_MAXIMUM, the drive group uses the sum of the number of available drives and DRIVE_MAXIMUM.

DRIVE_SCHEDULER

Names the resource scheduler that the drive group should run for the scheduling of tape drives. They are configured as objects of type resourcescheduler. The default is a resource scheduler of default type and parameters. For the defaults, see "Resource Scheduler Objects" on page 87.

DRIVES_TO_DOWN

Specifies an integer value that controls the number of "bad" drives the drive group is allowed to try to configure down. When more than this number are down, whether due to the drive group or to external influences such as the system administrator, the drive group does not attempt to disable any more. The default of 0 prevents the drive group from disabling any.

007–3681–019

LABEL TYPE

Specifies the label type used when writing tapes from the beginning. Possible values are:

- nl (no label)
- sl (standard label, for IBM tapes)
- al (ANSI label)

The default is al.

MAX_MS_RESTARTS

Specifies the maximum number of times DMF can attempt to restart the mounting service (TMF or OpenVault) without requiring administrator intervention. The default and recommended values are 1 for TMF and 0 for OpenVault.

MOUNT_SERVICE

Specifies the mounting service 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 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.

READ_IDLE_DELAY

Specifies the number of seconds an idle tape LS read child (dmatrc) can wait before being told to exit. If other DMF requests are waiting for a tape drive, the read child may be told to exit before READ_IDLE_DELAY seconds have passed. The default

is 5 seconds.

REINSTATE_DRIVE_DELAY

Specifies the number of minutes after which a drive that was configured down by the drive group will be automatically reinstated and made available for use again. A value of 0 means it should be left disabled indefinitely. The default is 1440 (one day).

REINSTATE VOLUME DELAY Specifies the number of minutes after which a volume that had its HLOCK flag set by DMF will be automatically reinstated and made available for use again. A value of 0 means they should be left disabled indefinitely. The default is 1440 (one day).

RUN TASK

See "Automated Maintenance Tasks" on page 36.

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 drive group parameters instead. There is no need for a -g option here. If it is provided, it must match the value of the MOUNT_SERVICE_GROUP parameter. To request compression, use -i. Options that are ignored are -a, -b, -c, -D, -f, -F, -l, -L, -n,

-o, -0, -p, -P, -q, -R, -t, -T, -U, -v, -V, -w, -x , and

-X.

82 007-3681-019 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 drive group object's VOLUME_GROUPS parameter.

Note: Do not change this parameter while DMF is running.

WRITE_CHECKSUM

Specifies that tape block should be checksummed before writing. If a tape block has a checksum, it is verified when read. The default is on.

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:

007-3681-019

Option

Description

TYPE

ALLOCATION_GROUP

volumegroup (required name for this type of object)

Specifies the 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 the ALLOCATION_GROUP option will not use an allocation group.

ALLOCATION_MAXIMUM

Specifies the 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

Specifies the 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

Specifies the 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

Specifies the 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

MERGE CUTOFF

MIN_VOLUMES

PUTS_TIME

READ_TIME

volume group. The maximum value is the value of DRIVE_MAXIMUM for the volume group's owning drive group. The minimum value is 1. The default is the same as the value that the volume group uses for DRIVE_MAXIMUM; if the value specified in the configuration file exceeds this default, the default is used.

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. The minimum value that you can specify is 2.

Specifies the 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.

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.

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 36.
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	Specifies the number of minutes after which the volume group will flush files to tape. The default is 120 minutes.
ZONE_SIZE	Specifies about how much data the write child should put in a zone. The write child adds files and chunks to a zone until the data written equals or exceeds this value, at which time it writes a tape mark and updates the data base. Smaller values can cause poorer write performance because a tape mark is written at the end of each zone. The volume group also uses zone size to determine when to start write children, and the number of write children to start. The default is 50000000 (50m) bytes. For increased write performance, choose a ZONE_SIZE such that the average time to write a tape mark for the drive type is a small percentage (for example, 1 percent) of the time to write a zone at the drive's native rate.

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	Specifies 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

Specifies 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	Specifies 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	dq1	
		TYPE	drivegroup
		VOLUME_GROUPS	vg_prim vg_sec
		MOUNT_SERVICE	openvault

MOUNT_SERVICE_GROUP drives OV_INTERCHANGE_MODES compression DRIVE_SCHEDULER DRIVES_TO_DOWN 2 REINSTATE DRIVE DELAY enddef define rs TYPE resourcescheduler 10 WEIGHT vg_prim WEIGHT 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 2-11, page 90 explain pertinent information for configuring each of the LS objects in the previous example.

Procedure 2-11 Configuring an LS 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 145.
- 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 the LS" on page 97.
- 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 LS Drive Groups

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

Procedure 2-12 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 then activate both a privileged and an 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 LS might result in tape mount failures.

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 hostname 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. 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.

- 7. Make the appropriate cartridges accessible to the allocation groups, volume groups, or filesystem backup scripts 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 -s -A dmf
```

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

```
dmov_makecarts -g cartgroup -t carttype lsname
dmov_makecarts -g cartgroup -t carttype -v vg1, vg2 lsname
dmov_makecarts -g cartgroup -t carttype taskgroupname
```

You can replace any of the references to a 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. Tapes will be added to the file controlling the run_full_dump.sh and

007–3681–019 95

run_partial_dump.sh scripts by specifying the name of the task group that refers to them.

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 -1 library -g cartgroup -t carttype vgname dmov_loadtapes -1 library -g cartgroup -t carttype agname dmov_loadtapes -1 library -g cartgroup -t carttype taskgroupname
```

This command will invoke a vi(1) session. In the vi(1) session, delete any cartridges that you do **not** want added to the database. Tapes will be added to the file controlling the run_full_dump.sh and run_partial_dump.sh scripts by specifying the name of the task group which refers to them.

- 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 LS Drive Groups

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

```
dmvoladm -1 Isname -c 'create vsn001-vsn010 vg vgname' dmvoladm -1 Isname -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 the LS

You can configure parameters for how the 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 volume 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 50.

Table 2-1 on page 37, 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. This example assumes the LS using this task has only one volume group. For information about volume merging when an LS has multiple volume groups, see Procedure 2-13, step 3 on page 98.

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.

007–3681–019 97

```
#DATA_LIMIT 5g
#
RUN_TASK $ADMINDIR/run_merge_stop.sh at 5:00
```

Procedure 2-13 Configuring the msp_tasks Object

- 1. Define the object to have the same name that you provided for the TASK_GROUPS parameter of the LS 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 and run_compact_tape_report.sh tasks generate a report on the tapes in the LS tape pool and on volume group activity. They are now superseded by the daemon task run daily report.sh.

- 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.

For LSs, you can configure volume merging as part of the LS object's TASK_GROUPS parameter or as part of a RUN_TASK parameter in the volume group object. If it is configured as part of the LS's TASK_GROUPS parameter, volumes from any of the volume groups in that LS may be marked as sparse. This can lead to drive scheduling and cache usage conflicts. To avoid this problem, configure volume merging as part of the volume group object and ensure there is no overlap in the times that the various merge tasks run.

As this might become cumbersome when there are large numbers of 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 requests first, minimizing interference with the production workload. To use this script, perform the following steps:

- 1.) Define a task group, which is referred to by the drive group object (not the volume group or LS object).
- 2.) Specify a RUN_TASK parameter for run_merge_mgr.sh in the task group 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.

c. Use the run_merge_stop.sh task to shut down volume merging at a time you specify by using a *time_expression*. This task is an alternative to using the VOLUME_LIMIT and DATA_LIMIT parameters to stop merging at specified points. In the example, the limit parameters are commented out because run_merge_stop.sh is used to control volume merging.

LS Database Records

After you have added the LS information to the configuration file, use the dmvoladm(8) command with the -m option to create any missing directories with the proper labels and to create the volume (VOL) and catalog (CAT) records in the LS database.

You can follow the steps in Procedure 2-14, for all of the LSs you have defined.



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

Procedure 2-14 Creating LS Database Records

The following procedure is shown as an example that assumes you have an LS called 1s1. 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 an FTP MSP

To enable a file transfer protocol (FTP) MSP, include a name for it on the MSP_NAMES or LS_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 2-16, page 109):

Parameter	Description
TYPE	msp (required name for this type of object)
CHILD_MAXIMUM	Specifies the maximum number of child processes the MSP is allowed to fork. The default is 4; the maximum is 100.
COMMAND	Specifies the binary file to execute in order to initiate this MSP. For the FTP MSP, this value must be dmftpmsp.
FTP_ACCOUNT	Specifies the account ID to use when migrating files to the remote system.

FTP_COMMAND Specifies additional commands to send to the remote

system. There may be more than one instance of this

parameter.

FTP_DIRECTORY Specifies the directory to use on the remote system.

FTP_HOST Specifies the Internet hostname of the remote machine

on which files are to be stored.

FTP_PASSWORD Specifies the file containing the password to use when

migrating files to the remote system. This file must be owned by root and be only accessible by root.

FTP_PORT Specifies the port number of the FTP server on the

remote system. The default value is the value configured for ftp in the services file.

FTP_USER Specifies the user name to use when migrating files to

the remote system.

GUARANTEED_DELETES Specifies the number of child processes that are

guaranteed to be available for processing delete requests. If CHILD_MAXIMUM is nonzero, its value must

be greater than the sum of GUARANTEED_DELETES and

GUARANTEED GETS. The default is 1.

GUARANTEED_GETS Specifies the number of child processes that are

guaranteed to be available for processing dmget(1) requests. If CHILD_MAXIMUM is nonzero, its value must be greater than the sum of GUARANTEED_DELETES and

GUARANTEED_GETS. The default is 1.

IMPORT_DELETE Specifies if the MSP should honor hard-delete requests

from the DMF daemon. 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 114.

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 2-3.

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

007-3681-019

	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 33.
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.
WRITE_CHECKSUM	Specifies that the DMF MSP's copy of the file should be checksummed before writing. If the file has been checksummed, it is verified when read. The default is on.

The MSP checks the DMF configuration file just before it starts child processes. If the DMF configuration file changed, it is reread.

If CHILD_MAXIMUM is nonzero, its value must be greater than the sum of GUARANTEED_DELETES and GUARANTEED_GETS.

The parameters COMMAND, FTP_HOST, FTP_USER, FTP_PASSWORD, and FTP_DIRECTORY must be present.

The MVS_UNIT parameter affects only IBM machines; they are further described in the dmf.conf(5) man page.

Note: The MSP will not operate if the FTP_PASSWORD file is readable by anyone other than root.

Table 2-3 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.

String	Evaluates To
%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 2-15 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 a Disk MSP

To enable a disk MSP, include a name for it on the MSP_NAMES or LS_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	Specifies the maximum number of child processes the MSP is allowed to fork. The default is 4; the maximum is 100.
COMMAND	Specifies the binary file to execute in order to initiate this MSP. For the disk MSP, this value must be dmdskmsp.
DSK_BUFSIZE	Specifies the transfer size in bytes used when reading from and writing to files within the disk MSP's STORE_DIRECTORY. The value must be in the range 4096 through 16 million. The default is 131072 when writing, and 1000000 when reading.
GUARANTEED_DELETES	Specifies the number of child processes that are guaranteed to be available for processing delete requests. The default is 1.
GUARANTEED_GETS	Specifies the number of child processes that are guaranteed to be available for processing dmget(1) requests. The default is 1.
IMPORT_DELETE	Applies only if IMPORT_ONLY is set to on. Set IMPORT_DELETE to on if you wish files to be deleted in STORE_DIRECTORY when hard deletes are processed.
	Does not apply to DCM-mode.
IMPORT_ONLY	Specifies the MSP is used for importing only. Set this parameter on when the data is stored as a bit-for-bit copy of the file and needs to be available to DMF as part of a conversion. The MSP will not accept dmput(1) requests when this parameter is enabled. By default, the MSP will ignore hard delete requests when this parameter is enabled.
	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

MIGRATION_LEVEL

NAME_FORMAT

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 114.

Specifies the level of migration service for the DCM, as in filesystem objects. Valid values are none (no flushing to a lower volume group), user (only requests from dmmigrate(8) or a manually invoked dmdskfree(1)), or auto (automated space management). The default is auto. (Applicable only to a DCM-mode MSP.)

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 2-3 on page 104.

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 33.

Specifies that the DMF MSP's copy of the file should be

checksummed before writing. If the file has been checksummed, it is verified when read. The default is

POLICIES	For DCM disk MSP use only. See "Setting Up a Disk MSP in DCM Mode" on page 110.
PRIORITY_PERIOD	Specifies the number of minutes after which a migrating file gets special treatment. Normally, if there is insufficient room in the STORE_DIRECTORY for a file, the DCM MSP will attempt to make room, while continuing to store files that will fit. If a file has not been stored into the STORE_DIRECTORY within PRIORITY_PERIOD, however, the DCM MSP will stop trying to store other files until either sufficient room has been made or it has determined that room cannot be made. This behavior may change in the future. The default value is 2 hours; the minimum is 1 minute, and the maximum value is 2000000. (Applicable only to DCM-mode disk MSPs.)
STORE_DIRECTORY	Specifies the directory used to hold 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.

The following example describes setting up a disk MSP:

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

Procedure 2-16 Configuring the dsk Object

WRITE_CHECKSUM

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.

007-3681-019 109

- 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 a Disk MSP in DCM Mode".)

Setting Up a Disk MSP 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 31 for instructions.
- The configuration stanza must contain at least one POLICIES parameter and the configuration stanza for that parameter must contain a SELECT_LOWER_VG parameter.
- There must also be a task group that runs the run_dcm_admin script during off-peak hours to perform routine maintenance for the MSP.
- In DCM mode, the BUFFERED_IO_SIZE, DIRECT_IO_SIZE, and MIN_DIRECT_SIZE have the following definitions:

BUFFERED_IO_SIZE

Specifies the size of I/O requests for buffered I/O when migrating files downward in the hierarchy from the STORE_DIRECTORY of this disk MSP running in DCM mode. The legal range of values is 4096 through 16777216. The default is 262144.

DIRECT_IO_SIZE

Specifies the size of I/O requests for direct I/O when migrating files downward in the hierarchy from the STORE_DIRECTORY of this disk MSP running in DCM mode (see O_DIRECT in the open(2) man page for a description of direct I/O). The legal range of values is 65536 through 18446744073709551615. The default depends on the filesystem, but will not exceed the value of DIRECT_IO_MAXIMUM_SIZE defined in the base object.

MIN_DIRECT_SIZE

Determines whether direct or buffered I/O is used when migrating files downward in the hierarchy from the STORE_DIRECTORY of this disk MSP running in DCM mode (see O_DIRECT in the open(2) man page for a description of direct I/O). If the number of bytes to be read is smaller than the value specified, buffered I/O is used, otherwise direct I/O is used. The legal range of values is 0 (direct I/O is always used) through 18446744073709551615 (direct I/O is never used). The default is 0.

Note: For real-time filesystems, this parameter is ignored.

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 33.

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
                                  dcm_msp copy2 when space > 4096 # [See note 3]
        SELECT_MSP
                                                                    # [See note 2]
enddef
define
        dcm_msp
        TYPE
                                  msp
        COMMAND
                                  dmdskmsp
                                                                    # [See note 4]
        STORE_DIRECTORY
                                  /dcm_cache
                                  10
                                                                    # [See note 5]
        CHILD_MAXIMUM
        POLICIES
                                  dcm_policy
                                  dcm_tasks
        TASK_GROUPS
enddef
define
        dcm_policy
                                                                    # [See note 6]
        TYPE
                                  policy
        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
                                  vgl 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_VG volume groups.

- 4. A **dedicated** DMAPI-mounted filesystem
- 5. Any other parameters applicable to a disk MSP may also be used, with the exception of IMPORT_ONLY and IMPORT_DELETE.
- 6. Several parameters in DCM policies have functions that are analogous to those in standard disk MSP policies; see "DCM Policies" on page 75.

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 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, but this is not the recommended method.

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:

- "Automated Space Management Log File" on page 125 for information about dmfsmon and autolog. yyyymmdd
- "Daemon Logs and Journals" on page 136 for information about dmfdaemon and dmdlog. yyyymmdd
- "dmlockmgr Communication and Log Files" on page 139 for information about dmlockmgr and dmlocklog. yyyymmdd
- "LS Logs" on page 150 and "FTP MSP Activity Log" on page 177 for information about dmatls, dmdskmsp, dmftpmsp, and msplog.yyyymmdd
- Chapter 7, "DMF Maintenance and Recovery" on page 189, 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 2-4. The table also lists the corresponding message levels in the configuration file.

Table 2-4 DMF Log File Message Types

Field	Message type	Message level
-E	Error	0
-0	Ordinary	0
-I	Informative	1
-V	Verbose	2
-1	Debug level 1	3
-2	Debug level 2	4
-3	Debug level 3	5
-4	Debug level 4	6

Parameter Table

Table 2-5 on page 116, 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

TG: Task group

VG: Volume group

 Table 2-5 Parameters for the DMF Configuration File

Parameter	BS	DM	DV	DG	DP	DC	FS	FP	LS	РО	RS	RW	TG	VG
ADMIN_EMAIL	Х													
AGE_WEIGHT										Χ				
ALGORITHM											Χ			
ALLOCATION_GROUP														Х
ALLOCATION_MAXIMUM														Χ
ALLOCATION_MINIMUM														Х
BLOCK_SIZE				Χ										
BUFFERED_IO_SIZE						Χ	Χ							
CACHE_AGE_WEIGHT										Χ				
CACHE_DIR									Χ					
CACHE_SPACE									Χ					
CACHE_SPACE_WEIGHT										Χ				
CHILD_MAXIMUM					Χ	Χ		Χ						
COMMAND					Χ	Χ		Χ	Х					
DATABASE_COPIES													Χ	
DATA_LIMIT													Χ	
DIRECT_IO_MAXIMUM_SIZE	Х													
DIRECT_IO_SIZE						Χ	Χ							
DRIVES_TO_DOWN				Χ										
DRIVE_GROUPS									Χ					
DRIVE_MAXIMUM				Χ										Х
DRIVE_SCHEDULER				Χ										
DUALRESIDENCE_TARGET										Χ				
DUMP_DATABASE_COPY													Χ	
DUMP_DEVICE													Х	

Parameter	BS	DM	DV	DG	DP	DC	FS	FP	LS	РО	RS	RW	TG	VG
DUMP_FILE_SYSTEMS													Х	
DUMP_FLUSH_DCM_FIRST													Χ	
DUMP_INVENTORY_COPY													Χ	
DUMP_MAX_FILESPACE													Χ	
DUMP_MIGRATE_FIRST													Χ	
DUMP_RETENTION													Χ	
DUMP_TAPES													Χ	
DUMP_VSNS_USED													Χ	
DUMP_XFSDUMP_PARAMS													Χ	
EXPORT_QUEUE		X												
FREE_DUALSTATE_FIRST										Х				
FREE_SPACE_DECREMENT										Х				
FREE_SPACE_MINIMUM										Х				
FREE_SPACE_TARGET										Х				
FTP_ACCOUNT								Χ						
FTP_COMMAND								Χ						
FTP_DIRECTORY								Χ						
FTP_HOST								Χ						
FTP_PASSWORD								Χ						
FTP_PORT								Χ						
FTP_USER								Χ						
GUARANTEED_DELETES					Χ	Χ		Χ						
GUARANTEED_GETS					Χ	Χ		Χ						
HFREE_TIME														X
HOME_DIR	Х													
HTML_REFRESH												Х		

Parameter	BS	DM	DV	DG	DP	DC	FS	FP	LS	РО	RS	RW	TG	VG
IMPORT_DELETE					Х			Х						
IMPORT_ONLY					Χ			Χ						
JOURNAL_DIR	Х													
JOURNAL_RETENTION													Х	
JOURNAL_SIZE	Х													
LABEL_TYPE				Χ										
LICENSE_FILE	Х													
LOG_RETENTION													Х	
LS_NAMES		Χ												
MAX_CACHE_FILE									Χ					
MAX_CHUNK_SIZE														Χ
MAX_MS_RESTARTS				Χ										
MAX_PUT_CHILDREN														Χ
MAX_VIRTUAL_MEMORY		Χ												
MERGE_CUTOFF														Χ
MESSAGE_LEVEL		Χ			Χ	Χ	Χ	Χ	X					
MIGRATION_LEVEL		X				Χ	Χ							
MIGRATION_TARGET										Χ				
MIN_DIRECT_SIZE						Χ	Χ							
MIN_VOLUMES														Χ
MODULE_PATH											Χ			
MOUNT_SERVICE			Χ	Χ										
MOUNT_SERVICE_GROUP				Χ										
MOUNT_TIMEOUT				Χ										
MOVE_FS		Χ												
MSG_DELAY				Χ										

Parameter	BS	DM	DV	DG	DP	DC	FS	FP	LS	РО	RS	RW	TG	VG
MSP_NAMES		Х												
MVS_UNIT								Χ						
NAME_FORMAT					Χ	Χ		Χ						
OV_ACCESS_MODES			Χ	Χ										
OV_INTERCHANGE_MODES			Χ	Χ										
OV_KEY_FILE	Х													
OV_SERVER	Х													
PENALTY											Χ			
POLICIES						Χ	Χ							
POSITIONING				Χ										
POSITION_RETRY				Χ										
PUTS_TIME														Χ
READ_IDLE_DELAY				Χ										
READ_TIME														Χ
REINSTATE_DRIVE_DELAY				Χ										
REINSTATE_VOLUME_DELAY				Χ										
RUN_TASK													Х	*
SELECT_LOWER_VG										Χ				
SELECT_MSP										Χ				
SELECT_VG										X				
SITE_SCRIPT										Χ				
SPACE_WEIGHT										Χ				
SPOOL_DIR	Х													
STORE_DIRECTORY					Χ	Χ								
TAPE_TYPE														
TASK_GROUPS		Χ		Χ	Х	Χ	Х	X	X					Χ

Parameter	BS	DM	DV	DG	DP	DC	FS	FP	LS	РО	RS	RW	TG	VG
									1					
THRESHOLD													X	
TIMEOUT_FLUSH														Χ
TMF_TMMNT_OPTIONS			Χ	Χ										
TMP_DIR	X													
VERIFY_POSITION				Χ										
VOLUME_GROUPS				Χ										
VOLUME_LIMIT													Χ	
WATCHER									Χ					
WEIGHT											Χ			
WRITE_CHECKSUM				Χ										
ZONE_SIZE														X

 $^{^{\}ast}$ 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 with automated space management enabled (auto). When the free space in one of the filesystems falls below the free-space minimum, dmfsmon invokes dmfsfree(8). The dmfsfree command attempts to bring the free space and migrated space of a filesystem into compliance with configured values. dmfsfree may also be invoked directly by system administrators.

When the free space in one of the filesystems falls below its minimum, the dmfsfree command performs the following steps:

- Scans the filesystem for files that can be migrated and freed or ranges of files that
 can be freed. Each of these candidates is assigned a weight. This information is
 used to create a list, called a *candidate list*, that contains an entry for each file or
 range and is ordered by weight (largest to smallest).
- Selects enough candidates to bring the free space back up to the desired level. Files or ranges of files are selected in order from largest weight to smallest.
- Selects enough non-migrated files from the candidate list to achieve the migration target, which is the percentage of filesystem space you want to have as free space and space occupied by migrated but online files. Files are selected from the candidate list in order from largest weight to smallest.

The dmf smon daemon should be running whenever DMF is active. You control automated space management by setting the filesystem and policy configuration parameters in the DMF configuration file. The configuration parameters specify targets for migration and free space as well as one or more policies for weighting. Only filesystems configured as MIGRATION_LEVEL auto in the configuration file are included in the space-management process. "Configuring DMF Policies" on page 62, 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 122
- "Selection of Migration Candidates" on page 123
- "Space Management and the Disk Cache Manager" on page 125

"Automated Space Management Log File" on page 125

Generating the Candidate List

The first step in the migration process occurs when dmfsmon determines it is time to invoke dmfsfree, which scans the filesystem and generates the candidate list. During candidate list generation, the inode of each online file in the specified filesystem is audited and a weight is computed for it.

A filesystem is associated with a weighting policy in the DMF configuration file. The applicable weighting policy determines a file's total weight, or, if a ranges clause is specified in the configuration file, the range's total weight. Total file or range weight is the sum of the AGE_WEIGHT and SPACE_WEIGHT parameters. Defaults are provided for these parameters, and you can configure either to make a change. You do not need to configure a weighting policy if the defaults are acceptable, but you should be aware that the default selects files based on age and not on size. If you want to configure a policy based on size that ignores file age, you should overwrite the default for AGE_WEIGHT.

The default weighting policy bases the weight of the file on the time that has passed since the file was last accessed or modified. Usually, the more recent a file's access, the more likely it is to be accessed again.

The candidate list is ordered by total file or range weight (largest to smallest). You can prevent a file from being automatically migrated by making sure that no ranges within the file have a positive weight value. You can configure the weighting parameters to have a negative value to ensure that certain files or ranges are never automatically freed.

Note: If you use negative weights to exclude files or ranges from migration, you must ensure that a filesystem does not fill with files or ranges that are never selected for automatic migration.

You can use the dmscanfs(8) command to print file information to standard output (stdout).

Selection of Migration Candidates

The dmfsfree(8) utility processes each ordered candidate list sequentially, seeking candidates to migrate and possibly free. The extent of the selection process is governed by values defined for the filesystem in the DMF configuration file as described in "Configuring DMF Policies" on page 62.

The most essential parameters are as follows:

- FREE_SPACE_MINIMUM specifies the minimum percentage of filesystem space that must be free. When this value is reached, dmfsmon will take action to migrate and free enough files or ranges to bring the filesystem into compliance. For example, setting this parameter to 10 indicates that when less than 10% of the filesystem space is free, dmfsmon will migrate and free files to achieve the percentage of free space specified by FREE_SPACE_TARGET. For the information on how this parameter is used when automated space management is not configured, see the dmf.conf(5) man page.
- FREE_SPACE_TARGET specifies the percentage of free filesystem space dmfsmon will try to achieve if free space falls below FREE_SPACE_MINIMUM. For example, if this parameter is set to 15 and FREE_SPACE_MINIMUM is set to 10, dmfsmon takes action when the filesystem is less than 10% free and migrates and frees files until 15% of the filesystem is available.
- MIGRATION_TARGET specifies the percentage of filesystem capacity that is
 maintained as a reserve of space that is free or occupied by dual-state files. DMF
 attempts to maintain this reserve in the event that the filesystem free space reaches
 or falls below FREE_SPACE_MINIMUM.

When dmfsmon detects that the free space on a filesystem has fallen below the level you have set as FREE_SPACE_MINIMUM, it invokes dmfsfree to select a sufficient number of candidates to meet the FREE_SPACE_TARGET. The dmfsfree utility ensures that these files are 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.

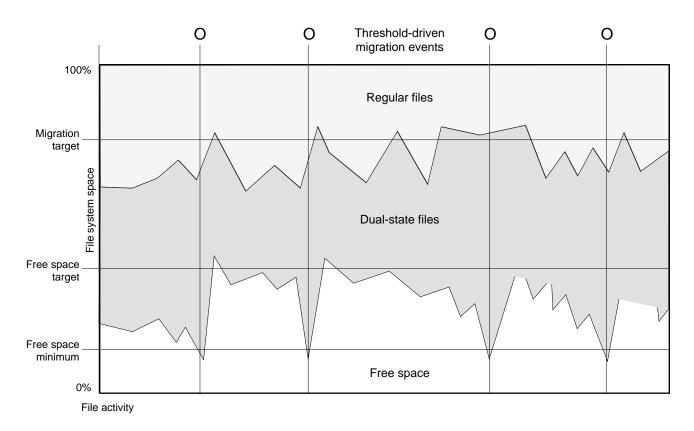


Figure 3-1 Relationship of Automated Space Management Targets

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 DMF-managed filesystems, with the following differences:

- The disk MSP (dmdskmsp) monitors the cache, instead of a separate monitoring program such as dmfsmon
- The dmdskfree utility controls the movement of cache files to tape. This is analogous to dmfsfree.

Note: DCM uses parameters that are similar to those used for the disk MSP, although some names are different. See "DCM Policies" on page 75.

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 42). 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:

```
11:37:40-V 147432-dmfsmon /dmf3 - free_space=15.44, minimum=15
11:39:40-V 147432-dmfsmon /dmf3 - free_space=15.18, minimum=15
11:40:30-I 147432-dmfsmon Started 151212 for execution on /dmf3
11:40:34-I 151212-dmfsfree /dmf3 - Number of blocks in the filesystem = <math>33544832
11:40:34-I 151212-dmfsfree /dmf3 - Number of blocks in the free space target = 8386208 (25%)
11:40:34-I 151212-dmfsfree /dmf3 - Number of blocks currently free = 5029896 (15.0%)
11:40:34-I 151212-dmfsfree /dmf3 - Number of blocks to free = 3356312 (10.0%)
11:40:34-I 151212-dmfsfree /dmf3 - Number of blocks in the migration target =
23481382 (70%)
11:40:34-I 151212-dmfsfree /dmf3 - Number of blocks currently migrated = 15817032 (47.2%)
11:40:34-I 151212-dmfsfree /dmf3 - Number of blocks to migrate = 2634454 (7.9%)
11:40:34-I 151212-dmfsfree /dmf3 - Summary of files: online = 6277, offline =
1909, unmigrating = 0, partial = 594.
11:40:34-I 151212-dmfsfree /dmf3 - Number of candidates = 6681, rejected files = 0, rejected ranges = 290
11:40:42-I 151212-dmfsfree /dmf3 - Migrated 2635416 blocks in 559 files.
11:40:42-I 151212-dmfsfree /dmf3 - Freed 3367326 blocks in 853 requests.
11:40:42-0 151212-dmfsfree /dmf3 - Exiting: minimum reached - targets met by outstanding requests.
```

The DMF Daemon

The DMF daemon, dmfdaemon(8), is the core component of DMF. The daemon exchanges messages 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.

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

- "Daemon Processing"
- "DMF Daemon Database and dmdadm" on page 129
- "Daemon Logs and Journals" on page 136

Daemon Processing

After initialization, dmfdaemon performs the following steps:

- 1. Isolates itself as a daemon process.
- 2. Checks for the existence of other dmfdaemon processes. If another dmfdaemon exists, the newer one terminates immediately.
- 3. Initializes the daemon log.
- 4. Opens the daemon database.
- 5. Initializes the daemon request socket.
- 6. Initiates the MSPs and LSs.
- 7. Enters its main request processing.

The daemon uses log files and journal files as described in "Daemon Logs and Journals" on page 136.

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 <code>HOME_DIR/daemon_name</code> (<code>HOME_DIR</code> is specified by the <code>HOME_DIR</code> 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 <code>command_number</code> > 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	Creates a database record.
delete	Deletes the specified database record(s).
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 the specified database record(s).

The syntax for the dmdadm directives is summarized as follows:

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

The parameters have the following meanings:

- The *selection* parameter specifies the records to be acted upon.
- The *limit* parameter restricts the records acted upon.
- The *bfid* parameter for the create directive specifies the bit-file-identifier (BFID) for the record being created.
- The *settings* parameter for the create and update directives specifies one or more fields and their values.
- The *format* parameter selects the way in which output is displayed. Any program or script that parses the output from this command should explicitly specify a format; otherwise the default is used, which may change from release to release.

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

- A BFID or range of BFIDs
- The keyword all
- A period (.), which recalls the previous selection
- An expression involving any of the above, field value comparisons, and, or, or parentheses.

A field value comparison may use the following to compare a field keyword to an appropriate value:

```
< (less than)
> (greater than)
= (equal to)
! = (not equal to)
<= (less than or equal to)
>= (greater than or equal to)
```

The syntax for *selection* is as follows:

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

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
7ffffffff000f4411-305c74b2000004c8
all
origsize>lm
. and origage<7d

dmdadm Field and Format Keywords

The *field* parameter keywords listed below can be used as part of a *selection* parameter to select records. They can also be used in a *settings* parameter, as part of a keyword-value pair, to specify new values for a field, or in a *format* parameter. When specifying new values for fields, some of the field keywords are valid only if you also specify the -u (unsafe) option.

Keyword	Description
checkage (ca)	The time at which the 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 or Linux 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 or Linux 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 or Linux 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 or Linux time.

The time field keywords (checktime, deletetime, origtime, and updatetime) have a value of either now or UNIX or Linux *raw time* (seconds since January 1, 1970). These keywords display their value as raw time. The value comparison > used with the date keywords means newer than the value given. For example, >36000 is newer than 10AM on January 1, 1970, and >852081200 is newer than 10AM on January 1, 1997.

The age field keywords (checkage, deleteage, origage, and updateage) let you express time as a string. They display their value as an integer followed by the following:

```
w (weeks)
```

d (days)

h (hours)

m (minutes)

s (seconds)

For example, 8w12d7h16m20s means 8 weeks, 12 days, 7 hours, 16 minutes, and 20 seconds old.

The comparison > used with the age keywords means older than the value given (that is, >5d is older than 5 days).

A *limit* parameter in a directive restricts the records acted upon. It consists of one of the following keywords followed by white space and then a value:

Keyword	Description
recordlimit (rl)	Limits the number of records acted upon to the value that you specify; an integer.
recordorder (ro)	Specifies the order that records are scanned; 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* parameter selects a format to use for the display. If, for example, you want to display fields in a different order than the default or want to include fields that are not included in the default display, you specify them with the format parameter. The format parameter in a directive consists of one of the following:

- format default
- format keyword
- format field-keywords

The format keyword form is intended for parsing by a program or script and therefore suppresses the headings.

The *field-keywords* may be delimited by colons or white space; white space requires the use of quotation marks.

Note: BFID is always included as the first field and need not be specified.

For any field that takes a byte count, you may append the letter k, m, or g (in either uppercase or lowercase) to the integer to indicate that the value is to be multiplied by one thousand, one million, or one billion, respectively.

The following is sample output from the dmdadm list directive; recordlimit 20 specifies that you want to see only the first 20 records.

adm 3>list all recordlimit 20

BFID	ORIG	ORIG	ORIG N	MSP	MSP	
	UID	SIZE	AGE N	NAME	KEY	
305c74b20000010	20934	69140480	537d s	silo1	88b49f	
305c74b20000013	26444	279290	537d s	silo1	88b4a2	
305c74b20000014	10634	67000	537d s	silo1	88b4a3	
305c74b20000016	10634	284356608	537d s	silo1	88b4a5	
305c74b20000018	10634	1986560	537d s	silo1	88b4a7	
305c74b20000001b	26444	232681	537d s	silo1	88b4aa	
305c74b20000001c	10015	7533688	537d s	silo1	88b4ab	
305c74b200000022	8964	23194990	537d s	silo1	88b4b1	
305c74b200000023	1294	133562368	537d s	silo1	88b4b2	
305c74b200000024	10634	67000	537d s	silo1	88b4b3	
305c74b200000025	10634	284356608	537d s	silo1	88b4b4	
305c74b200000026	10634	1986560	537d s	silo1	88b4b5	
305c74b200000027	1294	1114112	537d s	silo1	88b4b6	
305c74b200000028	10634	25270	537d s	silo1	88b4b7	
305c74b200000029	1294	65077248	537d s	silo1	88b4b8	
305c74b20000002b	9244	2740120	537d s	silo1	88b4ba	
305c74b200000064	9335	9272	537d s	silo1	88b4f3	
305c74b200000065	9335	10154	537d s	silo1	88b4f4	
305c74b200000066	9335	4624	537d s	silo1	88b4f5	
305c74b20000067	9335	10155	537d s	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 50.

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.19980611.110429
      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 <code>JOURNAL_SIZE</code> 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 50.



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 202.

The DMF Lock Manager

The dmlockmgr(8) process must be executing at all times for any DMF process to safely access and update a DMF database. The dmlockmgr process and its clients (such as dmatls, dmfdaemon(8), dmvoladm(8), and dmcatadm(8)) communicate through files, semaphores, and message queues. There are times when abnormal process terminations will result in non-orderly exit processing that will leave files and/or interprocess communication (IPC) resources allocated. As a DMF administrator, periodically you will want to look for these resources to remove them.

Note: *HOME_DIR* and *SPOOL_DIR* refer to the values of the HOME_DIR and SPOOL_DIR parameter, respectively, in the DMF configuration file. See Chapter 2, "Configuring DMF" on page 29.

The dmlockmgr files used by the database utilities are found in several different places. There are the following types of files:

- "dmlockmgr Communication and Log Files" on page 139
- "dmlockmgr Individual Transaction Log Files" on page 141

dmlockmgr Communication and Log Files

The dmlockmgr communication and activity log files are all found in a directory formed by HOME_DIR/RDM_LM. The HOME_DIR/RDM_LM and HOME_DIR/RDM_LM/ftok_files directories contain the token files used to form the keys that are used to create and access the IPC resources necessary for the dmlockmgr to communicate with its clients, its standard output file, and the transaction file.

The dmlockmgr token files have the form shown in Table 5-1 on page 140.

Table 5-1 dmlockmgr Token Files

Description
Used by the dmlockmgr and its clients to access dmlockmgr's semaphore and input message queue
Preallocated token files that are not currently in use. As processes attempt to connect to dmlockmgr, these files will be used and renamed as described below. <i>nnnn</i> is a four digit number from 0000 through 0099.
The renamed version of the preallocated token files. <i>nnnn</i> is a four digit number from 0000 through 0099. <i>xxx</i> is a three-character process identifier with the following meaning:
 atr = dmatread ats = dmatsnf cat = dmcatadm ddb = dmdadm dmd = dmfdaemon dmv = dmmove hde = dmhdelete lfs = dmloadfs lib = dmatls sel = dmselect vol = dmvoladm pid is the numeric process ID of the process connected to dmlockmgr.

The IPC resources used by DMF are always released during normal process exit cleanup. If one of the dmlockmgr client processes dies without removing its message queue, dmlockmgr will remove that queue when it detects the death of the client. The token files themselves are periodically cleaned up by the dmlockmgr process.

Note: Normally, the dmlockmgr process is terminated as part of normal shutdown procedures. However if you wish to stop dmlockmgr manually, you must use the following command:

/usr/sbin/dmclripc -u dmlockmgr -z HOME_DIR/RDM_LM

This command will do all of the necessary IPC resource and token file maintenance.

If the dmlockmgr process aborts, all DMF processes must be stopped and restarted in order to relogin to a new dmlockmgr process. If the dmfdaemon or dmatls processes abort during a period when the dmlockmgr has died, when they restart they will attempt to restart the dmlockmgr. The new dmlockmgr process will detect existing DMF processes that were communicating with the now-dead copy of dmlockmgr, and it will send a termination message to those DMF processes.

The dmlockmgr maintains a log file that is named as follows, where *yyyy, mm*, and *dd* are the year, month, and day:

HOME_DIR/RDM_LM/dmlocklog.yyyymmdd

The log file is closed and a new one opened at the first log request of a new day. These files are not typically large files, but a new file will be created each day. These log files are removed via the run_remove_log.sh daemon task command. For more information about run_remove_log.sh, see "Configuring Daemon Maintenance Tasks" on page 50.

dmlockmgr Individual Transaction Log Files

The individual transaction log files have the following form:

prefix.log

where *prefix* is the same format as the token file name described in Table 5-1 on page 140 as £tnnnn.xxxpid. The prefix associates a log file directly with the token file of the same name.

Most of these log files will be created in the *HOME_DIR* under the daemon's and library servers' subdirectories. In almost all cases, the processes that create these log files will remove them when they exit. However, if a process terminates abnormally, its log file may not be removed. Transaction log files can sometimes become quite

large, on the order of 10's of Mbytes. Most of these orphaned log files will be removed by the daemon as part of its normal operation.

Several DMF commands allow accessing copies of database files in places other than the <code>HOME_DIR</code>. If an orphaned log is encountered in a location other than in the <code>HOME_DIR</code>, it may be removed after it is clear that it is no longer in use. In order to verify that it is no longer in use, search the <code>HOME_DIR/RDM_LM/ftok_files</code> directory for a file with the same name as the prefix of the log file. If no such <code>ftok_files</code> file exists, it is safe to remove the log file.

The transaction activity file, $HOME_DIR/RDM_LM/vista.taf$, is the transaction log file that contains information about active transactions in the system. It is used to facilitate automatic database transaction processing.



Caution: Do not delete the HOME_DIR/RDM_LM/vista.taf file.

Media-Specific Processes and Library Servers

Media-specific processes (MSPs) and library servers (LSs) migrate files from one media to another:

- The file transfer protocol (FTP) MSP allows the DMF daemon to manage data by moving it to a remote machine.
- The disk MSP migrates data to a directory that is accessible on the current systems and would be a cache disk if used in disk cache manager (DCM) mode.
- The tape LS copies files from a disk to a tape or from a tape to a disk. The LS can manage multiple active copies of a migrated file. The LS contains of one or more volume groups. 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:

- "LS Operations" on page 144
- "FTP MSP" on page 176
- "Disk MSP" on page 178
- "Disk MSP and Disk Cache Manager (DCM)" on page 180
- "dmdskvfy Command" on page 181
- "Moving Migrated Data between MSPs and Volume Groups" on page 181
- "Converting from an IRIX DMF to a Linux DMF" on page 182
- "LS Error Analysis and Avoidance" on page 184
- "LS Drive Scheduling" on page 186
- "LS Status Monitoring" on page 187

LS Operations

The LS consists of the following programs:

- dmatls
- dmatwc
- dmatrc

The DMF daemon executes dmatls as a child process. In turn, dmatls executes dmatwc (the write child) to write data to tape and dmatrc (the read child) to read data from tape.

The dmatls program maintains the following types of records in its database:

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

The database 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 147, and "VOL Database Records" on page 148.

The LS provides a mechanism for copying active data from volumes that contain largely obsolete data to volumes that contain mostly active data. This process is referred to as *volume merging*. Data on LS volumes becomes obsolete when users delete or modify their files. Volume merging can be configured to occur automatically (see "Configuring Maintenance Tasks for the LS" on page 97). It can also be triggered by marking LS volumes as sparse with the dmvoladm(8) command.

The LS provides two utilities that read LS volumes directly:

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

LS Directories

Each instance of the 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 <code>HOME_DIR</code>, <code>JOURNAL_DIR</code>, and <code>SPOOL_DIR</code>, whose values are referred to as <code>HOME_DIR</code>, <code>JOURNAL_DIR</code>, and <code>SPOOL_DIR</code> in this document. A given instance of the LS creates a subdirectory named after itself in each of these three directories.

For example, if an instance of the LS is called cart1, its database files reside in directory $HOME_DIR/cart1$. If another instance of the LS is called cart2, its database files reside in $HOME_DIR/cart2$. If an instance of the LS is called cart3, its database files reside in $HOME_DIR/cart3$.

Similarly, LS cart1 stores its journal files in directory JOURNAL_DIR/cart1 and its log files and other working files in SPOOL_DIR/cart1.

Media Concepts

The LS takes full advantage of the capabilities of modern tape devices, including data compression and fast media positioning. To accommodate these capabilities and to provide recovery from surface or other media defects, dmatls uses a number of structural concepts built on top of traditional tape structure.

The components are as follows:

- The *block* is the basic structural component of most tape technologies. It is the physical unit of I/O to and from the media. The optimal block size varies with the device type. For example, the default block size for 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 147). 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 volume group writes chunks into the zone until one of three conditions occurs:

- The zone size is exceeded
- The 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 dmatls to start a process to write files to tape.
- The 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.

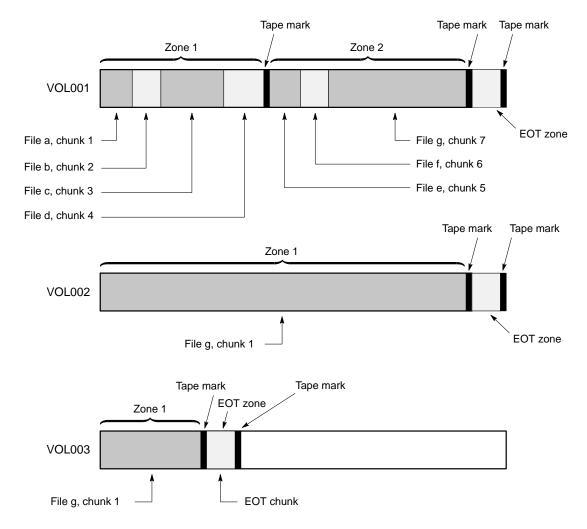


Figure 6-1 Media Concepts

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 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

The database definition file (in the same directory) that describes these files and their record structure is named 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 154).

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 LS database, tpvrdm, contain information about each volume that exists in the pool of tapes to be used by dmatls. These records are indexed by the volume serial number (VSN) of each volume and contain information such as the following:

- Volume's type
- Estimated capacity
- Label type
- A number of flags indicating the state of the volume
- Volume group or allocation group

Unlike the CAT records, you must create the VOL records in the database before using dmatls for the first time.

The VOL portion of the LS database consists of the following files:

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

The database definition file (in the same directory) that describes these files and their record structure is named 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 163, provides facilities to create, query, and modify VOL records in the database. Additional database maintenance utilities are described in "Database Recovery" on page 202.

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

LS Journals

Each instance of dmatls protects its database by recording every transaction in a journal file. Journal file pathnames have the following format:

JOURNAL_DIR/ls_name/libsrv_db.yyyymmdd[.hhmmss]

The LS creates journal files automatically.

Existing journal files are closed and new ones created in two circumstances:

- When the first transaction after midnight occurs
- When the journal file reaches the size defined by the JOURNAL_SIZE configuration parameter

When the first transaction after midnight occurs, the existing open journal file is closed and the suffix .235959 is appended to the current file name no matter what the time (or date) of closing. The closed file represents the last (or only) transaction log of the date *yyyymmdd*. A new journal file with the current date is then created.

When the journal file reaches JOURNAL_SIZE, the file is closed and the suffix .hhmmss is added to the name; hh, mm, and ss represent the hour, minute, and second of file closing. A new journal file with the same date but no time is then created.

For example, the following shows the contents of a *JOURNAL_DIR/ls_name* directory on 15 June 2004:

```
libsrv_db.20040527.235959 libsrv_db.20040606.235959 libsrv_db.20040529.235959 libsrv_db.20040608.235959 libsrv_db.20040530.235959 libsrv_db.20040601.235959 libsrv_db.20040601.235959 libsrv_db.20040602.235959 libsrv_db.20040602.235959 libsrv_db.20040603.235959 libsrv_db.20040603.235959 libsrv_db.20040603.235959 libsrv_db.20040604.235959 libsrv_db.20040604.235959 libsrv_db.20040614.235959 libsrv_db.20040604.235959 libsrv_db.20040614.235959 libsrv_db.20040605.235959 libsrv_db.20040615
```

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 50.

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

LS Logs

All DMF MSPs and LSs maintain log files named msplog. *yyyymmdd* in the MSP/LS spool directory which, by default, is *SPOOL_DIR/mspname*. *SPOOL_DIR* is configured in the base object of the configuration file; *mspname* is the name of the MSP/LS in the daemon object of the configuration file; *yyyymmdd* is the current year, month, and day.

These log files are distinct from the logs maintained by the DMF daemon; however, some of the messages that occur in the daemon log are responses that the MSP/LS generates. The content of the log is controlled by the MESSAGE_LEVEL configuration parameter. For a description of the levels of logging available, see the dmf.conf(5) man page.

The msplog. *yyyymmdd* file is the primary log for the LS and contains most of the messages. This file is written by dmatls, 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 <code>SPOOL_DIR/msp_name/msplog.yymmdd</code> files. Timing information provided (such as MB transferred per second) should not be used as an accurate benchmark of actual data transfer rates. This information is provided for monitoring DMF and should only be used in comparison to similar data provided by DMF. Text in all uppercase references a parameter defined in the DMF configuration file. You can reference the comments in the sample configuration file or in the dmf.conf(5) man page for a more detailed definition of these parameters.

Note: Because the LS will continue to create log files and journal files without limit, you must remove obsolete files periodically by configuring the run_remove_logs.sh and run_remove_journals.sh tasks in the configuration file, as described in "Configuring Daemon Maintenance Tasks" on page 50.

Example 6-1 LS Statistics Messages

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

```
00:02:00-I 13902144-dmatls vq9a16.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: Get_File -
                                                    0 1
                                                                  0
                                                                        0
                                                  0
00:02:01-I 13902144-dmatls vg9a16.stats: Delete_File -
                                                          130
                                                                 Ο
00:02:01-I 13902144-dmatls vg9a16.stats: Cancel_Req -
                                                    0 12
00:02:01-I 13902144-dmatls vg9a16.stats: Flushall -
                                                          2
                                                                 0
                                                    0
                                                                        Ω
00:02:01-I 13902144-dmatls vg9a16.stats:
                                       Merge -
                                                    45
                                                          25
                                                                  0
                                                                       16
                                                                 0
00:02:01-I 13902144-dmatls vg9a17.stats: Put_File - 14 210
00:02:01-I 13902144-dmatls vg9a17.stats: Get_File -
                                                    0
                                                          1
                                                    0
                                                          178
00:02:01-I 13902144-dmatls vg9a17.stats: Delete_File -
                                                                 0
00:02:01-I 13902144-dmatls vg9a17.stats: Cancel_Reg -
                                                     Ο
                                                           8
                                                                  0
00:02:01-I 13902144-dmatls vg9a17.stats: Flushall -
                                                    0
                                                           2
                                                                  Λ
                                                                        Ω
00:02:01-I 13902144-dmatls vg9a17.stats:
                                       Merge -
                                                  18
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 information provided by these entries is defined as follows:

- children=1/0/0/7 represents the total child processes (1), the active child processes (0), the clean processes running (0), and the current maximum number of children the volume group may have (7). Clean children are used when a dmatrc or dmatwc process dies without cleaning up.
- btp=28098297/0/0 represents the bytes queued for putting (28098297), the threshold at which to start the next put child (0), and the bytes assigned to socket I/O (0)
- wc=0/7 represents the active write child processes (0) and the configured value of MAX_PUT_CHILDREN (7)
- 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 set of lines give the total amount of data put (such as 608.607 MB) and recalled (such as 114.270 MB).

The next set of six lines provide statistics for each type of volume group request. Statistics information is provided only for requests that have been issued since the LS was started. These lines have the following format:

```
request_name active successful errors canceled
```

active represents the number of requests not yet completed; successful represents the number of successfully completed requests; error represents the number of requests that completed with errors; canceled represents the number of canceled requests.

The last set of lines provide the following information:

- mc is the configured value for MERGE_CUTOFF, the cutoff to stop scheduling tapes for merging (such as 7)
- ms is the configured value for CACHE_SPACE, the merge cache space available (such as 500000000 bytes)
- mu is the merge cache space used (such as 133107712 bytes)
- sm is the number of socket merge children (0)

The LS write child (dmatwc) and read child (dmatrc) also produce statistics messages in the 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 *sparse*. To reclaim the unused space on these volumes, DMF provides a *volume merge* facility, which copies the active data from several sparse volumes to a new volume, thus freeing the sparse volumes for reuse. Volume merging can be configured to occur automatically by using the run_merge_tapes.sh or run_merge_mgr.sh tasks (see "Configuring Maintenance Tasks for the LS" on page 97).

Volume merging can also be done manually. dmatls performs merge operations whenever sparse volumes and the necessary resources exist at the same time. Use the dmvoladm select directive to mark volume group volumes as sparse. (The select directive is described in "dmvoladm Command" on page 163.) Because the merge processing occurs simultaneously with other DMF activities, it is easiest to configure DMF to automatically perform merges at night or during other periods of relatively low activity.

The dmatls utility can perform volume-to-volume merging. Volume-to-volume merging is accomplished by moving data across a socket connection between the LS tape read-child and the LS tape write-child. The benefit of using a socket to transfer data between volumes is that you do not have to reserve disk space. The drawback to using a socket for data transfer is the cost of linking the process that performs the read with the process that performs the write.

In busy environments that have heavy contention for tape drives, the close coupling between the socket's tape reader and tape writer can be costly, especially when short files are being transferred. For large files, the overhead and possible delays in waiting for both tapes to be mounted is small compared to the benefit of rapid transfer and zero impact on free disk space. For this reason, you can move small files through a disk cache and big files through a socket. This process is mediated by the following configuration parameters:

Parameter	Description
CACHE_SPACE	Specifies the amount of disk space that will be used to temporarily store chunks during a merge operation.
CACHE_DIR	Specifies the directory into which the LS will store chunks while merging them from sparse tapes. If CACHE_DIR is not specified, TMP_DIR is used.

MAX_CACHE_FILE Specifies the largest chunk that will be stored

temporarily on disk during a merge operation.

MERGE_CUTOFF Specifies the number of child processes after which the

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 volume group simultaneously if they share a tape device. If you initiate a merge process on more than one 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_merge_tape.sh or run_merge_mgr.sh tasks, ensure that your cron jobs that automatically initiate volume merging refrain from initiating a second merge process until after all previously initiated merges are complete. You can accomplish this by using the dmvoladm command within the cron job to check for tapes that have the hsparse flag, as shown in the following example for an LS with two volume groups:

dmcatadm Command

The dmcatadm(8) command provides maintenance services for CAT records in the 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 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 the specified CAT records.
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 LS database obtained from running the dump directive.
quit	Stops program execution after flushing any changed database records to disk. The abbreviation ${\tt q}$ and the string exit produce the same effect.
set	Specifies the fields to be displayed in subsequent list directives.
update	Modifies the specified CAT records.
verify	Verifies the LS database against the 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 bfid settings ...
delete selection [limit]
dump selection [limit]
help
list selection [limit] [format]
load filename
quit (or q or exit)
set [format]
update selection [limit] to settings...
verify selection [entries] [vgnames] [limit]
```

The parameters are as follows:

- The *selection* parameter specifies the records to be acted upon. The value for *selection* can be one of the following:
 - A bfid or range of bfids in the form bfid [-] [bfid]. bfid- specifies all records starting with bfid, and -bfid specifies all records up to bfid.
 - The keyword all
 - A period (.), which recalls the previous selection
 - An expression involving any of the above, field value comparisons, and, or, or parentheses.

A field value comparison may use the following to compare a field keyword to an appropriate value:

```
< (less than),
> (greater than)
= (equal to)
!= (not equal to)
<= (less than or equal to)
>= (greater than or equal to)
```

The syntax for selection is as follows:

```
      selection
      ::= or-expr

      or-expr
      ::= and-expr [ or or-expr ]

      and-expr
      ::= nested-expr [ and or-expr ]

      nested-expr
      ::= comparison | (or-expr )
```

```
key-range | field-keyword op field-value
comparison
               ::=
                       < | > | = | != | <= | >=
               := bfid [ - bfid] | [bfid - [bfid]] | key-macro
bfid-range
key-macro
               ::=
                      all
field-keyword
               ::=
                       name or abbreviation of the record field
field-value
               ::=
                       appropriate value for the field
key
               ::=
                      character representation of the record bfid
```

Thus valid selections could be any of the following:

```
305c74b200000010-305c74b200000029
7ffffffff000f4411-
-305c74b2000004c8
all
chunkoffset>0
chunknumber>0 and writeage<5d
. and writeage>4d
vsn=S07638
```

- The *limit* parameter restricts the records acted upon.
- The *bfid* parameter for the create directive specifies the bit-file-identifier (BFID) for the record being created. The value for *bfid* may be a bit-file identifier (BFID) designator in the form of a hexadecimal number.
- The settings parameter for the create and update directives specify one or more fields and their values.
- The *format* parameter selects the way in which output is displayed. Any program or script that parses the output from this command should explicitly specify a format; otherwise the default is used, which may change from release to release.
- The *entries* parameter specifies a file of daemon database entries.
- The vgnames parameter specifies the names of the volume groups associated with the records.

dmcatadm Keywords

You can use the *field* keywords listed below as part of a *selection* parameter to select records, in a *format* parameter, or in a *settings* parameter to specify new values for a field; in that case, you must specify a keyword-value pair. A keyword-value pair consists of a keyword followed by white space and then a value. When specifying new values for fields, some of the keywords are valid only if you also specify the -u

(unsafe) option. The abbreviation for each of the keywords is given in parenthesis following its name.

Keyword	Description
cflags (cf)	For future use.
chunkdata (cd)	Specifies the actual number of bytes written to tape by the 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. Not valid as part of a <i>settings</i> parameter in an update directive.
chunkoffset (co)	The byte offset within the file where the chunk begins; an integer. For example, the first chunk of a file has chunkoffset 0. If that first chunk is 1,000,000 bytes long, the second chunk would have chunkoffset 1000000. This is valid only in unsafe (-u) mode.
chunkpos (cp)	The block offset within the zone where the chunk begins — a hexadecimal integer. For example, the first chunk in a zone has chunkpos 1. A value of 0 means unknown. Valid only in unsafe (-u) mode.
filesize (fs)	The original file size in bytes, an integer. This is valid only in unsafe (-u) mode.
readage (ra)	The date and time when the chunk was last read; the same as readdate, except specified as <i>age</i> .
readcount (rc)	The number of times the chunk has been recalled to disk; an integer.
readdate (rd)	The date and time when the chunk was last read, an integer that reflects raw UNIX or Linux time.
volgrp (vg)	The volume group name. This keyword is valid for LSs only. This keyword is not valid as part of a <i>settings</i> parameter.

vsn (v) The volume serial numbers; a list of one or more 6-character alphanumeric volume serial numbers separated by colons (:). This keyword is not valid as part of a settings parameter in an update directive. writeage (wa) The date and time when the chunk was written; the same as writedate, except specified as age. This is valid only in unsafe (-u) mode. writedate(wd) The date and time when the chunk was written, an integer that reflects raw UNIX or Linux time. This is valid only in unsafe (-u) mode. zoneblockid (zb) Allows just the block ID portion of the zonepos to be displayed, returned, or changed. This is valid only in unsafe (-u) mode. zonenumber (zn) Allows just the zone number portion of the zonepos to be displayed, returned, or changed. This is valid only in unsafe (-u) mode. zonepos (zp) The physical address of the zone on the volume, expressed in the form integer/hexadecimal-integer, designating a zone number and block ID. A value of zero is used for hexadecimal-integer if no block ID is known. integer is the same as zonenumber, and hexadecimal-integer is the same as zoneblockid. This is valid only in unsafe (-u) mode.

The date field keywords (readdate and writedate) have a value of either now or raw UNIX or Linux time (seconds since January 1, 1970). These keywords display their value as raw UNIX or Linux time. The value comparison > used with the date keywords means newer than the value given. For example, >36000 is newer than 10AM on January 1, 1970, and >852081200 is newer than 10AM on January 1, 1997.

The age field keywords (readage and writeage) let you express time as *age* in a string in a form. They display their value as an integer followed by the following:

```
w (weeks)
```

d (days)

h (hours)

m (minutes)

s (seconds)

For example, 8w12d7h16m20s means 8 weeks, 12 days, 7 hours, 16 minutes, and 20 seconds old.

The comparison > used with the age keywords means older than the value given (that is, >5d is older than 5 days).

The *limit* parameter in a directive limits the records acted upon. It consists of one of the following keywords followed by white space and then a value:

Keyword	Description
recordlimit (rl)	Limits the number of records acted upon to the value that you specify; an integer.
recordorder (ro)	Specifies the order that records are scanned; may be key, vsn, or data. key specifies that records are scanned in ascending order of the chunk key. vsn specifies that records are scanned in ascending order of the chunk VSN. data specifies that records are scanned in the order in which they are stored in the database, which is fastest but essentially unordered.

The following keywords specify files of daemon database entries:

Keyword	Description
entries (e)	Specifies a file of daemon database entries. This keyword applies to the verify directive and consists of the word entries (or its abbreviation e) followed by a string.
vgnames (vn)	Specifies the names of the volume groups associated with the record. This keyword applies to the verify directive and consists of the word vgnames (or its abbreviation vn) followed by a quoted, space-separated list of names.

The *format* parameter in a directive consists of the word format followed by white space and then either the word default, the word keyword, or a list of field keywords.

The keyword form, intended for parsing by a program or script, suppresses the headings.

If a list of field keywords is used in the *format* parameter, they may be delimited by colons or spaces, but spaces will require the use of quoting.

Note: The BFID is always included as the first field and need not be specified.

For any field that takes a byte count, you may append the letter k, m, or g (in either uppercase or lowercase) to the integer to indicate that the value is to be multiplied by one thousand, one million, or one billion, respectively.

For information about the role of the dmcatadm(8) command in database recovery, see "Database Recovery" on page 202.

Example 6-2 dmcatadm list Directive

The following is sample output from the dmcatadm list directive. The file with key 3273d5420001e244 has two chunks because it spans two physical tape volumes; the first chunk contains bytes 0 through 24821759, and the second chunk bytes 24821760 (the CHUNK OFFSET) to the end of the file.

adm 3>list 3273d5420001e242- recordlimit 10

	CHUNK	CHUNK	CHUNK	WRITE	
VSN	NUM	LENGTH	OFFSET	AGE	KEY
S12940	13	77863935	0	61d	3273d5420001e242
S12936	168	24821760	0	61d	3273d5420001e244
S12945	1	23543808	24821760	61d	3273d5420001e244
S12945	2	51019776	0	61d	3273d5420001e245
S12938	59	45629440	0	61d	3273d5420001e246
S12938	60	35586048	0	61d	3273d5420001e247
S12944	3	9568256	0	61d	3273d5420001e248
S12944	4	14221312	0	61d	3273d5420001e249
S12944	5	458752	0	61d	3273d5420001e24a
S12944	6	14155776	0	61d	3273d5420001e24b

The following is sample output from the dmcatadm list directive for an LS. The file with key 3b4b28f2000000000000000e80 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
3b4b28f20000000000000ae80	1d	0	2305938	120	000700
3b4b28f20000000000000ae80	4d	0	2305938	32	000727
3b4b28f20000000000000ae82	1d	0	234277	247	003171
3b4b28f20000000000000ae82	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
aan 2-11	.bc vb.ii=000727		Баша	EOT	EOT		tan / Ein
			DATA	FOI	FOI		WR/FR
VSN	VOLGRP LB	DATA LEFT	WRITTEN	CHUNK	ZONE	HFLAGS	AGE
000727	vg8a05 al	159.107337	200.443980	102	6		1d

dmcatadm Text Field Order

The text field order for chunk records generated by the dmdump(8), dmdumpj(8), and the dump directive in dmcatadm is listed below. This is the format expected by the load directives in dmcatadm:

- 1. C (indicates the chunk record type)
- 2. bfid (hexadecimal digits)
- 3. filesize
- 4. writedata
- 5. readdate
- 6. readcount

- 7. chunkoffset
- 8. chunklength
- 9. chunkdata
- 10. chunknumber
- 11. flags (in octal)
- 12. zoneposition (zonenumber/zoneblockid) (in hexadecimal)
- 13. vsn
- 14. chunkpos (in hexadecimal)

dmvoladm Command

The dmvoladm(8) command provides maintenance services for VOL records in the 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 <code>command_number</code> > 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 <code>dmvoladm</code> 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 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 the specified VOL records.

dump	Prints the specified VOL records to standard output in ASCII. Each database field is separated by the pipe character (1).
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 <i>expression</i> to hsparse on.
set	Specifies the fields to be shown in subsequent list directives.
update	Modifies the specified VOL records.
verify	Verifies the LS databases against the dmfdaemon databases.

The syntax for the dmvoladm directives is summarized as follows:

```
count [limit]
create vsnlist volgrpspec [settings]
delete selection [limit]
dump selection [limit]
help
list selection [limit] [format]
load filename
quit (or q, or exit)
repair selection
select selection [limit]
set format
update selection [limit] to settings
verify selection
```

The *volgrpspec* parameter consists of the keyword volgrp (or vg), followed by a value for that keyword.

The value for *vsnlist* may be a single 6-character volume serial number (VSN) or a range of VSNs separated by the hyphen (-) character. A VSN string is case insensitive and may consist entirely of letters, entirely of digits, or a series of letters followed by digits. In a range of VSNs, the first must be lexically less than the second.

The value for selection may be one of the following:

- A *vsnlist* or range of VSNs in the form *vsn*[-*vsn*]. *vsn* specifies all records starting with *vsn*, and -*vsn* specifies all records up to *vsn*.
- A period (.), which recalls the previous selection
- The name of one of the flags in the keyword list that follows in this section.
- One of the words all, used, empty, or partial or any of the hflags, whose meanings are as follows:

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

• An expression involving *vsnlists*, field-value comparisons, and, or, or parentheses.

A field value comparison may use the following to compare a field keyword to an appropriate value:

```
< (less than)
> (greater than)
= (equal)
!= (not equal)
<= (less than or equal to)
>= (greater than or equal to)
```

The syntax for *selection* is as follows:

```
      selection
      ::=
      or-expr

      or-expr
      ::=
      and-expr [ or or-expr ]

      and-expr
      ::=
      nested-expr [ and or-expr ]

      nested-expr
      ::=
      comparison | (or-expr )

      comparison
      ::=
      vsnlist | field-keyword op field-value

      op
      ::=
      < | > | = | != | >= | <=</td>
```

Thus valid selections could be any of the following:

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

dmvoladm Field Keywords

You can use the *field* keywords listed below as part of a *selection* parameter to select records, in a *format* parameter, or in a *settings* parameter to specify new values for a field; in that case, a keyword-value pair must be specified. A keyword-value pair consists of a keyword followed by white space and then a value. When specifying new values for fields, some of the keywords are valid only if you also specify the -u (unsafe) option:

Keyword	Description
blocksize (bs)	Specifies the data block size in bytes when the tape was first written; an integer. This keyword is used only when mounting tapes with existing valid data. When an empty tape is first written, the volume group uses the default value for the tape type, unless it is overridden by a value in the BLOCK_SIZE parameter for the drive group in the DMF configuration file. This is valid only in unsafe (-u) mode.
chunksleft(cl)	Specifies the number of active chunks on the volume; an integer. This is valid only in unsafe (-u) mode.
dataleft (dl)	Specifies the number of bytes of active data on the volume. You specify this number as an integer, but for

readability purposes it is displayed in megabytes (MB). This is valid only in unsafe (-u) mode. datawritten (dw) Specifies the maximum number of bytes ever written to the volume. You specify this number as an integer, but for readability purposes it is displayed in MB. This is valid only in unsafe (-u) mode. eotblockid (eb) Specifies the blockid of the chunk containing the end-of-tape marker; a hexadecimal integer. This is valid only in unsafe (-u) mode. eotchunk (ec) Specifies the number of the chunk containing the end-of-tape marker; an integer. This is valid only in unsafe (-u) mode. eotpos (ep) Specifies the absolute position of the end-of-tape marker zone in the form *integer/hexadecimal-integer*, designating a zone number and block ID. A value of zero is used for hexadecimal-integer if no block ID is known. integer the same as eotzone, and *hexadecimal-integer* is the same as eotblockid. This is valid only in unsafe (-u) mode. eotzone (ez) Specifies the number of the zone containing the end-of-tape marker; an integer. This is valid only in unsafe (-u) mode. hflags (hf) Specifies the flags associated with the record. See the description of *flags* keywords. Not valid as part of a settings parameter. label (lb) Specifies the label type: al for ANSI standard labels; sl for IBM standard labels; or nl for nonlabeled volumes. The default is al. tapesize (ts) Specifies the estimated capacity in bytes; an integer. The default is 215 MB. threshold (th) Specifies the ratio of dataleft to datawritten as a percentage. This field is valid only as part of a selection parameter. upage (ua) Specifies the date and time of the last update to the volume's database record. The same as for update, except that it is expressed as age. This is not valid as part of a settings parameter.

update (ud)	Specifies the date and time of the last update to the volume's database record, expressed as an integer that reflects raw UNIX or Linux time. This is not valid as part of a <i>settings</i> parameter.
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.
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 <i>age</i> . This is valid only in unsafe (-u) mode.
wfdate (wd)	Specifies the date and time that the volume was written to or freed for reuse, expressed as an integer that reflects raw UNIX or Linux time. This is valid only in unsafe (-u) mode.

The date field keywords (update and wfdate) have a value of either now or UNIX or Linux *raw time* (seconds since January 1, 1970). These keywords display their value as raw time. The value comparison > used with the date keywords means newer than the value given. For example, >36000 is newer than 10AM on January 1, 1970, and >852081200 is newer than 10AM on January 1, 1997.

The age field keywords (upage and wfage) let you express time as age as a string.

The age keywords display their value as an integer followed by the following:

```
w (weeks)
d (days)
h ( hours)
m ( minutes)
s (seconds)
```

For example, 8w12d7h16m20s means 8 weeks, 12 days, 7 hours, 16 minutes, and 20 seconds old.

The comparison > used with the age keywords means older than the value given (that is, >5d is older than 5 days).

The *limit* parameter in a directive limits the records acted upon. It consists of one of the following keywords followed by white space and then a value. The abbreviation for the keyword is given in parentheses following its name, if one exists:

Keyword	Description
datalimit (no abbreviation)	Specifies a value in bytes. The directive stops when the sum of dataleft of the volumes processed so far exceeds this value.
recordlimit (rl)	Specifies a number of records; an integer. The directive stops when the number of volumes processed equals this value.
recordorder (ro)	Specifies the order that records are scanned; may be either data or vsn. vsn specifies that the records are scanned in ascending order of the chunk VSN. data specifies that the records are scanned in the order in which they are found in the database, which is fastest but essentially unordered.

The *format* parameter in a directive consists of the word format followed by white space and then either the word default, the word keyword, or a list of field and or flag keywords.

The keyword form, intended for parsing by a program or script, suppresses the headings.

If a list of field or flag keywords is used in the format expression, they may be delimited by colons or spaces, but spaces will require the use of quoting. The VSN is always included as the first field and need not be specified.

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

Keyword	Description
herr (he)	Indicates a database inconsistency for this volume. It is displayed as e
hflags (no abbreviation)	(Not valid as part of a <i>settings</i> parameter.) Shows the complete set of hold flags as a 9–character string. Each flag has a specific position and alphabetic value. If the flag is off, a dash (–) is displayed in its position; if the flag is on, the alphabetic character is displayed in that position.
hfree (no abbreviation)	Indicates that the volume has no active data and is available for reuse after HFREE_TIME has expired, displayed as -f See the dmf.conf(5) man page for information about the HFREE_TIME

	configuration parameter. This is valid only in unsafe (-u) mode.
hfull (hu)	Indicates that the volume cannot hold any more data; displayed asu
hlock (hl)	Indicates that the tape cannot be used for either input or output. This is a transient condition; the flag will be cleared by the LS after REINSTATE_VOLUME_DELAY has expired and at LS startup. Displayed as1
hoa (ho)	Indicates that the volume is not to be used for either input or output, displayed aso This value is only set or cleared by the site administrator.
hro (hr)	Indicates that the volume is read-only, displayed asr; this inhibits the LS from using the volume for output. This value is only set or cleared by the site administrator.
hsitel (hl)	Reserved for site use; ignored by DMF. Not normally displayed; see the dmvoladm(8) man page for details. hsite2, hsite3, and hsite4 are also available.
hsparse (hs)	Indicates that the volume is considered sparse and thus a candidate for a volume merge operation, displayed ass
hvfy (hv)	Indicates that this tape should be tested and/or replaced when next empty; until that time, it is read-only. Displayed asv This value is set by DMF but only cleared by the site administrator.

For any field that takes a byte count, you may append the letter k, m, or g (in either uppercase or lowercase) to the integer to indicate that the value is to be multiplied by one thousand, one million, or one billion, respectively.

For information about the role of the dmvoladm command in database recovery, see "Database Recovery" on page 202. For details about dmvoladm syntax, see the man page.

Example 6-3 dmvoladm list Directives

The following example illustrates the default format for the list directive when using an LS. The column marked HFLAGS uses a format similar to the ls -l command in that each letter has an assigned position and its presence indicates that

the flag is "on". The positions spell out the string eforvlus, representing herr, hfree, hoa, hro, hvfy, hlock, hfull, and hsparse.

adm 1> list 000683-000703

				DATA	EOT	EOT		WR/FR
VSN	VOLGRP	LB	DATA LEFT	WRITTEN	CHUNK	ZONE	HFLAGS	AGE
000683	vg8a01	al	0.00000	0.000000	1	1		3d
000700	vg8a00	al	267.539255	287.610294	124	7	u	2d
000701	vg8a00	al	288.342795	308.147798	136	8	u	2d
000702	vg8a00	al	255.718902	288.302830	120	7	u	2d
000703	ag8	al	0.00000	0.000000	1	1		3d

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

adm 1>list hfull

				DATA	EOT	EOT		WR/FR
VSN	VOLGRP	LB	DATA LEFT	WRITTEN	CHUNK	ZONE	HFLAGS	AGE
000701	vg8a00	al	288.342795	308.147798	136	8	u	2d
000702	vg8a00	al	255.718902	288.302830	120	7	u	2d
000704	vg8a00	al	252.294122	292.271410	119	7	u	2d
000705	vg8a00	al	250.207666	304.603059	143	7	u	2d
000706	vg8a00	al	265.213875	289.200534	144	7	u	2d
000707	vg8a00	al	278.744448	310.408119	140	7	u	2d
000708	vg8a00	al	260.827748	295.956588	136	7	u	2d
000709	vg8a00	al	253.481897	283.615678	138	8	u	2d
000710	vg8a00	al	265.100985	291.243235	141	7	u	2d
000711	vg8a00	al	276.288446	305.782035	144	7	u	2d
000712	vg8a00	al	250.415786	275.606243	138	7	u	2d
000716	vg8a00	al	287.964765	304.321543	144	7	u	2d
000717	vg8a00	al	280.795058	287.084534	144	7	u	2d
000718	vg8a00	al	0.000415	300.852018	180	27	u	3d
003127	vg9a01	al	417.383784	461.535047	209	10	u	2d
003128	vg9a01	al	427.773679	460.716741	229	11	u	2d

The following example shows one way you can customize the list format to show only the fields that you want to see. The other way is to use the set format command with the same keyword list.

adm 21>list S03232-S03254 format "eotchunk eotzone eotpos"

	EOT	EOT	
VSN	CHUNK	ZONE	EOTPOS
	1.0		
S03232	10	2	2/4294967295
S03233	2	2	2/4294967295
S03234	598	2	2/4294967295
S03235	18	2	2/4294967295
S03236	38	2	2/4294967295
S03237	92	2	2/4294967295
S03238	1	1	1/4294967295
S03239	1	1	1/4294967295
S03240	1	1	1/4294967295
S03241	325	2	2/4294967295
S03242	81	2	2/4294967295
S03243	26	2	2/4294967295
S03244	1	1	1/4294967295
S03245	26	2	2/4294967295
S03246	5	2	2/4294967295
S03247	186	2	2/4294967295
S03248	17	2	2/4294967295
S03249	526	2	2/4294967295
S03250	1	1	1/4294967295
S03251	533	2	2/4294967295
S03252	157	17	17/2147483648
S03253	636	2	2/4294967295
S03254	38	2	2/4294967295

The following example gives a convenient way to show the several flag bits in a way different from their usual representation.

adm 23>list 003232-003254 format "hfree hfull hlock hoa hro"

	hfree	hfull	hlock	hoa	hro
VSN					
003232	off	on	off	off	off
003233	off	off	off	off	off
003234	off	off	off	off	off

003235	off	off	off	off	off
003236	off	on	off	off	off
003237	off	on	off	off	off
003238	off	on	off	off	off
003239	off	on	off	off	off
003240	off	off	off	off	off
003241	off	on	off	off	off
003242	off	on	off	off	off
003243	off	off	off	off	off
003244	off	off	off	off	off
003245	off	on	off	off	off
003246	off	off	off	off	off
003247	off	on	off	off	off
003248	off	on	off	off	on
003249	on	off	off	off	on
003250	on	off	off	off	on
003251	on	off	off	off	on
003252	on	off	off	off	on
003253	off	on	off	off	on
003254	off	on	off	off	on

The following example shows how to display only those tapes assigned to the volume group named vg9a00.

adm 3>list vg=vg9a00

			DATA	EOT	EOT		WR/FR
VSN	VOLGRP LB	DATA LEFT	WRITTEN	CHUNK	ZONE	HFLAGS	AGE
003210	vg9a00 al	1.048576	1.048576	3	2		11d
003282	vg9a00 al	11.534336	11.534336	13	2		7d

dmvoladm Text Field Order

The text field order for volume records generated by the dmdump(8), dmdumpj(8), and the dump directive in dmvoladm is listed below. This is the format expected by the load directives in dmvoladm:

- 1. V (indicates the volume record type)
- 2. vsn
- 3. volgrp

- 4. lbtype
- 5. capacity
- 6. blocksize
- 7. hflags (in octal)
- 8. version
- 9. datawritten
- 10. eotchunk
- 11. eotposition (eotzone/eotblockid) (in hexadecimal)
- 12. dataleft
- 13. chunksleft
- 14. wfdate
- 15. update
- 16. id (in octal). This field indicates the type of process that last updated the record.

dmatread Command

Use the dmatread(8) command to copy all or part of the data from a migrated file back to disk. You might want to do this if, for example, a user accidentally deleted a file and did not discover that the deletion had occurred until after the database entries had been removed by the hard delete procedure. Using backup copies of the databases from before the hard delete was performed, dmatread can restore the data to disk, assuming that the tape volume has not been reused in the meantime.

Example 6-4 Restoring Hard-deleted Files Using dmatread

To copy migrated files back to disk, perform the following steps:

 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 LS databases, use a dmatread(8) command similar to the following:

dmatread -p /a/dmbackup -B 342984C5000000000084155

342984C5000000000084155 is the BFID of the file to be restored, and /a/dmbackup is the directory containing the backup copies of the LS databases. Your file will be restored to the current directory as B342984C500000000000084155

DMF does not know the original name of the file; you must manually move the restored data to the appropriate file.

If you have access to chunk and VSN information for the file to be restored, you can use the dmatread -c and -v options and avoid using backup copies of the LS database. In this case, dmatread will issue messages indicating that the chunk is not found in the current 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 LS volumes. You may also generate text database records that can be applied to the LS databases (using the load directive in dmcatadm and dmvoladm, respectively), in order to add the contents of a volume to the LS database (although this is impractical for large numbers of volumes).

dmatsnf can be used to verify one or more tape volumes against the LS databases. It also can be used to generate journal entries, which can be added to the 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 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 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.

FTP MSP Processing of Requests

The FTP MSP is always waiting for requests to arrive from the DMF daemon, but, to improve efficiency, it holds PUT and DELETE requests briefly and groups similar requests together into a single FTP session. No PUT request will be held longer than 60 seconds. No DELETE request will be held longer than 5 seconds. GET requests are not held. The MSP will stop holding requests if it has a large amount of work to do (more than 1024 individual files or 8 MB of data). The FTP MSP also limits the number of FTP sessions that can be active at once and the rate at which new sessions can be initiated.

After a request has been held for the appropriate amount of time, it enters a ready state. Processing usually begins immediately, but may be delayed if resources are not available.

The following limits affect the maximum number of requests that can be processed:

- An administrator-controlled limit on the maximum number of concurrent FTP sessions per MSP (CHILD_MAXIMUM).
- An administrator-controlled limit on the number of child processes that are guaranteed to be available for processing delete requests (GUARANTEED_DELETES).
- An administrator-controlled limit on the number of child processes that are guaranteed to be available for processing dmget(1) requests (GUARANTEED_GETS).

 A system-imposed limit of 85 FTP sessions in any 60-second period. This limit is seldom a concern because of the MSP's ability to transfer many files in one session. Because requests are grouped into batches only when resources are immediately available, GET requests (which are not normally held) are batched when resources are in short supply.

Requests are processed by forking off a child process. The parent process immediately resumes waiting for requests to arrive from the DMF daemon. The child process attempts to initiate an FTP session on the remote FTP server. If the remote machine has multiple Internet Protocol (IP) addresses, all of them are tried before giving up. If the child process cannot connect, it waits 5 minutes and tries again until it succeeds.

Once a connection is established, the child process provides any required user name, password, account, and default directory information to the remote FTP server. PUT, GET, or DELETE operations are then performed as requested by the DMF daemon. PUT, GET, or DELETE operations are not intermixed within a batch. If an individual request does not complete successfully, it does not necessarily cause other requests in the same batch to fail. Binary transfer mode is used for all data transfer.

The stored files are not verbatim copies of the user files. They are stored using the same format used to write tapes, and you can use MSP utilities such as dmatread and dmatsnf to access the data in them.

FTP MSP Activity Log

All DMF MSPs maintain log files named msplog. yyyymmdd in the MSP spool directory which, by default, is SPOOL_DIR/mspname. SPOOL_DIR is configured in the base object of the configuration file; mspname is the name of the MSP in the daemon object of the configuration file; yyyymmdd is the current year, month, and day.

The activity log shows the arrival of new requests, the successful completion of requests, failed requests, creation and deletion of child processes, and all FTP transactions. Sensitive information (passwords and account information) does not appear in the activity log. In addition, the MSP lists the contents of its internal queues in its activity log if it is given an INTERRUPT signal.

Note: Because the FTP MSP will continue to create log files files without limit, you must remove obsolete files periodically by configuring the run_remove_logs task in the configuration file, as described in "Configuring Daemon Maintenance Tasks" on page 50.

FTP MSP Messages

The MSP also recognizes and handles the following messages issued from the DMF daemon:

Message	Description
CANCEL	Issued when a previously requested action is no longer necessary, for example, when a file being migrated with a PUT request is removed. The MSP is able to cancel a request if it is being held or if it is waiting for resources. A request that has begun processing cannot be canceled and will run to normal completion.
FINISH	Issued during normal shutdown. When the MSP receives a FINISH message, it finishes all requested operations as quickly as it can and then exits.
FLUSHALL	Issued in response to the dmdidle(8) command. When the MSP receives a FLUSHALL message, it finishes all requested operations as quickly as it can.



Caution: If the remote filesystem must be restored to a previous state, inconsistencies may arise: remote files that reappear after being deleted are never removed, and remote files that disappear unexpectedly result in data loss. There is presently no way to detect these inconsistencies. You should avoid situations that require the remote filesystem to be restored to a previous state.

Disk MSP

The disk MSP (dmdskmsp) migrates data into a directory that is accessed on the current system. It uses POSIX file interfaces to open, read, write, and close files. The directory may be NFS-mounted, unless the disk MSP is configured as a disk cache manager (see "Disk MSP and Disk Cache Manager (DCM)" on page 180). 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.

Disk MSP Processing of Requests

The disk MSP is always waiting for requests to arrive from the DMF daemon, but, to improve efficiency, it holds PUT and DELETE requests briefly and groups similar requests together into a single session. No PUT request will be held longer than 60 seconds. No DELETE request will be held longer than 5 seconds. GET requests are not held. The MSP will stop holding requests if it has a large amount of work to do (more than 1024 individual files or 8 MB of data).

After a request has been held for the appropriate amount of time, it enters a ready state. Processing usually begins immediately, but may be delayed if resources are not available.

The following limits affect the maximum number of requests that can be processed:

- An administrator-controlled limit on the maximum number of concurrent operations per MSP (CHILD_MAXIMUM).
- An administrator-controlled limit on the number of child processes that are guaranteed to be available for processing delete requests (GUARANTEED_DELETES).
- An administrator-controlled limit on the number of child processes that are guaranteed to be available for processing dmget(1) requests (GUARANTEED_GETS).

Requests are processed by forking off a child process. The parent process immediately resumes waiting for requests to arrive from the DMF daemon.

PUT, GET, or DELETE operations are performed as requested by the DMF daemon. PUT, GET, or DELETE operations are not intermixed within a batch. If an individual request does not complete successfully, it does not necessarily cause other requests in the same batch to fail. Binary transfer mode is used for all data transfer.

The stored files are not verbatim copies of the user files. They are stored using the same format used to write tapes, and you can use MSP utilities such as dmatread and dmatsnf to access the data in them.

Disk MSP Activity Log

All DMF MSPs maintain log files named msplog.yyyymmdd in the MSP spool directory which, by default, is SPOOL_DIR/mspname. SPOOL_DIR is configured in the base object of the configuration file; mspname is the name of the MSP in the daemon object of the configuration file; yyyymmdd is the current year, month, and day).

The log file shows the arrival of new requests, the successful completion of requests, failed requests, and creation and deletion of child processes. In addition, the MSP lists the contents of its internal queues in its activity log if it is given an INTERRUPT signal.

Note: Because the disk MSP will continue to create log files without limit, you must remove obsolete files periodically by configuring the run_remove_logs task in the configuration file, as described in "Configuring Daemon Maintenance Tasks" on page 50

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.

dmdskvfy Command

The dmdskvfy command verifies that copies of migrated files in disk MSPs are consistent with the dmfdaemon database entries that refer to them. This command applies both to regular disk MSPs and to those running in DCM mode.

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 <code>-d</code> 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

Note: If you have a tape MSP, you must first convert it to a volume group in an LS while still on IRIX, using dmmsptols, and then convert to Linux as a second and independent step. (The tape MSP is not available in the Linux DMF release.) For more information, see the 3.0 version of the *DMF Administrator's Guide for SGI InfiniteStorage* (007-3681-008).

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 Filesystem Audit 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 LSs. For more information, see the dmdump(8) man page.

4. Set up the /etc/dmf/dmf.conf file on the Linux system. The conversion will be simpler if you name all of the FTP MSPs, disk MSPs, tape volume groups, and LSs with the same names used on IRIX. This assumes that you do not already have MSPs, LSs, 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.

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. (Optional) Sort the daemon and CAT text database records after they have been dumped to text for better overall performance of the text-record load process. (The time to sort and load will be less than the time to load unsorted text records when the number of records is in the millions.)

To sort the daemon text record file, use a command similar to the following, where *tmpdir* is a directory in a filesystem with sufficient free space for sort(1) to complete the sort:

/bin/sort -t"|" -y -T tmpdir -k 1,1 -o daemontext daemontext

To sort the CAT text record file, use a command similar to the following, where *tmpdir* is a directory in a filesystem with sufficient free space for sort(1) to complete the sort:

/bin/sort -t"|" -y -T tmpdir -k 2,2 -o cattext cattext

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

- 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 LSs. Use dmvoladm to load the VOL database for each of the LSs.
- 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 6-5 IRIX to Linux Conversion (Single LS)

In the following example, the IRIX machine has a single LS named 1s1. 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 182, 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
```

LS Error Analysis and Avoidance

The drive group component of the LS monitors tape use, analyzes failures, and usinges 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 HVFY, which results in the tape being used in a read-only mode until eventually emptied by merges or hard deletion of its files. At that time, the administrator may choose to test it and possibly replace or delete it. If it is to be returned to service, the HVFY flag should be cleared by using dmvoladm. Full details of these procedures are included in the email sent to the administrator at the time of the error.

If it is considered desirable to return a volume or drive to service earlier than defined in the DMF configuration, the appropriate command (dmvoladm, tmconfig, or ov_drive) can be safely used.

LS Drive Scheduling

When multiple 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 XFS dumps any direct tape use by the system's users; it does not prevent the LS from working properly, though it might be less than optimal.

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 87).

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.

LS Status Monitoring

You can observe the performance of the LS in two ways:

- Monitor its log file with a tool like tail -f, which allows an experienced administrator to follow the flow of events as they happen
- Use the resource watcher component, when enabled by use of the WATCHER parameter in the libraryserver configuration stanza

The resource watcher is intended to give the administrator a view of the status of an LS and some of its components. It maintains a set of text files on disk that are rewritten as events happen. These files can be found in the SPOOL_DIR/lsname/_rwname directory, where SPOOL_DIR is defined in the DMF configuration file, as are the names of the LS and resource watcher; for example, lsname and rwname. The easiest way to find the precise path is to look in the LS log file for messages like the following:

rwname.config_changed:

URL of home page is file:/dmf/spool/lsname/_rwname/lsname.html

This message is issued at DMF startup or whenever the configuration file is altered or its modification time changes; for example, by using the touch(1) command.

The SPOOL_DIR/lsname/_rwname directory contains files with names ending in .html, which are automatically refreshing HTML files. You can access these files by using a browser running on the same machine. The following example shows an LS page that contains links to drive group pages, and they in turn have links to 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 following:

- DMF version shown by dmversion(1)
- IRIX version shown by uname(1)
- Linux kernel version shown by uname(1)
- Linux distribution version shown by head /etc/*release

DMF Maintenance and Recovery

This chapter contains information for the administrative maintenance of DMF:

- "Retaining Old DMF Daemon Log Files" on page 189
- "Retaining Old DMF Daemon Journal Files" on page 189
- "Soft- and Hard-Deletes" on page 190
- "Backups and DMF" on page 191
- "Using dmfill" on page 202
- "Database Recovery" on page 202

Retaining Old DMF Daemon Log Files

The daemon generates the <code>SPOOL_DIR/daemon_name/dmdlog.yyyymmdd</code> 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 <code>SPOOL_DIR/msp_or_ls_name/msplog.yyyymmdd</code> log file, which also contains 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 50, for more information.

Retaining Old DMF Daemon Journal Files

The daemon and the LS generate journal files that are needed to recover databases in the event of filesystem damage or loss. You also configure DMF to generate backup copies of those databases on a periodic basis. You need only retain those journal files

that contain records created since the oldest database backup that you keep. In theory, you should need only one database backup copy, but most sites probably feel safer with more than one generation of database backups.

For example, if you configure DMF to generate daily database backups and retain the three most recent backup copies, then at the end of 18 July there would be backups from the 18th, 17th, and 16th. Only the journal files for those dates need be kept for recovery purposes.

To manage the journal files and the backups, configure the run_remove_journals.sh and run_copy_databases.sh tasks. These tasks automatically delete old journal files and generate backups of the databases according to a policy you set. See "Configuring Daemon Maintenance Tasks" on page 50, for more information.

Soft- and Hard-Deletes

When a file is first migrated, a bit-file identifier (BFID) is placed in the inode; this is the key into the daemon database. When a migrated file is removed, its BFID is no longer needed in the daemon database.

Initially, it would seem that you could delete daemon database entries when their files are modified or removed. However, if you actually delete the daemon database entries and then the associated filesystem is damaged, the files will be irretrievable after you restore the filesystem.

For example, assume that migrated files were located in the /x filesystem, and you configured DMF to generate a full backup of /x on Sunday as part of your site's weekly administrative procedures (the run_full_dump.sh task). Next, suppose that you removed the migrated files in /x on Monday morning and removed the corresponding daemon database entries. If a disk hardware failure occurs on Monday afternoon, you must restore the /x filesystem to as recent a state as possible. If you restore the filesystem to its state as of Sunday, the migrated files are also returned to their state as of Sunday. As migrated files, they contain the old BFID from Sunday in their inodes, and, because you removed their BFIDs from the daemon database, you cannot recall these files.

Because of the nature of the filesystem, a daemon database entry is not removed when a migrated file is modified or removed. Instead, a deleted date and time field is set in the database. This field indicates when you were finished with the database entry, except for recovery purposes; it does not prohibit the daemon from using the

database entry to recall a file. When the /x filesystem is restored in the preceding example, the migrated files have BFIDs in their inodes that point to valid database entries. If the files are later modified or removed again, the delete field is updated with this later date and time.

The term *soft-deleted* refers to a database entry that has the delete date and time set. The term *hard-deleted* refers to a file that is removed completely from the daemon database and the MSPs/LSs. You should hard-delete the older soft-deleted entries periodically; otherwise, the daemon database continues to grow in size without limit as old, unnecessary entries accumulate. Configure the run_hard_deletes.sh task to perform hard-deletes automatically. See "Configuring Daemon Maintenance Tasks" on page 50, for more information.

If you look at all of the tapes before and after a hard-delete operation, you will see that the amount of space used on some (or all) of the tapes has been reduced.

Note: Because hard-deletions normally use the same expiry times as backups, the run_hard_deletes.sh is normally run from the same task group.

Backups and DMF

This section discusses the interrelationships between DMF and backup products:

- "DMF-managed User Filesystems" on page 192
- "Storage Used by an FTP MSP or a Standard Disk MSP" on page 200
- "Filesystems Used by a Disk MSP in DCM Mode" on page 200
- "DMF's Private Filesystems" on page 201



Caution: The fact that DMF maintains copies of data on another medium does not mean that it is a backup system. The copies made by DMF may become inaccessible if there is a failure and proper backups have not been made.

In addition, although using RAID may protect you against the failure of one disk spindle, data can still be endangered by software problems, human error, or hardware failure.

Therefore, backups are essential.

DMF-managed User Filesystems

Many backup and recovery software packages make backup copies of files by opening and reading them using the standard UNIX or Linux system calls. In a user filesystem managed by DMF, this causes files that are offline to be recalled back to disk before they can be backed up. If you have a DMF-managed filesystem in which a high percentage of the files are offline, you may see a large amount of tape or other activity caused by the backup package when it initially does its backups. You should take this behavior into account when deciding whether or not to use such backup packages with filesystems managed by DMF.

Using SGI xfsdump and xfsrestore with Migrated Files

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.
- The dmmigrate(8) command lets you implement a 100% migration policy that does not interfere with customary management of space thresholds.

The xfsdump command supports the -a option specifically for DMF. If you specify the -a option, xfsdump will dump DMF dual-state (DUL) files as if they were offline (OFL) files. That is, when xfsdump detects a file that is backed up by DMF, it retains only the inode for that file because DMF already has a copy of the data itself. This dramatically reduces the amount of tape space needed to back up a filesystem and it also reduces the time taken to complete the dump, thereby minimizing the chances of it being inaccurate due to activity elsewhere in the system. An added advantage of using -a is that files that are actively being recalled will still be backed up correctly by xfsdump because it does not need to copy the file's data bytes to tape.

You can also use dmmigrate to force data copies held only in a 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 backup utility. As the cache is not a DMF-managed filesystem, you are not restricted to using xfsdump.

Most installations periodically do a full (level 0) dump of filesystems. Incremental dumps (levels 1 through 9) are done between full dumps; these may happen once per

day or several times per day. You can continue this practice after DMF is enabled. When a file is migrated (or recalled), the inode change time is updated. The inode change time ensures that the file gets dumped at the time of the next incremental dump.

To automatically manage dump tapes, DMF includes configurable administrative scripts called run_full_dump.sh and run_partial_dump.sh, which employ xfsdump. Both of these tasks are simple wrappers around a script called do_xfsdump.sh, which performs the following actions:

- (optional) Migrates all eligible files to dual-state
- (optional) Copies all eligible DCM files on a DCM system to dual-residency state
- Performs a database snapshot using dmsnap
- Backs up the directory containing that snapshot
- Backs up other filesystems
- · After a successful full backup, frees up old backup tapes for future reuse

DMF also supports a matching wrapper around xfsrestore named dmxfsrestore to be used when restoring files that were dumped by these backup scripts. See the dmxfsrestore(8) man page for more information on running the command.

You can configure tasks in the dump_tasks object to automatically do full and incremental dumps of the DMF-managed filesystems. See "Configuring Daemon Maintenance Tasks" on page 50, for more information.

A typical dump_tasks stanza might look like the following:

```
define dump_tasks
        TYPE
                                taskgroup
        RUN_TASK
                                $ADMINDIR/run_full_dump.sh on sunday at 03:00
        RUN_TASK
                                $ADMINDIR/run_partial_dump.sh \
                                   on monday tuesday wednesday \
                                   thursday friday saturday at 03:00
                                $ADMINDIR/run_hard_deletes.sh at 23:00
        RUN_TASK
        DUMP_TAPES
                                HOME_DIR/tapes
        DUMP_RETENTION
                                4w
        DUMP_DEVICE
                                dq1
        DUMP_MIGRATE_FIRST
                                yes
                                           # Only if you run a DCM
        DUMP_FLUSH_DCM_FIRST
                                yes
        DUMP_INVENTORY_COPY
                                /save/dump_inventory
enddef
```

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

For more information about parameters, see Chapter 2, "Configuring DMF" on page 29.

Sites using OpenVault can add new backup tapes by using dmov_makecarts and/or dmov_loadtapes by providing the name of the task group as a parameter. Sites using TMF do not need any special steps to add new tapes, as TMF does not record details of which tapes are available to it.

Recycling old backup tapes is performed automatically after the successful completion of a full dump. In certain situations, such as running out of dump tapes, this pruning must be done manually by running dmxfsprune.

Ensuring Accuracy with xfsdump

The xfsdump program is written such that it assumes dumps will only be taken within filesystems that are not actively changing. xfsdump cannot detect that a file

has changed while it is being dumped, so if a user should modify a file while it is being read by xfsdump, it is possible for the backup copy of the file to be inaccurate.

To ensure that all file backup copies are accurate, perform the following steps when using xfsdump to dump files within a DMF filesystem:

- 1. Make sure that there is no user activity within the filesystem.
- 2. Ensure that DMF is not actively migrating files within the filesystem.
- 3. Run xfsdump, preferably with the -a option.

Dumping and Restoring Files without the DMF Scripts

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

- The DMF scripts use xfsdump with the -a option to dump only data not backed up by DMF. You may also wish to consider using the -a option on xfsdump when dumping DMF filesystems manually.
- Do not use the -A option on either xfsdump or xfsrestore. The -A option avoids dumping or restoring extended attribute information. DMF information is stored within files as extended attributes, so if you do use -A, migrated files restored from those dump tapes will not be recallable by DMF.
- When restoring migrated files using xfsrestore, you must specify the -D option in order to guarantee that all DMF-related information is correctly restored.
- If you use the Tape Management Facility (TMF) to mount tapes for use by xfsdump, be aware that xfsdump will not detect the fact that the device is a tape, and will behave as if the dump is instead being written to a regular disk file. This means that xfsdump will not be able to append new dumps to the end of an existing tape. It also means that if xfsdump encounters end-of-tape, it will abort the backup rather than prompting for additional volumes. You must ensure that you specify enough volumes using the tmmnt -v option before beginning the dump in order to guarantee that xfsdump will not encounter end-of-tape.

Filesystem Consistency with xfsrestore

When you restore files, you might be restoring some inodes containing BFIDs that were soft-deleted since the time the dump was taken. (For information about soft-deletes, see "Soft- and Hard-Deletes" on page 190.) 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 losing user data. The dmhdelete(8) program responsible for removing unused database entries always first scans all DMF-managed filesystems to make sure that there are no remaining files which reference the database entries it is about to remove. It is able to detect either of these inconsistencies and will not remove the database entries if inconsistencies are found.

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

Using DMF-aware Third-Party Backup Packages

Some third-party backup packages can use a DMF library to perform backups in a DMF-aware manner. When the DMF-aware feature is enabled, these packages will not cause offline (OFL) files to be recalled during a backup. Dual-state (DUL) files will be dumped as if they were offline, which will reduce the time and space needed for a backup.

To use a DMF-aware third-party backup package to back up DMF filesystems, do the following:

- 1. Configure the backup package to include the DMF filesystems in the backups.
- 2. Enable the DMF-aware feature on those filesystems.

For more information about third-party backup packages, see Appendix F, "Third-Party Backup Package Configuration" on page 287.

DMF provides a script called do_predump.sh that is meant to be run just prior to a backup of the DMF filesystems using a third-party backup package. The do_predump.sh script does the following:

- (Optional) Migrates all eligible files to dual-state
- (Optional on a DCM system) Copies all eligible DCM files to dual-residency state
- (Optional) Performs a database snapshot using dmsnap

To use do_predump.sh, do the following:

- 1. Configure the backup package to run do_predump.sh as the pre-backup command. For details, see the application-specific information in Appendix F, "Third-Party Backup Package Configuration" on page 287.
- 2. Define a task group in the dmf.conf file that is referred to by the dmdaemon object. In the supplied configurations, this task group is called dump_tasks.

The parameters do_predump.sh uses are as follows:

Specifies a path to where a snapshot of the DMF databases will be placed. The backup package should be configured to backup this directory. If not specified, no snapshot will be taken.
If set to YES, specifies that dmmigrate is run to ensure that all non-dual-resident files in the DCM-mode MSP caches are migrated to tape. If DUMP_MIGRATE_FIRST is also enabled, that is processed first.
If DUMP_MIGRATE_FIRST is enabled, specifies the DMF-managed filesystems on which to run the dmmigrate command. The default value for this parameter is all DMF-managed filesystems.
If set to YES, specifies that dmmigrate is run to ensure that all migratable files in the DMF-managed filesystems are migrated, thus reducing the number of tapes needed for the dump and making it run much faster.

Because hard-deletions normally use the same expiry time as backups, run_hard_deletes.sh is normally run from the same task group. The DUMP_RETENTION parameter should match the retention policy of the backup package.

When using a third-party backup package, a typical dump_tasks stanza might look like the following:

```
define dump_tasks

TYPE taskgroup

RUN_TASK $ADMINDIR/run_hard_deletes.sh at 23:00

DUMP_RETENTION 4w # match backup package's policy

DUMP_MIGRATE_FIRST yes

DUMP_FLUSH_DCM_FIRST yes # only if running a DCM

DUMP_DATABASE_COPY /path/to/db_snapshot

enddef
```

Note: Backups and restores must be run from the DMF server.

Only root can perform backups and restores. Although some third-party backup packages normally allow unprivileged users to restore their own files, unprivileged users cannot restore their own files from a DMF filesystem because doing so requires root privilege to set the DMF attribute.

Files backed up from a DMF filesystem should only be restored to a DMF filesystem. Otherwise, files that are offline (or treated as such) will not be recallable.

Using XVM Snapshots and DMF

You can use the xfsdump facility to backup XVM snapshots of a DMF-managed user filesystem. Note the following:

- XVM snapshots of DMF-managed filesystems should not be added to the DMF configuration file.
- You should not attempt to migrate or recall files from an XVM snapshot of a DMF filesystem.
- You can only restore DMF offline, partial, and unmigrating files by using xfsdump and xfsrestore. You cannot use a previously taken snapshot of a DMF filesystem to directly copy any of these file types back into the live filesystem. You may copy migrating or dual-state files back into the live filesystem (to DMF, they will appear to be new files).

For more information about XVM snapshots, see the XVM Volume Manager Administrator's Guide.

Optimizing Backups of Filesystems

You can greatly reduce the amount of time it takes to back up filesystems by configuring DMF to migrate all files. Do the following:

- Set the DUMP_MIGRATE_FIRST parameter to yes, which specifies that the dmmigrate command is run before the dumps are done to ensure that all migratable files in the DMF-managed user filesystems are migrated.
- Execute one of the following scripts:
 - run_full_dump to perform a full backup of the filesystems
 - run_partial_dump to perform a partial backup of the filesystems

For more information, see Chapter 2, "Configuring DMF" on page 29.

Migrating all files before performing a backup has the following benefits:

- The backup image will be smaller because it contains just the metadata information, not the file data itself
- The backup will complete more quickly because:
 - It is reading just the metadata
 - There is less time spent performing random disk seeks to back up the data of unmigrated files

For any files that you want to remain permanently on disk (that is, permanently dual-state), you can assign a negative priority weight to those files, which would leave the files on disk. The result is that when the filesystem is filled up, DMF will never free the blocks for these files. The files therefore are always dual-state, ready to be used. When the filesystem is backed up, the backup facility will recognize that they are dual-state and therefore back them up as offline. The net effect is that there is no file data in the backup at all for these files, just their inodes, while keeping the files always available. In the case of millions of small files, this speed-up of the backup process can be dramatic. For example, for a filesystem with a large number of small files (files of up to 64 KB), you could assign the following AGE_WEIGHT value:

AGE_WEIGHT -1 0 when space < 64k

Be aware of the following:

- For extremely small files (under a few hundred bytes), the disk space required for DMF database entries may exceed the size of the original file. For extremely large numbers of such files, this issue should be considered.
- The space value in a when clause, as used above, refers to the space the file occupies on disk, which for sparse files may actually be smaller than the size of the file as shown by 1s -1. The space value will be rounded upward to a multiple of the disk blocksize defined by mkfs(8); the default is 4096 bytes. For example, attempting to discriminate between files above or below 1000 bytes based on their space value is futile because all non-empty files will have a space value that is a multiple of (typically) 4096 bytes.

If you use negative weights with AGE_WEIGHT or SPACE_WEIGHT, DMF automatic migration will never free the space for these files but a user can still do a dmput -r on them to manually free the space.

However, if you do not want files to migrate for any reason, then you must continue to use the SELECT_VG method despite the slower and larger backups.

Storage Used by an FTP MSP or a Standard Disk MSP

If you are depending on an FTP MSP or a standard disk MSP to provide copies of your offline files in order to safeguard your data, then they should also be backed up.

If you use them just to hold extra copies for convenience or to speed data access, they need not be backed up. But you should consider how you would handle their loss. You would probably need to remove references to lost copies from the DMF daemon's database, using dmdadm, which can only be done when the daemon is not running.

Filesystems Used by a Disk MSP in DCM Mode

A DCM differs from a conventional disk MSP in that it uses DMAPI to manage the files. It will not operate properly if the files are reloaded by a package that cannot also restore the DMAPI information associated with each file.

Note: For simplicity, this discussion assumes that the site wishes to keep two copies of migrated files at all times to guard against media problems. (Keeping only one copy is considered risky, and keeping more than two copies is frequently impractical.)

The DCM can have one of the following configurations:

- A DCM may be holding an extra copy of files in addition to the normal number of tape-based copies. That is, after the initial migration has completed, there will be two tape copies and a third in the cache. The DCM may easily remove this third copy from the cache after some period of time, just leaving two tape-based copies. With this configuration, there is normally no need to back up the cache filesystem.
- The initial migration could result in one cache copy and one on tape. Later on, when the cache has to be flushed, a second tape copy is written by the DCM before the cache-resident one is deleted. If the file is hard-deleted before the cache flushes, the second tape copy will never be made, thereby saving time and tape. The tradeoff is that cache-flushing is slower and the cache filesystem should be backed up; otherwise a tape media problem in conjunction with a disk failure would result in data loss. With this configuration, the cache filesystem should be backed up. Otherwise, the loss of the cache disk could leave you with just one copy of data on tape. This is considered to be risky.

For both configurations, any backups require the use of a DMF-aware backup package (as listed in Appendix F, "Third-Party Backup Package Configuration" on page 287) to back up the cache.

To use do_xfsdump.sh to backup any of these filesystems, include the pathname of its mountpoint in the DUMP_FILE_SYSTEMS parameter.

DMF's Private Filesystems

The following DMF private filesystems do not require a DMF-aware backup package:

- HOME DIR
- JOURNAL_DIR
- SPOOL_DIR
- TMP_DIR
- CACHE_DIR
- MOVE_FS

Care should be taken when backing up the databases in *HOME_DIR* if there is any DMF activity going on while the backup is underway, due to the risk of making the copy of the database while it is being updated. A safe technique is to take a snapshot

of the databases with dmsnap and back up the snapshot. The do_xfsdump.sh script does this automatically.

The journal files in JOURNAL_DIR should also be backed up if you keep older snapshots of the databases that may have to be reloaded and brought up-to-date with dmdbrecover. Preferably, journals should be backed up when DMF activity (apart from recalls) is minimal. The do_xfsdump parameters DUMP_MIGRATE_FIRST and DUMP_FLUSH_DCM_FIRST help achieve this by processing any queued up migration requests immediately before starting the backup.

SPOOL_DIR contains log files that may be of use for problem diagnosis, as well as history files controlling things like tape error recovery and reporting scripts. The loss of these files will not endanger user data, although DMF may act a little differently for a while until it reestablishes them. Back up SPOOL_DIR if you can.

The TMP_DIR, CACHE_DIR, and MOVE_FS filesystems do not require backup.

To use do_xfsdump.sh to backup any of these filesystems, simply include the pathnames of their mountpoints in the DUMP_FILE_SYSTEMS parameter.

Using dmfill

The dmfill(8) command allows you to fill a restored filesystem to a specified capacity by recalling offline files. When you execute xfsdump -a, only inodes are dumped for all files that have been migrated (including dual-state files). Therefore, when the filesystem is restored, only the inodes are restored, not the data. You can use dmfill in conjunction with xfsrestore to restore a corrupted filesystem to a previously valid state. dmfill recalls migrated files in the reverse order of migration until the requested fill percentage is reached or until there are no more migrated files left to recall on this filesystem.

Database Recovery

The basic strategy for recovering a lost or damaged DMF database is to recreate it by applying journal records to a backup copy of the database. For this reason it is essential that 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" on page 203
- "Database Recovery Procedures" on page 203

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 50, 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 LS has two databases in the HOME_DIR/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 libsrv_db.dbd.

Database Recovery Procedures

The DMF daemon and LS write journal file records for every database transaction. These files contain binary records that cannot be edited by normal methods and that must be applied to an existing database with the dmdbrecover(8) command. The following procedure explains how to recover the daemon database.



Warning: If you are running on multiple LSs, always ensure that you have the correct journals restored in the correct directories. Recovering a database with incorrect journals can cause irrecoverable problems.

Procedure 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:
 - # /etc/init.d/dmf stop
- If you have configured the run_copy_databases task, restore the files from the directory with the most recent copy of the databases that were in HOME_DIR to HOME_DIR/daemon or HOME_DIR/LS_NAME.
- 3. If you have **not** configured the run_copy_databases task, reload an old version of the daemon or LS database. Typically, these will be from the most recent dump tapes of your filesystem.
- Ensure that the default JOURNAL_DIR/daemon_name (or JOURNAL_DIR/ls_name)
 directory contains all of the time-ordered journal files since the last update of the
 older database.

For the daemon, the files are named dmd_db.yyyymmdd[.hhmmss].

For the LS, the journal files are named libsrv_db.yyyymmdd[.hhmmss].

5. Use dmdbrecover to update the old database with the journal entries from journal files identified in step 4.

Example 7-1 Database Recovery Example

Suppose that the filesystem containing *HOME_DIR* was destroyed on February 1, 2004, and that your most recent backup copy of the daemon and LS databases is from January 28, 2004. To recover the database, you would do the following:

- 1. Stop DMF:
 - # /etc/init.d/dmfstop

2. Ensure that <code>JOURNAL_DIR/daemon_name</code> (or <code>JOURNAL_DIR/ls_name</code>) contains the following journal files (one or more for each day):

```
JOURNAL_DIR/daemon_name
```

```
dmd_db.20040128.235959
dmd_db.20040129.235959
dmd_db.20040130.235959
dmd_db.20040131.235959
dmd_db.20040201
```

JOURNAL_DIR/ls_name

```
libsrv_db.20040128.235959
libsrv_db.20040129.235959
libsrv_db.20040130.235959
libsrv_db.20040131.235959
libsrv_db.20040201
```

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

```
HOME_DIR/daemon_name
```

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

HOME_DIR/ls_name

```
tpcrdm.dat
tpcrdm.key1.keys
tpcrdm.key2.keys
tpvrdm.dat
tpcrdm.vsn.keys
```

4. Update the database files created in step 3 by using the following commands:

```
dmdbrecover -n daemon_name dmd_db
dmdbrecover -n ls_name libsrv_db
```

Messages

This appendix describes the format and interpretation of messages reported by dmcatadm(8) and dmvoladm(8). It contains the following sections:

- "Message Format" on page 207
- "dmcatadm Message Interpretation" on page 209
- "dmvoladm Message Interpretation" on page 211

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</pre>
```

```
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</pre>
```

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
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:

is less than 0.

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.
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

007-3681-019

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.)
<pre>for vsn DMF001 chunk 77 filesize != file size in daemon entry (nnn)</pre>	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.

entry for volume group name missing from daemon db

No daemon entry was found for the entry in the CAT database.

for volgrp name; no chunk for bytes nnn - nnn

There is no chunk that contains the specified bytes of the file.

dmvoladm Message Interpretation

chunksleft != number of cat chunks (256)

eotchunk < largest chunk in cat (443)</pre>

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.

the ta

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.

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.

007–3681–019 211

The zone number of the EOT eotzone < largest zone in cat (77) 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 When a volume is empty, the hfull, volume is empty but hsparse is on and hsparse hold flags should be volume is empty but datawritten != 0 When the hfree hold flag is cleared, volume is empty but eotpos != 1/0 the datawritten field is set to 0, the volume is empty but eotchunk != 1 eotpos field is set to 1/0, and the eotchunk is set to 1. The entry is inconsistent and should be checked. volume is not empty but hfree is on When a volume contains data, the hfree hold flag must be off. volume is not empty and version is n but hfull is off Tapes containing data with a version value of less than 4 must have hfull set, because the LS cannot append to the tape. volume is not empty and version is n but eotpos != 2/0Tapes imported from the old MSP only have one zone of data, so eotpos must be 2/0. zonesize is too small The zonesize field for the tape is an impossible number.

DMF User Library libdmfusr.so

The subroutines that constitute the DMF user command application program interface (API) are available to user-written programs by linking to the DMF user library, libdmfusr.so. Sites can design and write their own custom DMF user commands, which eliminates the need to use wrapper scripts around the DMF user commands.

This appendix discusses the following:

- "Overview of the Distributed Command Feature and libdmfusr.so"
- "Considerations for IRIX" on page 216
- "libdmfusr.so Library Versioning" on page 216
- "libdmfusr.so.2 Data Types" on page 217
- "User-Accessible API Subroutines for libdmfusr.so.2" on page 232

Overview of the Distributed Command Feature and libdmfusr.so

The distributed command feature allows DMF commands to execute on a host other than the host on which the DMF daemon is running. (This feature was first made available with DMF 2.7.) A host that imports DMF-managed filesystems from the DMF daemon host machine can execute the DMF commands locally as defined in "Hardware and Software Requirements" on page 4. The distributed command feature requires topmux (RFC 1078).

The DMF user commands communicate with a process named dmusrcmd, which is executed as setuid root. dmusrcmd performs validity checks and communicates with the DMF daemon. (In releases prior to DMF 2.7, user commands communicated directly with the DMF daemon and were installed as setuid root processes.)

007–3681–019 213

In order for the DMF user commands to communicate in an efficient and consistent manner with the dmusrcmd process, they must access the DMF user library, libdmfusr.so[.n], which is installed in the following directories:

• IRIX platforms:

/usr/lib
/usr/lib32
/usr/lib64

• All other platforms:

/usr/lib

Each of the DMF user commands is linked to the library for its protocol-based communications. (The DMF user library became a versioned shared-object library in DMF 3.1. See "libdmfusr.so Library Versioning" on page 216 for more information on accessing the correct version of libdmfusr.so.)

The underlying design of the API calls for the user command to make contact with a dmusrcmd process by creating an opaque context object via a call to the API. This context is then used as a parameter on each function API call (put, get, fullstat, or copy). The context is used by each API subroutine to perform the requested operation and to correctly return the results of the operation to the command.

In addition to the library, the libdmfusr.H, libdmfcom.H, and dmu_err.h header files are provided. These files are required for sites to effectively create their own commands. All header files are installed in /usr/include/dmf. The libdmf* header files contain all of the object and function prototype definitions required by the API subroutine calls. The dmu_err.h file contains all of the API error code definitions. Along with each error code definition is a text string that is associated with each of the error codes. This text string is the same message that is generated automatically when the error occurs as part of the DmuErrInfo_t object (see "DmuErrInfo_t" on page 226). The text string is included in the file as informational only, and is not accessible by a program that includes dmu_err.h.

Each type of function request (put, get, fullstat, or copy) can be made via a synchronous or an asynchronous API subroutine call:

• Synchronous subroutine calls do not return to the caller until the request has completed, either successfully or unsuccessfully. These synchronous subroutines return an error object to the caller that can be processed to determine the success or failure of the call. If an application is making more than one call, these calls

will usually perform less efficiently than their asynchronous counterparts because of the serial nature of their activity.

• Asynchronous subroutine calls return immediately to the caller. The return codes of these asynchronous subroutines indicate whether the request was successfully forwarded to dmusrcmd for processing. A successful return allows the calling program to continue its own processing in parallel with the processing being performed by dmusrcmd (or the DMF daemon) to complete the request. If the request was successfully forwarded, a request ID that is unique within the scope of the opaque context is returned to the caller. It is the responsibility of the caller to associate the request ID with the correct completion object (described in "DmuCompletion_t" on page 224) to determine the eventual result of the original request.

There are several API subroutine calls for processing asynchronous request completion objects. The user can choose to do any of the following:

- Be notified when all requests have completed without processing the return status of each request.
- Process the return status of each request in the order in which they complete.
- Wait synchronously on an individual asynchronous request's completion by specifying the request ID on which to wait. By using this method, each request return status can be processed in the order in which it was sent, known as request ID order.

The API includes well-defined protocols that it uses to communicate with the dmusrcmd process. Because these protocols make use of the pthreads(5) mechanism, any user application program making use of the API via libdmfusr.so must also link to the libpthread.so shared object library via one of the following:

- -lpthread compiler option using cc(1) or CC(1)
- -lpthread loader option using ld(1) or rld(1)

In many cases, the API subroutines pass the address of an object back to the caller by setting a ** pointer accordingly. If errors occur and the subroutine is unable to complete its task, the address returned may be NULL. It is up to the caller to check the validity of an object's address before using it in order to avoid causing a SIGSEGV fault in the application program.

Considerations for IRIX

The DMF user library for each IRIX platform (lib, lib32, and lib64) was compiled using a MIPSpro compiler. Compiling user applications that call DMF user library API subroutines with compilers other than MIPSpro compilers may result in incompatibilities causing load-time or run-time errors.

libdmfusr.so Library Versioning

DMF 3.1 introduced a new version of the DMF user library. This new version is not compatible with the previous library nor with applications that were written and linked with the previous library. To allow the use of older applications after installing DMF 3.2 and to facilitate upgrading older applications, DMF 3.2 provides both the old version and the new version and introduces a linking mechanism.

When an application is created and linked with a shared object, the name of the actual library that the application is ultimately linked with is stored in the executable file and used at execution time to find a library of the same name for dynamic linking. In previous releases, the library was named libdmfusr.so. Therefore, all existing DMF commands and site-developed applications that use the library contain the filename libdmfusr.so in the executable for linking with the library at execution time.

A common practice when creating a new version of a library is to add the suffix n to the library name, where n is an ever-increasing integer that refers to the current version number.

In previous releases of DMF, the library named libdmfusr.so was an actual library, rather than a link to a library. DMF 3.2 provides the old library (renamed libdmfusr.so.1) and the new library (named libdmfusr.so.2). All DMF 3.2 user commands (such as dmput) were created and linked with libdmfusr.so.2 and their executables contain the filename libdmfusr.so.2 for linking with the library.

The libdmfusr.so.1 library is identical to the libdmfusr.so library shipped prior to DMF 3.1. The DMF 3.2 installation process will install a link named libdmfusr.so that will point to libdmfusr.so.2. If needed, you can change the link to point to libdmfusr.so.1 in order to satisfy linking for executables built with a pre-DMF 3.1 libdmfusr.so.

The locations of the libraries and the link have not changed from previous releases (see "Overview of the Distributed Command Feature and libdmfusr.so" on page 213).

The new libdmfusr.so link provides the following advantages:

- You can use the default setting, which does not require any knowledge about the latest version of the library. When developing new site applications using the library, the non-version-specific ld option -ldmfusr will result in the loader following the link and using the new version of the library, libdmfusr.so.2. The resulting applications will contain the name libdmfusr.so.2 in their executable files for dynamic loading.
- You can reset the link to point to libdmfusr.so.1, which allows existing site-developed applications to continue to work with the older version of the library. This will not affect any of the DMF user commands because they contain the name of the new library and make no use of the link at execution time. When an older application executes, if filename libdmfusr.so is encountered by the loader and the link points to libdmfusr.so.1, the application will continue to work exactly as it did before the DMF 3.2 installation.

The two uses of the link as described above are mutually exclusive of each other. Take care when using the link to enable older applications to run with the old library while at the same time developing new applications using the new library. If the link points to libdmfusr.so.1 and -ldmfusr is used to create a new application, the older version of the library will be found and the resulting executable will contain the filename libdmfusr.so.1 for use at execution time. If older applications are required to run correctly while new applications are being developed, you must use specific loader command options to ensure that the new applications are linked with the latest library. This can be done by including the specific library name, such as libdmfusr.so.2, on the ld or cc command instead of the generic library specification -ldmfusr.

libdmfusr.so.2 Data Types

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

All of the data types defined in this section are C++ objects, and all have constructors and destructors. Many have copy constructors and some have operator override functions defined. Please refer to the appropriate .H header file to see what C++ functions are defined for each object in addition to the member functions described in this section.

DmuAllErrors_t

The DmuAllErrors_t object provides the caller with as much information regarding errors as is practical. The complex nature of the API and its communications allows for many types of errors and several locations (processes) in which they can occur. For example, a request might fail in the API, in the dmusrcmd process, or in the DMF daemon.

The public member fields and functions of this class are as follows:

entry Specifies a read-only pointer allowing access to all

DmuErrInfo_t entries in the DmuAllErrors_t

internal array.

numErrors() Returns the number of DmuErrInfo_t entries in the

DmuAllErrors_t internal array.

resetErrors() Clears the DmuAllErrors_t internal array.

Following is an example using a DmuAllErrors_t object.

Note: The following code is a guideline. It may refer to elements of a DmuAllErrors_t structure that are not defined in your installed version of libdmfcom.H.

```
report_errors(DmuAllErrors_t *errs)
                        i;
       int
        if (!errs) {
                return;
        for (i = 0; i < errs->numErrors(); i++) {
                fprintf(stdout, "group '%s' errcode '%d' who '%s' "
                        "severity '%s' position '%s' host '%s' message '%s'\n",
                        errs->entry[i].group ? errs->entry[i].group : "NULL",
                        errs->entry[i].errcode,
                        DmuLogGetErrWhoImage(errs->entry[i].errwho),
                        DmuLogGetSeverityImage(errs->entry[i].severity),
                        errs->entry[i].position ? errs->entry[i].position : "NULL",
                        errs->entry[i].host ? errs->entry[i].host : "NULL",
                        errs->entry[i].message ? errs->entry[i].message : "NULL");
       }
}
```

DmuAttr t

The DmuAttr_t object defines the DMF attribute for a DMF-managed file.

The public member fields and functions of this class are as follows:

bfid	Specifies a DmuBfid_t object (defined in libdmfcom.H) that defines the file's bitfile-ID (bfid).		
fsys	Specifies a DmuFileIoMethod_t object (defined in libdmfcom.H) that defines the file's filesystem type.		
version	Specifies a DmuFileIoVersion_t object (defined in libdmfcom.H) that defines the filesystem version.		
dmstate	Specifies a dmu_state_t object that defines the file state. Valid states are:		
	DMU_ST_REGULAR	Regular	
	DMU_ST_MIGRATING	Migrating	
	DMU_ST_DUALSTATE	Dual-state	
	DMU_ST_OFFLINE	Offline	

007–3681–019 219

DMU_ST_UNMIGRATING Unmigrating

DMU_ST_NOMIGR No migration allowed

dmflags Specifies an integer defining a file's DMAPI flags.

Currently unused.

sitetag Defines the file site tag value. See dmtag(1).

regbuf Specifies a DmuFullRegbuf_t object that defines the

file full region information. See "DmuFullRegbuf_t"

on page 228.

DmuByteRange_t

The DmuByteRange_t object defines a range of bytes that are to be associated with a put or get request.

The public member fields and functions of this class are as follows:

start_off Starting offset in bytes of the range in the file.
end_off Ending offset in bytes of the range in the file.

Nonnegative values for start_off or end_off indicate an offset from the beginning of the file. The first byte in the file has offset 0. Negative values may be used to indicate an offset from the end of the file. The value -1 indicates the last byte in the file, -2 is the next-to-last byte, and so on. The range is inclusive, so if start_off has a value of 2 and end_off has a value of 2, it indicates a range of one byte.

DmuByteRanges_t

The DmuByteRanges_t object defines a set of DmuByteRange_t objects that are to be associated with a put or get request.

The public member fields and functions of this class are as follows:

rounding

Specifies the rounding method to be used to validate range addresses. Only DMU_RND_NONE is valid.

entry

Specifies a read-only pointer allowing access to all DmuByteRange_t entries in the DmuByteRanges_t internal array.

numByteRanges()

Returns the number of DmuByteRange_t objects contained in the entry array.

resetByteRanges()

Resets the number of DmuByteRange_t objects in the array to zero.

setByteRange()

Adds a new range. If the range being added overlaps or is adjacent to an existing range in the array, the items may be coalesced. It is expected that the starting offset not be closer to the end-of-file than the ending offset. For example, a starting offset of 5 and an ending offset of 4 is invalid, and the setByteRange() function may not add it to the array. The setByteRange() function cannot determine the validity of some ranges, however, and may add ranges that the put or get request will later ignore.

fromByteRangesImage()

Converts a string that represents a byte range and adds it to the DmuByteRanges_t object. Strings that represent byte ranges are described on the dmput man page.

Note: In a string representing a byte range, -0 represents the last byte in the file, while in a DmuByteRange_t object, -1 represents the last byte in the file.

For example, suppose byteranges is declared as the following:

```
DmuByteRanges_t byteranges;
```

Then each of the following statements will add the DmuByteRange_t object that covers the entire file:

```
\label{lem:byteranges.setByteRange(0,-1);} byteranges.fromByteRangesImage("0:-0" ,&errstr);
```

007–3681–019 221

If the byte range overlaps or is adjacent to an existing range in the array, the items may be coalesced.

clearByteRange

Clears the specified byte range in the <code>DmuByteRanges_t</code> object. The <code>clearByteRange()</code> routine is restricted in how it handles negative offsets, both in the <code>DmuByteRange_t</code> members of the <code>DmuByteRanges_t</code> class and in its parameters. The following items give the details of these restrictions. In the following items, <code>start</code> and <code>end</code> are the parameters to the <code>clearByteRange()</code> routine, using the following format:

clearByteRange(start,end)

- If *start* and *end* exactly match a DmuByteRange_t entry, then that entry will be cleared. This includes negative numbers.
- If start is 0 and end is -1, all DmuByteRange_t entries will be cleared. resetByteRanges() is the preferred method for clearing all ranges.
- If *start* is positive and *end* is -1, then:
 - All DmuByteRange_t entrys that have a positive start_off value greater than or equal to *start* will be cleared
 - All DmuByteRange_t entrys that have a positive start_off value that is less than *start* and an end_off value of -1 will be changed to have an end_off value of *start-1* (that is, *start* minus 1). For example, if DmuByteRanges_t has a single range, 3:-1, then clearByteRange(4,-1) will leave a single range, 3:3.
 - All DmuByteRange_t entrys that have a positive start_off value that is less than *start* and an end_off value that is greater than *start* will be changed to have an end_off value of *start-1*. For example, if DmuByteRanges_t has a single range 3:9, then clearByteRange(4,-1) will leave a single range 3:3.
- If start and end are both positive and a DmuByteRange_t entry
 has positive start_off and end_off values, then the range
 specified by start and end is cleared from the DmuByteRange_t.

• If start, end, and the start_off and end_off values of a DmuByteRange_t are all negative, the range specified is cleared from DmuByteRange_t.

You can create a valid DmuByteRanges_t object using the default constructor with or without the new operator, depending on the need. For example:

```
DmuByteRanges_t ranges;

DmuByteRanges_t *ranges = new DmuByteRanges_t;
```

The following example creates a DmuByteRanges_t named byteranges, adds a DmuByteRange_t to it, then prints the entry to stdout:

```
DmuByteRanges_t byteranges;
                i;
byteranges.rounding = DMU_RND_NONE;
byteranges.setByteRange(0, 4095); /* specifies the first 4096 bytes in the file */
for (i = 0; i < byteranges.numByteRanges(); i++) {</pre>
        fprintf(stdout, "Starting offset %lld, ending offset %lld\n",
                byteranges.entry[i].start_off,
                byteranges.entry[i].end_off);
}
                       The output to stdout would be as follows:
                       starting offset 0, ending offset 4095
                       The following example creates a DmuByteRanges t named b, adds a
                       DmuByteRange_t to it, then clears a byte range:
DmuByteRanges_t b;
int i;
b.setByteRange(0,40960);
b.clearByteRange(4096,8191);
printf("Num byte ranges %d\n",b.numByteRanges());
for (i = 0; i < b.numByteRanges(); i++)</pre>
        printf("%lld %lld\n",b.entry[i].start_off, b.entry[i].end_off);
                       The output to stdout would be as follows:
                      Num byte ranges 2
                       0 4095
```

8192 40960

007-3681-019 223

Note: The toByteRangesImage() member function is not yet supported.

DmuCompletion_t

The DmuCompletion_t object is returned by one of the API request completion subroutines (see "Request Completion Subroutines" on page 247) with the results of an asynchronous request.

The public member fields and functions of this class are as follows:

request_id Associates the completion object with an asynchronous

request that was previously issued. This value coincides with the request ID value that any of the

asynchronous subroutines return to the user.

request_type Specifies the type of the original request.

reply_code Contains the overall success or failure status of the

request. If this value is DmuNoError, the request was successful. If not, the allerrors field should be checked for the appropriate error information.

ureq_data Specifies a pointer to user request-type specific data.

For a fullstat user request, this will point to a DmuFullstat_t object. This field has no meaning for

put, get, or copy user requests.

fhandle Specifies the file handle of the file associated with the

request.

DmuCopyRange t

The DmuCopyRange_t object defines a range of bytes that are to be associated with a copy request.

The public member fields and functions of this class are as follows:

src_offset Specifies the starting offset in bytes of the range in the

source file to be copied.

src_length Specifies the length in bytes of the range to be copied.

dst_offset

Specifies the starting offset in bytes in the destination file to which the copy is sent.

DmuCopyRanges_t

The DmuCopyRanges_t class defines an array of DmuCopyRange_t objects that are to be associated with a copy request.

The public member fields and functions of this class are as follows:

rounding Specifies the rounding method to be used to validate range addresses. Only DMU_RND_NONE is supported.

entry Specifies a read-only pointer allowing access to all the

DmuCopyRange_t entries in the array.

numCopyRanges()
Returns the number of DmuCopyRange_t objects

contained in the entry array. Only a single range is

supported.

setCopyRange Adds a new DmuCopyRange_t object to the array.

resetCopyRanges() Resets the number of DmuCopyRange_t objects in the

array to zero.

Example: Create a DmuCopyRanges_t, add a DmuCopyRange_t to it, then print the entry to stdout:

DmuErrHandler f

The DmuErrHandler_f object defines a user-specified error handling subroutine. Many of the API subroutines may result in the receipt of error information from the dmusrcmd process or the DMF daemon in the processing of the request. As these errors are received, they are formatted into a DmuErrInfo_t object (see "DmuErrInfo_t" on page 226) and are generally returned to the caller either via a calling parameter or as part of a DmuCompletion_t object.

In addition, however, if the error occurs in the course of processing internal protocol messages, the DmuErrInfo_t object can also be passed into the DmuErrHandler_f that the caller defined when the opaque context was created.

As part of the DmuCreateContext() API subroutine call, the caller can specify a site-defined DmuErrHandler_f subroutine or the caller can use one of the following API-supplied subroutines:

DmuDefErrHandler Outputs the severity of error and the message

associated with the error to stderr.

DmuNullErrHandler Does nothing with the error.

DmuErrInfo_t

The DmuErrInfo_t object contains the information about a single error occurrence.

The public member fields and functions of this class are as follows:

group Defines the originator of the error:

sgi_dmf (DMF routine)

sgi_dmf_site (site-defined policy routine)

errcode Specifies an integer value generated by the originating

routine. This code may have many different meanings for a single value, depending on who the originator is.

errwho Specifies an integer value that describes in more detail

the originator of the error. Use the

DmuLogGetErrWhoImage() subroutine to access a

character string corresponding to this value.

severity Specifies an integer value that describes the severity of

the error. Use the DmuLogGetSeverityImage()

subroutine to access a character string corresponding to

this value.

position Specifies a character pointer to a string that contains the

position of where the error was generated. For example, this could be a pointer to a character string generated using the __FILE__ and __LINE__ cpp(1)

macros. This field may be NULL.

host Specifies a character pointer to a string that contains the

hostname where the error originated.

message Specifies a character pointer to a string that contains the

body of the error message.

DmuError t

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

DmuEvents t

The DmuEvents_t object defines the various event mask settings that a file may contain.

Valid settings are defined as the logical OR of any of the following:

DMF_EVENT_READ Generates a kernel event for each read request on the

file.

DMF_EVENT_WRITE Generates a kernel event for each write request on the

file.

DMF_EVENT_TRUNCATE Generates a kernel event for each truncate request on

the file.

DMF_EVENT_DESTROY Generates a kernel event will be generated for each

destroy request on the file.

DmuFhandle t

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

007–3681–019

The public member fields and functions of this class are as follows:

hanp Specifies a character array containing the file handle.

fromFhandleImage() Copies an ASCII file handle image string into the hanp

field.

toFhandleImage() Copies the hanp field into a DmuStringImage_t

object.

is_valid() Verifies the validity of the hanp field.

DmuFullRegbuf_t

The DmuFullRegbuf_t object defines the DMF fullregion buffer information for a file Only a single region constituting of the whole file is supported.

The public member fields and functions of this class are as follows:

arrent Specifies the number of regions in the regions array.

regent Specifies the number of regions in the regions array

that are valid. Only 0 and 1 are supported.

regions Specifies a DmuFullRegion_t array. See

"DmuRegion_t" on page 229.

DmuFullstat_t

The DmuFullstat_t object is a user-accessible version of the internal DMF fullstat object. It contains all of the basic stat(2) information regarding the file, as well as all of the DMAPI-related fields.

The public member fields and functions of this class are as follows:

inconsistencies in the fields.

stat Specifies a DmuStat_t object that contains the fields

representing those in the stat(5) structure. See the

stat(2) system call.

evmask Specifies a DmuEvents_t object that defines the event

mask for the file. See "DmuEvents_t" on page 227

regbuf Specifies a DmuRegionbuf_t object that defines the

regions of the file. See "DmuRegionbuf_t" on page 229.

attr Specifies a DmuAttr_t object that defines the DMF

attribute of the file. See "DmuAttr_t" on page 219.

host Specifies the hostname where the file is native.

mntpt Specifies a DmuOpaque_t object (defined in

libdmfcom. H) defining the mount point of the

filesystem containing the file on host.

relpath Specifies the relative path of the file in mntpt on host.

is valid() Returns 1 if the DmuFullstat t is valid.

DmuRegion_t

The DmuRegion_t object defines a filesystem region.

The public member fields and functions of this class are as follows:

rg_offset Defines the region starting offset in bytes. The start of

the file is byte 0.

rg_size Defines the region size in bytes.

rg_flags Defines the region event flag bitmask. See

"DmuEvents_t" on page 227.

DmuRegionbuf_t

The DmuRegionbuf_t object defines the filesystem region buffer information for a file. Only a single region constituting the whole file is supported.

007–3681–019

The public member fields and functions of this class are as follows:

arrent Specifies the number of regions in the regions array.

Specifies the number of regions in the regions array

that are valid. Only 0 and 1 are supported.

regions Specifies the DmuRegion_t array. See the

DmuRegion_t description.

DmuReplyOrder_t

The DmuReplyOrder_t object is used to select the order in which asynchronous replies are to be returned by the API reply processing subroutines.

Valid settings are defined as follows:

DmuAnyOrder Returns replies in the order the replies are received.

DmuReqOrder Returns replies in the order the requests were issued.

DmuReplyType_t

The DmuReplyType_t object is used to select the type of reply that an API can receive after sending a request. All requests will receive a final reply when the dmusrcmd process has completed processing the request whether it was successful or not.

Valid settings are defined as follows:

DmuIntermed Specifies an intermediate reply. An informational

message to alert the caller that the request is being processed and may not complete for some time. An example of this is the intermediate reply that is sent when a put request has been forwarded to an MSP or LS for processing and that the completion reply is

deferred until that operation is complete.

DmuFinal Specifies the final reply for the request.

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

DmuSeverity_t

The DmuSeverity_t object specifies the level of message reporting.

Valid settings are defined as follows:

DmuSevDebug4 Highest level of debug reporting.

DmuSevDebug3 Second-highest level of debug reporting.

DmuSevDebug2 Third-highest level of debug reporting.

DmuSevDebug1 Lowest level of debug reporting.

DmuSevVerbose Verbose message reporting.

DmuSevInform Informative message reporting.

DmuSevWarn Warning message reporting.

DmuSevFatal Error message reporting.

DmuVolGroup t

The DmuVolGroup_t object defines a volume group name. As an entry in a DmuVolGroups_t array, it is used to specify one of the volume groups to be used for a DMF put request. For more information about volume groups, see "How DMF Works" on page 6.

The public member field and function of this class is as follows:

vgname Specifies a character pointer to string containing the

name of a valid volume group.

DmuVolGroups t

The DmuVolGroups_t object defines an array of DmuVolGroup_t objects. This object is used to specify the list of volume groups to which a caller would like a file to be written in a DMF put request.

The public member fields and functions of this class are as follows:

setVolGroup() Adds a DmuVolGroup_t object to the internal

DmuVolGroup_t array.

clearVolGroup() Removes a DmuVolGroup_t object from the internal

DmuVolGroup_t array.

007-3681-019 231

numVolGroups() Returns the number of DmuVolGroup_t objects in the

internal DmuVolGroup_t array.

resetVolGroups() Clears the internal DmuVolGroup_t array.

toVolGroupsImage() Converts a DmuVolGroups_t object to a

DmuStringImage_t (defined in libdmfcom.H) in the

following format:

vgname1 vgname2 ...

The delimiter between multiple vgname values may be

a space, a tab, or a comma.

fromVolGroupsImage()
Converts a string image of the following format to a

DmuVolGroups_t object:

vgname1 vgname2 ...

The delimiter between multiple vgname values may be

a space, a tab, or a comma.

User-Accessible API Subroutines for libdmfusr.so.2

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

- "Context Manipulation Subroutines"
- "DMF File Request Subroutines" on page 235
- "Request Completion Subroutines" on page 247

Context Manipulation Subroutines

The DmuContext_t object manipulated by the DmuCreateContext(), DmuDestroyContext(), and DmuChangedDirectory() subroutines is designed to be completely opaque to the application. The context is used on all API subroutine calls so that the API can successfully manage user request and reply processing, but its internal contents are of no interest or use to the application.

You can use multiple DmuContext_t objects within the same process if desired.

DmuCreateContext() Subroutine

The DmuCreateContext() subroutine creates an opaque context for the API to use to correctly communicate with the dmusrcmd process. This subroutine should be the first API subroutine called by a DMF user command. Not only is the context created, but the communication channel to the dmusrcmd process is initialized.

Normally, a context would be used for multiple requests and only destroyed when no more requests are to be made. Creating and destroying a context for each request is likely to be inefficient if done frequently.

The prototype is as follows:

The parameters are as follows:

prog_name Contains the name of the program. This field can be the

full pathname of the program or some other

representation.

libdmfusr.H) that specifies create options. The only

valid create_flags option is:

CREATE_CHDIR

Allows change-directory requests via the DmuChangedDirectory() routine.

See "DmuChangedDirectory()

Subroutine" on page 234.

severity Specifies a DmuSeverity_t object that specifies the

level of error reporting. See "DmuSeverity_t" on page

231.

007-3681-019 233

err_handler	Specifies a user-defined error handling subroutine. The DmuErrHandler_f object is defined in libdmfusr.H. If the err_handler parameter is NULL, the default error handler DmuDefErrHandler is used. For more information, see "DmuErrHandler_f" on page 226.
dmuctxt	Specifies a DmuContext_t object (defined in libdmfusr.H) that is returned with the address of the newly created API to be used on all subsequent subroutine calls that require the program's API context.
child_pid	Specifies the process ID (PID) of the child that is forked and executed to create the dmusrcmd process. This value is returned to the caller so that the caller is free to handle the termination of child signals as desired.
errs	Specifies a pointer to a DmuAllErrors_t object. This value may be NULL. If it is not NULL, the subroutine will use it to return errors. See "DmuAllErrors_t" on page 218.

If the DmuCreateContext call completes successfully, it returns DmuNoError.

DmuChangedDirectory() Subroutine

The DmuChangedDirectory subroutine changes the current directory of the context. This subroutine is useful to a process that will be making multiple API file requests using relative pathnames while the process might also be making chdir(3) subroutine calls.

When a process makes a chdir call, if the DmuChangedDirectory() subroutine is called before the next API file request that references a relative pathname is made, the file reference will be successfully made by the process.

The prototype is as follows:

new_directory Specifies a read-only character pointer to the string

containing the directory path that was passed on the

last chdir(3) subroutine call.

errs Specifies a pointer to a DmuAllErrors_t object. This

value may be NULL. If it is not NULL, the subroutine will use it to return errors. See "DmuAllErrors_t" on

page 218.

DmuDestroyContext() Subroutine

The DmuDestroyContext() subroutine destroys the API context dmuctxt. The memory that had been allocated for its use is freed.

The prototype is as follows:

extern DmuError_t
DmuDestroyContext(

DmuContext_t dmuctxt,
DmuAllErrors_t *errs)

The parameters are as follows:

dmuctxt Specifies a DmuContext_t object that was previously

created by DmuCreateContext().

errs Specifies a pointer to a DmuAllErrors_t object. This

value may be NULL. If it is not NULL, the subroutine will use it to return errors. See "DmuAllErrors_t" on

page 218.

DMF File Request Subroutines

Each of the following subroutines makes a DMF file request. The context parameter that is included in each of these subroutines must have been already initialized via DmuCreateContext.

007–3681–019

Copy File Requests

The DmuCopyAsync() and DmuCopySync() subroutines perform copy requests in the manner of the dmcopy(1) command.

The DmuCopyAsync() subroutine returns immediately after the copy request has been forwarded to the dmusrcmd process. If a reply is desired, the caller must process the reply to this request. See "Request Completion Subroutines" on page 247.

The DmuCopySync() subroutine does not return until the requested copy has either completed successfully or been aborted due to an error condition.

The prototypes are as follows:

```
extern DmuError_t
DmuCopyAsync(
      const DmuContext_t dmuctxt,
      DmuCopyFlags_t copy_flags,
      const DmuCopyRanges_t *copyranges,
           DmuPriority_t priority,
                       *request_id,
           DmuReqid_t
           DmuAllErrors_t *errs)
extern DmuError_t
DmuCopySync(
      const DmuContext_t dmuctxt,
      DmuCopyFlags_t copy_flags,
      const DmuCopyRanges_t *copyranges,
           DmuPriority_t priority,
           DmuAllErrors_t *errs)
```

The parameters are as follows:

dmuctxt Specifies a DmuContext_t object that was previously

created by ${\tt DmuCreateContext()}.$

srcfile_path Specifies the pathname of the source (input) file for the

copy operation. It must be an offline or dual-state DMF

file.

dstfile_path

Specifies the pathname of the destination (output) file for the copy operation. This path must point to a file that exists or can be created in a DMF-managed filesystem that is native on the same host as that of the source file's filesystem.

copy flags

Specifies the OR'd value of the following copy operation flags as defined in libdmfcom. H:

- COPY_NONE No flags specified.
- COPY_PRESV_DFILE Do not truncate the destination file before the copy operation.
- COPY_ADDR_ALIGN Allow an address in the destination file that is greater than the size of the file.
- COPY_NOWAIT If the daemon is not available to process the request, do not wait. Return immediately.

copyranges

Specifies a pointer to a DmuCopyRanges_t object, as defined in "DmuCopyRanges_t" on page 225 and in libdmfcom.H. This object can have only one DmuCopyRange_t as defined in "DmuCopyRange_t" on page 224 and in libdmfcom.H.

priority

Specifies a DmuPriority_t object (defined in libdmfcom. H) that defines the request priority. (Deferred implementation.)

request_id

Specifies a pointer to a DmuReqid_t object (defined in libdmfcom.H) parameter that will be returned with the unique request ID of the asynchronous request. This value can be used when processing DmuCompletion_t objects (see "Request Completion Subroutines" on page 247).

errs

Specifies a pointer to a DmuAllErrors t object. This value may be NULL. If it is not NULL, the subroutine will use it to return errors. See "DmuAllErrors_t" on page 218.

If the subroutine succeeds, it returns DmuNoError.

007-3681-019 237

fullstat Requests

The following subroutines send a fullstat request to the dmusrcmd process:

```
DmuFullstatByFhandleAsync()
DmuFullstatByFhandleSync()
DmuFullstatByPathAsync()
DmuFullstatByPathSync()
```

These subroutines have the following things in common:

- The 'Sync' versions of these subroutines do not return until the DmuFullstat_t has been received or the request has been aborted due to errors.
- The 'Async' versions of these subroutines return immediately after successfully forwarding the fullstat request to the dmusrcmd process. If a reply is desired, the caller must process the reply to this request. See "Request Completion Subroutines" on page 247. That is the only way to actually receive the DmuFullstat_t object for an 'Async' fullstat request, however. The DmuFullstatCompletion() subroutine has been supplied to extract the fullstat information from a fullstat completion object.
- The 'ByPath' versions of these subroutines allow the target file to be defined by its pathname.
- The 'ByFhandle' versions of these subroutines allow the target file to be defined by its filesystem handle, the fhandle. These subroutines are valid only when the command making the call is on the DMF server machine, and they are valid only when a user has sufficient (root) privileges.

These subroutines can return a successful completion (DmuNoError), but might not return valid DmuFullstat_t information. The subroutines are designed to return the normal stat type information regardless of whether a DMAPI fullstat could be successfully completed. Upon return from these subroutines, the caller can use the DmuFullstat_t is_valid() member function to verify the validity of the DMAPI information in the DmuFullstat_t block.

The ultimate result of this request is the transfer of a DmuFullstat_t object to the caller.

The prototypes are as follows:

*client_fhandle,

Specifies the pointer that will be returned with the

Specifies the pointer that will be returned with the

DmuReqid_t *request_id, DmuAllErrors_t *errs) extern DmuError_t DmuFullstatByFhandleSync(const DmuContext_t dmuctxt, DmuFhandle_t *client_fhandle, const DmuFullstat_t *dmufullstat, DmuAllErrors_t *errs) extern DmuError_t DmuFullstatByPathAsync(const DmuContext_t dmuctxt, const char *path, DmuReqid_t *request_id, DmuAllErrors_t *errs) extern DmuError_t DmuFullstatByPathSync(DmuContext_t const dmuctxt, const *path, DmuFullstat_t *dmufullstat, DmuFhandle_t *fhandle, DmuAllErrors_t *errs) The parameters are as follows: dmuctxt Specifies a DmuContext_t object that was previously created by DmuCreateContext(). Specifies the DMF filesystem fhandle of the target file. client_fhandle Valid for use only by a privileged (root) user on the DMF server machine. Specifies the relative or absolute pathname of the target path file.

DmuFullstat_t object.

DmuFhandle_t value.

DmuFhandle_t

const

007–3681–019

dmufullstat

fhandle

request_id Specifies a pointer to a DmuReqid_t object (defined in

libdmfcom.H) parameter that will be returned with the unique request ID of the asynchronous request.

This value can be used when processing

DmuCompletion_t objects (see "Request Completion

Subroutines" on page 247).

errs Specifies a pointer to a DmuAllErrors_t object. This

value may be NULL. If it is not NULL, the subroutine will use it to return errors. See "DmuAllErrors_t" on

page 218.

If the subroutine succeeds, it returns DmuNoError.

put File Requests

The following subroutines perform the put DMF request:

```
DmuPutByFhandleAsync()
DmuPutByFhandleSync()
DmuPutByPathAsync()
DmuPutByPathSync()
```

These subroutines have the following things in common:

- The 'Sync' versions do not return until the put request has either completed successfully, or been aborted due to errors.
- The 'Async' versions return immediately after successfully forwarding the put request to the dmusrcmd process. If a reply is desired, the caller must process the reply to this request. See "Request Completion Subroutines" on page 247.
- The 'ByPath' versions allow the target file to be defined by its pathname.
- The 'ByFhandle' versions allow the target file to be defined by its filesystem handle, the fhandle. These subroutines are valid only when the command making the call is on the DMF server machine, and they are valid only when a user has sufficient (root) privileges.

The prototypes are as follows:

```
DmuFhandle_t
                                     *client_fhandle,
              const
                      DmuMigFlags_t mig_flags,
              const
                      DmuByteRanges_t *byteranges,
                      DmuVolGroups_t *volgroups,
              const
                      DmuPriority_t priority,
                      DmuReqid_t
                                     *request_id,
                      DmuAllErrors_t *errs)
extern DmuError_t
DmuPutByFhandleSync(
              const
                      DmuContext t dmuctxt,
                      DmuFhandle_t *client_fhandle,
              const
                      DmuMigFlags_t mig_flags,
                      DmuByteRanges_t *byteranges,
              const
                      DmuVolGroups_t *volgroups,
              const
                      DmuPriority_t priority,
                      DmuAllErrors_t *errs)
extern DmuError_t
DmuPutByPathAsync(
              const
                      DmuContext_t dmuctxt,
              const
                      char
                                     *path,
                      DmuMigFlags_t
                                     mig_flags,
              const
                      DmuByteRanges_t *byteranges,
                      DmuVolGroups_t *volgroups,
              const
                      DmuPriority_t priority,
                      DmuReqid_t
                                     *request_id,
                      DmuAllErrors_t *errs)
extern DmuError_t
DmuPutByPathSync(
                      DmuContext_t
                                     dmuctxt,
              const
                      char
                                     *path,
              const
                      DmuMigFlags_t mig_flags,
                      DmuByteRanges_t *byteranges,
              const
              const
                      DmuVolGroups_t *volgroups,
                      DmuPriority_t priority,
                      DmuAllErrors_t *errs)
```

The parameters are as follows:

dmuctxt	Specifies a DmuContext	_t object that was p	oreviously
---------	------------------------	----------------------	------------

created by DmuCreateContext().

client_fhandle Specifies the DMF filesystem fhandle of the target file.

Valid for use only by a privileged (root) user on the

DMF server machine.

path Specifies the relative or full pathname of the target file.

mig_flags Specifies the following migration flags as defined in

libdmfcom.H:

MIG_NONE – No flags specified.

• MIG_FREE – Free the space associated with the file.

 MIG_NOWAIT – If the daemon is not available to process the request, do not wait. Return

immediately.

byteranges Specifies a pointer to a DmuByteRanges_t object. See

"DmuByteRanges_t" on page 220.

volgroups Specifies a pointer to a DmuVolGroups_t object. See

"DmuVolGroups_t" on page 231.

priority Specifies a DmuPriority_t object (defined in

libdmfcom. H) that defines the request priority.

(Deferred implementation.)

request_id Specifies a pointer to a DmuReqid_t object (defined in

libdmfcom.H) parameter that will be returned with the unique request ID of the asynchronous request.

This value can be used when processing

DmuCompletion_t objects (see "Request Completion

Subroutines" on page 247).

errs Specifies a pointer to a DmuAllErrors_t object. This

value may be NULL. If it is not NULL, the subroutine will use it to return errors. See "DmuAllErrors_t" on

page 218.

If the subroutine succeeds, it returns DmuNoError.

get File Requests

The following subroutines perform the get DMF request:

```
DmuGetByFhandleAsync()
DmuGetByFhandleSync()
DmuGetByPathAsync()
DmuGetByPathSync()
```

These subroutines have the following things in common:

- The 'Sync' versions do not return until the get request has either completed successfully or has been aborted due to errors.
- The 'Async' versions return immediately after successfully forwarding the get request to the dmusrcmd process. If a reply is desired, the caller must process the reply to this request. See "Request Completion Subroutines" on page 247.
- The 'ByPath' versions of these calls allow the target file to be defined by its pathname.
- The 'ByFhandle' versions allow the target file to be defined by its filesystem handle, the fhandle. These subroutines are valid only when the command making the call is on the DMF server machine, and they are valid only when a user has sufficient (root) privileges.

The prototypes are as follows:

```
extern DmuError_t
DmuGetByFhandleAsync(
               const DmuContext_t dmuctxt,
const DmuFhandle_t *client_fhandle,
                      DmuRecallFlags_t recall_flags,
                       DmuByteRanges_t *byteranges,
               const
                       DmuPriority_t priority,
                       DmuReqid_t *request_id,
                       DmuAllErrors_t *errs)
extern DmuError_t
DmuGetByFhandleSync(
               const DmuContext_t dmuctxt,
                       DmuFhandle_t *client_fhandle,
               const
                       DmuRecallFlags_t recall_flags,
                       DmuByteRanges_t *byteranges,
               const
```

	OmuPriority_t OmuAllErrors_t		
extern DmuError_t DmuGetByPathAsync(
const	OmuContext_t char OmuRecallFlags_t	<pre>dmuctxt, *path, recall flags.</pre>	
const I I I	OmuByteRanges_t OmuPriority_t OmuReqid_t OmuAllErrors_t	*byteranges, priority, *request_id, *errs)	
extern DmuError_t			
const	OmuContext_t char OmuRecallFlags_t	<pre>dmuctxt, *path, recall flags,</pre>	
const I	OmuByteRanges_t OmuPriority_t OmuAllErrors_t	*byteranges, priority, *errs)	
The parameters are as follow	vs:		
dmuctxt		ontext_t object that was previously reateContext().	
client_fhandle	Specifies the DMF filesystem fhandle of the target file. Valid for use only by a privileged (root) user on the DMF server machine.		
path	Specifies the relative or full pathname of the target fil		
recall_flags	Specifies the following recall flags as defined in libdmfcom. H:		
	• RECALL_NONE - No flags specified.		
	 RECALL_NOWAIT – If the daemon is not available to process the request, do not wait. Return immediately. 		
byteranges Specifies a pointer to a DmuByteRanges_t obj "DmuByteRanges_t" on page 220.			

priority Specifies a DmuPriority_t object (defined in

libdmfcom. H) that defines the request priority.

(Deferred implementation.)

request_id Specifies a pointer to a DmuReqid_t (defined in

libdmfcom. H) parameter that will be returned with the unique request ID of the asynchronous request.

This value can be used when processing

DmuCompletion_t objects (see "Request Completion

Subroutines" on page 247).

errs Specifies a pointer to a DmuAllErrors_t object. This

value may be NULL. If it is not NULL, the subroutine will use it to return errors. See "DmuAllErrors_t" on

page 218.

If the subroutine succeeds, it returns DmuNoError.

settag File Requests

The settag request performs the same functional task as the dmtag(1) command. The following subroutines perform the settag DMF request:

DmuSettagByFhandleAsync()
DmuSettagByFhandleSync()
DmuSettagByPathAsync()
DmuSettagByPathSync()

These subroutines have the following things in common:

- The 'Sync' versions do not return until the settag request has either completed successfully or has been aborted due to errors.
- The 'Async' versions return immediately after successfully forwarding the settag request to the dmusrcmd process. If a reply is desired, the caller must process the reply to this request. See "Request Completion Subroutines" on page 247.
- The 'ByPath' versions allow the target file to be defined by its pathname.
- The 'ByFhandle' versions allow the target file to be defined by its filesystem
 handle, the fhandle. These subroutines are valid only when the command
 making the call is on the DMF server machine and when a user has sufficient
 (root) privileges.

007–3681–019

The prototypes are as follows:

```
extern DmuError_t
DmuSettagByFhandleAsync(
              const DmuContext_t
                                       dmuctxt,
              const
                      DmuFhandle_t *client_fhandle,
                      DmuSettagFlags_t settag_flags,
                      DmuSitetag_t sitetag,
                      DmuPriority_t priority,
                      DmuRegid_t
                                      *request_id,
                      DmuAllErrors_t *errs)
extern DmuError_t
DmuSettagByFhandleSync(
                      DmuContext_t dmuctxt,
DmuFhandle_t *client_fhandle,
                      DmuContext_t
              const
              const
                      DmuSettagFlags_t settag_flags,
                      DmuSitetag_t
                                      sitetag,
                      DmuPriority_t
                                       priority,
                      DmuAllErrors_t *errs)
extern DmuError_t
DmuSettagByPathAsync(
              const
                      DmuContext_t
                                       dmuctxt,
              const
                      char
                                       *path,
                      DmuSettagFlags_t settag_flags,
                                     sitetag,
                      DmuSitetag_t
                      DmuPriority_t priority,
                      DmuRegid_t
                                      *request_id,
                      DmuAllErrors_t *errs)
extern DmuError_t
DmuSettagByPathSync(
              const
                      DmuContext_t
                                       dmuctxt,
              const
                      char
                                       *path,
                      DmuSettagFlags_t settag_flags,
                      DmuSitetag_t
                                     sitetag,
                      DmuPriority_t
                                       priority,
                      DmuAllErrors_t *errs)
```

The parameters are as follows:

dmuctxt Specifies a DmuContext_t object that was previously

created by DmuCreateContext().

client_fhandle Specifies the DMF filesystem fhandle of the target file.

Valid for use only by a privileged (root) user on the

DMF server machine.

path Specifies the relative or full pathname of the target file.

settag_flags Specifies the following settag flags as defined in

libdmfcom.H:

SETTAG_NONE – No flags specified.

SETTAG_NOWAIT – If the daemon is not available to

process the request, do not wait. Return

immediately.

sitetag Defines the file site tag value. See dmtag(1).

priority Specifies a DmuPriority_t object (defined in

libdmfcom. H) that defines the request priority.

(Deferred implementation.)

request_id Specifies a pointer to a DmuReqid_t (defined in

libdmfcom. H) parameter that will be returned with the unique request ID of the asynchronous request.

This value can be used when processing

DmuCompletion_t objects (see "Request Completion

Subroutines" on page 247).

errs Specifies a pointer to a DmuAllErrors_t object. This

value may be NULL. If it is not NULL, the subroutine will use it to return errors. See "DmuAllErrors_t" on

page 218.

If the subroutine succeeds, it returns DmuNoError.

Request Completion Subroutines

The request completion subroutines are provided so that the application can process the completion events of any asynchronous requests it might have issued. The caller can choose to process each request's completion object (DmuCompletion_t) or to be

notified when each request has responded with either an intermediate or final (completion) reply.

The asynchronous requests described previously along with the following completion subroutines allow the user to achieve maximum parallelization of the processing of all requests.

DmuAwaitReplies() Subroutine

The DmuAwaitReplies() subroutine performs a synchronous wait until the number of outstanding request replies of the type specified is less than or equal to max_outstanding. This subroutine is called by a user who does not want to perform individual processing of each outstanding request, but wants to know when a reply (intermediate or final) has been received for each request that has been sent to this point.

The prototype is as follows:

The parameters are as follows:

dmuctxt	Specifies a DmuContext_t object that was previously created by DmuCreateContext().	
type	Defines the type of reply to be received. The caller can wait for an intermediate or final reply for the outstanding requests.	
	See the definition of DmuReplyType_t in "DmuReplyType_t" on page 230 or in libdmfcom.H.	
max_outstanding	Specifies the number of outstanding requests allowed for which the type reply has not been received before the subroutine returns. If this parameter is 0, all type replies will have been received when the subroutine returns.	
errs	Specifies a pointer to a DmuAllErrors_t object. This value may be NULL. If it is not NULL, the subroutine	

will use it to return errors. See "DmuAllErrors_t" on page 218.

If no errors occurred getting the next reply, this subroutine returns DmuNoError.

DmuFullstatCompletion() Subroutine

The DmuFullstatCompletion() subroutine can be called when asynchronous fullstat replies are being processed by DmuGetNextReply() or DmuGetThisReply(). When the reply is received, the DmuCompletion_t object that is part of the reply can be used as an input parameter to this routine, which will then extract the DmuFullstat_t object and the DmuFhandle_t objects that are contained in the DmuCompletion_t object's ureq_data field.

The prototype is as follows:

```
extern DmuError_t
DmuFullstatCompletion(
               DmuCompletion_t *comp;
                                *dmufullstat,
               DmuFullstat_t
               DmuFhandle_t
                                *fhandle,
               DmuAllErrors_t
                                *errs)
```

The parameters are as follows:

comp	Specifies the DmuCompletion_t object from an
	asynchronous fullstat request

asynchronous fullstat request.

dmufullstat Specifies a pointer to an existing DmuFullstat_t

> object. If comp references a successful fullstat request, dmufullstat will be set to be equal to the DmuFullstat_t that was returned with the reply.

fhandle Specifies a pointer to an existing DmuFhandle_t object.

> If comp references a successful fullstat request, fhandle will be set to be equal to the DmuFhandle_t

that was returned with the reply.

Specifies a pointer to a DmuAllErrors_t object. This errs

value may be NULL. If it is not NULL, the subroutine

007-3681-019 249 will use it to return errors. See "DmuAllErrors_t" on page 218.

DmuGetNextReply() Subroutine

The DmuGetNextReply() subroutine returns the completion object of the next reply based on the order specified on the call.

The caller can specify DmuIntermed or DmuFinal for the type parameter. If DmuIntermed is specified and an intermediate reply is the next reply received and there are no completed replies available for processing, the comp parameter is not set (will be NULL) when the subroutine returns. An intermediate reply has no completion object associated with it; a return of this type is informational only.

This subroutine performs a synchronous wait until a request reply of the type specified on the call is received. At the time of the call, any reply that has already been received and is queued for processing is returned immediately.

The prototype is as follows:

The parameters are as follows:

dmuctxt	Specifies a DmuContext	t ob	iect that	was :	previously
amacciic	pecnies a binaconicenc_		ICCC CITCAC	***	pictionsi

created by DmuCreateContext().

order Defines the order in which the request replies should be

returned. The caller can process the replies in the order the replies are received (DmuAnyOrder) or in the order

the requests were issued (DmuReqOrder).

See the definition of DmuReplyOrder_t in

"DmuReplyOrder_t" on page 230 or in libdmfcom.H.

type Defines the type of reply to be received. The caller can

wait for an intermediate or final reply for the

outstanding requests. The receipt of an intermediate

reply returns no data.

comp

Specifies a pointer to an existing DmuCompletion_t object. If a reply was available for processing according to the parameters on the calling subroutine, the DmuCompletion_t object pointed to by comp will be set with all of the appropriate values. See "DmuCompletion_t" on page 224.

If the reply_code field of the comp parameter is not DmuNoError, the comp->allerrors object will contain the error information needed to determine the cause of the error.

Note: The errs parameter on the subroutine call does not contain the error information for the failed request.

errs

Specifies a pointer to a DmuAllErrors_t object. This value may be NULL. If it is not NULL, the subroutine will use it to return errors. See "DmuAllErrors_t" on page 218.

Note: This object will return errors that occurred while waiting for or receiving this reply. It does not refer to the errors that might have occurred during the request processing that resulted in the reply Those errors are available in the comp object.

If no errors occurred getting the next reply, this subroutine returns <code>DmuNoError</code>. If there are no outstanding requests pending, a return code of <code>DME_DMU_QUEUEEMPTY</code> is returned. You can use a check for <code>DME_DMU_QUEUEEMPTY</code> to terminate a while loop based on this subroutine. Any other error return code indicates an error, and the <code>errs</code> parameter can be processed for the error information.

DmuGetThisReply() Subroutine

The DmuGetThisReply() subroutine returns the completion object of the specified request. This subroutine performs a synchronous wait until a request reply specified on the call is received.

The prototype is as follows:

```
extern DmuError_t
DmuGetThisReply(
       const DmuContext_t dmuctxt,
              DmuReqid_t request_id,
              DmuCompletion_t *comp,
              DmuAllErrors_t *errs)
```

The parameters are as follows:

Specifies a DmuContext_t object that was previously dmuctxt

created by DmuCreateContext().

Specifies the unique request ID of the request for which request_id

comp Specifies a pointer to an existing DmuCompletion_t

object. If a reply was available for processing according to the parameters on the calling subroutine, the DmuCompletion_t object pointed to by comp will be

set with all of the appropriate values. See

"DmuCompletion_t" on page 224.

The reply_code field of the comp parameter is the ultimate status of the request. A successful comp has a reply_code of DmuNoError. If the reply_code of comp is not DmNoError, the comp->allerrors object will contain the error information needed to determine the cause of the error.

Note: The errs parameter on the subroutine call does not contain the error information for the failed request.

Specifies a pointer to a DmuAllErrors_t object. This

value may be NULL. If it is not NULL, the subroutine will use it to return errors. See "DmuAllErrors_t" on

page 218.

the caller wants to wait.

errs

252 007-3681-019 **Note:** This object will return errors that occurred while waiting for or receiving this reply. It does not refer to the errors that might have occurred during the request processing that resulted in the reply Those errors are available in the comp object.

If no errors occurred getting the next reply, this subroutine returns DmuNoError. Any other error return code indicates an error, and the errs parameter can be processed for the error information.

Site-Defined Policy Subroutines and the sitelib.so Library

This appendix provides an overview of the site-defined policy feature and a summary of the policy subroutines sites may write:

- "Overview of Site-Defined Policy Subroutines" on page 255
- "Getting Started" on page 256
- "Considerations" on page 258
- "sitelib.so Data Types" on page 259
- "Site-Defined Policy Subroutines" on page 263
- "Helper Subroutines for sitelib.so" on page 269

Overview of Site-Defined Policy Subroutines

Site-defined policy subroutines are loaded dynamically by DMF to provide custom decision-making at key points in its processing. Several DMF processes, including dmfdaemon, can call subroutines within sitelib.so.

You do not need to use this feature, in which case DMF will function as documented in the manuals and man pages. But if you wish, you can implement one or more of these subroutines in order to override DMF's default behavior.

If you use the site-defined policy feature, you must communicate the policy changes to your user community; otherwise, they will not be able to predict how the user commands will work. The man page for any command with a site-defined policy will state something like the following:

If your site is using the site-defined policy feature, the default behavior may be overridden. Please check with your administrator for any behavior differences due to site-defined policies.

You should also consider adding ERROR, WARN, and INFO messages into the reply stream for commands you customize so that you can routinely return messages to the user that explain what was changed in their request. Doing so will allows the users to understand why the behavior was different from what they expected.

The subroutines are written in C++ according to the subroutine prototypes in /usr/include/dmf/libdmfadm.H. They are placed in a shared-object library called /usr/lib/dmf/sitelib.so.

The parameters and return values of the subroutines and the name of the sitelib.so library are fixed and cannot be altered by the site. In general, the parameters provide all of the information DMF has that is relevant to the purpose of the subroutine, which is described in the comments preceding each subroutine.

The code within the subroutines performs whatever processing the site wishes. To assist in several common operations, such as extracting information from the DMF configuration file, optional helper subroutines are provided in /usr/include/dmf/libdmfadm.H.

Getting Started

The /usr/share/doc/dmf-*/info/sample directory contains the following files to demonstrate generating the sitelib.so library:

- sample_sitelib.C contains source code of sample subroutines
- sample_sitelib.mk is the makefile

Note: If you use these files as a base for implementing subroutines of your own, be sure to keep them in a different directory and/or rename them to avoid any conflict when DMF is upgraded and new sample files are installed. For example, you could rename the files sitelib.c and sitelib.mk.

Do the following:

1. Copy sample_sitelib.C and its associated makefile sample_sitelib.mk from /usr/share/doc/dmf-*/info/sample to a directory of your own with names of your own choice.

For example, if you wanted to work in the /tmp/testdmf directory:

- \$ cp /usr/share/doc/dmf-*/info/sample_sitelib.C /tmp/testdmf/sitelib.C
 \$ cp /usr/share/doc/dmf-*/info/sample_sitelib.mk /tmp/testdmf/sitelib.mk
 - 2. In the makefile, specify the stem from which the library filename and source code filename will be derived by editing the value for the SITELIB parameter. For

example, to use a stem of sitelib (that is, sitelib.so for the library and sitelib.c for the source code file):

SITELIB=sitelib

Note: Although you can set the SITELIB value to something other than sitelib for testing purposes, when you actually want to run with DMF, it must be sitelib.

3. Read the comments at the start of each subroutine and alter the supplied code to suit your requirements. As supplied, each subroutine is disabled. To enable one or more subroutines, modify the SiteFncMap variable at the bottom of the source file (in our example, sitelib.C).

Note: The name of the SiteFncMap variable is fixed and cannot be altered. However, you can change the names of the site-defined subroutines such as SiteCreateContext().

4. Build the sitelib.so library by using the make(1) command:

```
$ make -f sitelib.mk
```

5. Print a list of the subroutines that have been enabled and visually verify that it is what you expect:

```
# make -f sitelib.mk verbose
```

6. Install the library on a DMF server, which requires you to be the root user:

```
$ su
# make -f sitelib.mk install
```

Note: You do not need to install sitelib.so on a machine that functions only as a DMF client.

For subroutines that affect the operation of the DMF daemon, library server, or MSP, you must wait for a minute or so for the new sitelib.so library to be noticed. You will see a message in the relevant log file when this happens.

7. Test your new library by monitoring the relevant logfile with tail -f while you present test cases to DMF. You may also find it useful to have a Resource Watcher configured and running or to use dmstat.

Considerations

As you write your own custom subroutines, be aware of the following:

- The sitelib.so file must be owned by root and must not be writable by anyone else, for security reasons. If these conditions are not met, DMF will ignore sitelib.so and use the default behavior.
- The sitelib.so library should not use the stdin, stdout, or stderr files as this could cause problems for DMF, possibly endangering data. For information about sending messages to users or to log files, see "DmaSendLogFmtMessage()" on page 279 and "DmaSendUserFmtMessage()" on page 280.
- If you overwrite the sitelib.so file while it is in use (for example by copying a new version of your file over the top of the old one), DMF processes may abort or run improperly. The DMF daemon may or may not be able to restart them properly.

To update the file, you should do one of the following:

- Use the mv(1) command to move the new file over the top of the old one, so that any existing DMF processes will continue to use the previous version of the file, which is now unlinked pending removal. The install target in the supplied makefile is also a safe way to update the file.
- Delete the old file with rm(1) before installing the new one using cp, mv, or make install.
- Shut down DMF while the update takes place.

This warning also applies to changes to the DMF configuration file.

- Site-defined policy subroutines should not call subroutines in libdmfusr.so, such as DmuSettagByPathSync(). They are free to call member functions of classes defined in libdmfcom.H, such as DmuVolGroups_t::numVolGroups().
- At times, the site-defined subroutines may be called many times in rapid succession. They should therefore be as efficient as possible, avoiding any unnecessary processing, especially of system calls.

For example, when dmfsfree is invoked to prevent a filesystem from filling, site-defined subroutines may be called one or more times for every file in the filesystem as dmfsfree prepares its list of candidates prior to migrating and/or freeing some of them. If the functions are slow, DMF may not be able to react to the situation in time to prevent the filesystem from filling.

• For IRIX systems, DMF processes that call routines in sitelib.so were compiled using MIPSpro compilers. Compiling the C++ routines that make up sitelib.so on IRIX systems with compilers other than MIPSpro compilers may result in incompatibilities causing load-time or run-time errors.

sitelib.so Data Types

The data types described in this section are defined in libdmfadm.H. The information in this section is provided as a general description and overall usage outline. Other data types that are referenced in this file are defined in libdmfcom.H; see Appendix B, "DMF User Library libdmfusr.so" on page 213.

Note: For the most current definitions of these types, see the libdmfadm. H file.

DmaContext_t

The DmaContext_t object stores information for DMF in order to provide continuity from one subroutine call to the next. It is an opaque object that is created when a DMF process first loads sitelib.so and it exists until that process unloads it. This context is provided as a parameter for each of the site-defined policy subroutines.

Site-defined subroutines cannot directly access the information held in the context, but they can obtain information from it by using the following subroutines:

- "DmaGetContextFlags()" on page 276
- "DmaGetProgramIdentity()" on page 277
- "DmaGetUserIdentity()" on page 278

Site-defined subroutines can also store their own information in the context and retrieve it on subsequent calls by using the following subroutines:

- "DmaSetCookie()" on page 281
- "DmaGetCookie()" on page 276

DmaFrom t

The DmaFrom_t object specifies the type of policy statement being evaluated.

There are the following possible values:

DmaFromAgeWeight Indicates that an AGE_WEIGHT policy statement is being

evaluated.

DmaFromSpaceWeight Indicates that a SPACE_WEIGHT policy statement is

being evaluated.

DmaFromVgSelect Indicates that a SELECT_MSP or SELECT_VG policy

statement is being evaluated.

DmaIdentity_t

The DmaIdentity_t object provides information, if known, about the program calling the site-defined subroutine and the user whose request generated the call.

The public member fields and functions of this class are as follows:

realm_type

Specifies the environment in which the type of data that is contained in the realm_data field is meaningful.

The following settings are defined:

- DMF_REALM_UNIX, which means that the unix_1 member of realm_data contains valid information
- DMF_REALM_UNKNOWN, which means that realm_data is not reliable

realm_data

Specifies user identity information that is specific to the environment defined by realm_type. Only the unix_1 member of the union is defined for the realm_type of DMF_REALM_UNIX.

If the UID and/or GID values are <code>Oxfffffffff</code>, the values are not reliable.

logical_name

Specifies a character string containing the program name of the process. This may be an absolute or relative pathname. If the value is unknown, the program name was unavailable.

product_name_and_revision

Specifies a character string containing the product name and revision (for example, DMF_3.1.0.0).

locale_1

Specifies a character string containing the locale value. See the locale(1) man page.

host

Specifies a character string containing the host on which the DmaIdentity_t originated.

pid

Specifies the process ID where the DmaIdentity_t originated.

instance_id

Specifies a further refinement of the PID field. Because a process may create more than one Dmaldentity_t, this value is incremented by one for each new Dmaldentity_t.

os_type

Specifies a character string containing a description of the operating system where the DmaIdentity_t originated.

007-3681-019 261

os_version

Specifies a character string containing a description of the operating system version where the DmaIdentity_t originated.

cpu_type

Specifies a character string containing a description of the CPU type where the Dmaldentity_t originated.

Note: Any of the descriptive character strings may be set to unknown if the field's true value cannot be determined.

DmaLogLevel t

The DmaLogLevel_t object specifies the level of a message. The administrator may select a log level in the DMF configuration file; messages with a less severe level than what is specified in the configuration file will not appear in the log.

DmaRealm_t

The DmaRealm_t object specifies the realm. Only the UNIX realm is supported.

DmaRecallType t

The DmaRecallType_t object specifies the type of kernel recall being performed.

SiteFncMap_t

The SiteFncMap_t object specifies the site subroutine map. The various DMF processes that can call subroutines in sitelib.so look for a variable named SiteFncMap, of type SiteFncMap_t, in the sitelib.so library. It then uses the addresses provided in this variable to find the site-defined subroutines. If the variable is not found, DMF will not make any calls to subroutines in sitelib.so.

Site-Defined Policy Subroutines

DMF looks for the variable named SiteFncMap, of type SiteFncMap_t, in the sitelib.so library. It then uses the addresses provided in this variable to find site-defined subroutines listed in this section. You can provide any number of these subroutines in the sitelib.so library.

SiteCreateContext()

The SiteCreateContext() subroutine provides the opportunity to create a site-specific setup. It is called when sitelib.so is loaded. If no such setup is required, it need not be implemented. If this subroutine returns anything other than DmuNoError, no other subroutines in sitelib.so, including SiteDestroyContext(), will be called by the current process, unless sitelib.so is changed and therefore reloaded.

This subroutine may not issue messages to the user because the user details are unknown at the time it is invoked. If it is invoked by a program with a log file, such as dmfdaemon, it can issue log messages by calling DmaSendLogFmtMessage(). You can call DmaGetContextFlags() to determine if it can issue log messages.

The prototype is as follows:

The parameter is as follows:

dmacontext

Refers to the context established when sitelib.so was loaded.

SiteDestroyContext()

The SiteDestroyContext() subroutine provides the opportunity for site-specific cleanup. It is called when sitelib.so is unloaded. If no such cleanup is required, it need not be implemented. This subroutine may not issue messages to the user because the user details are no longer valid at the time it is invoked. If it is invoked by a program with a log file, such as dmfdaemon, it can issue log messages by calling DmaSendLogFmtMessage(). You can call DmaGetContextFlags() to determine if it can issue log messages.

The prototype is as follows:

The parameter is as follows:

dmacontext

Refers to the context established when sitelib.so was loaded.

SiteKernRecall()

The SiteKernRecall() subroutine allows sites some control over kernel requests to recall a file. It is invoked when DMF receives a kernel request to recall a file. For example, a read() system call for a file that is currently in OFL state would result in SiteKernRecall() being called. The dmget command or the equivalent libdmfusr.so library call would not result in a call to SiteKernRecall().

This subroutine may accept or reject the request or change its priority; no other changes are possible. If the subroutine returns a value other than DmuNoError, the request will be rejected. Changing the priority has no effect at this time.

Note: offset and length pertain to the range of the file that the user's I/O request referenced, not the byte range that dmfdaemon will actually recall.

The subroutine may not issue messages to the user, but it can issue messages to the DMF daemon log.

The prototype is as follows:

dmacontext Re	efers to the context established when sitelib.so
---------------	--

was loaded.

fullstat Specifies the DmuFullstat_t of the file being recalled fhandle Specifies the DmuFhandle_t of the file being recalled offset Pertains to the range of the file that the user's I/O

request referenced.

length Pertains to the length of the file that the user's I/O

request referenced.

recall_type Specifies the type of recall.
operative_priority (Deferred implementation.)

SitePutFile()

The SitePutFile() subroutine allows sites some control over the DMF put requests. It is invoked when a dmput command is issued or when one of the following libdmfusr.so subroutines is called:

DmuPutByPathAsync()
DmuPutByPathSync()
DmuPutByFhandleAsync()
DmuPutByFhandleSync()

This subroutine is not called when automatic space management migrates a file.



Caution: If SitePutFile() is implemented, it takes precedence over any when clause being used to control MSP or volume group selection, whether or not SiteWhen() has been implemented.

If this subroutine returns a value other than DmuNoError, the put request will be rejected. The subroutine may not issue log messages, but it can issue messages to the user.

The prototype is as follows:

```
const
       DmuFullstat_t
                       *fstat,
       char
                       *path,
const
const DmuFhandle_t
                       *fhandle,
                       flags,
       int
const
const
       DmuVolGroups_t *policy_volgrps,
const
       DmuPriority_t
                       user_priority,
const
                       user_flags,
       DmuByteRanges_t *user_byteranges,
const
const
       DmuVolGroups_t *user_volgrps,
                       *operative_priority,
       DmuPriority_t
                        *operative_flags,
       DmuByteRanges_t *operative_byteranges,
       DmuVolGroups_t *operative_volgrps);
```

dmacontext	Refers to the context established who	en sitelib.so

was loaded.

fstat Specifies the DmuFullstat_t information of the target

file for the put request.

path Specifies the pathname of the target file for the put

request (if known) or NULL.

fhandle Specifies the DmuFhandle_t of the target file for the

put request.

flags Specifies whether the SitePutFile() subroutine is

called for the first time (0) or is replayed (nonzero). SitePutFile() can be called multiple times for the same request. For example, if dmfdaemon is not running, a dmput request will periodically try to establish a connection with it, and SitePutFile() may be called. If flags is 0, this is the first time that SitePutFile() has been called for a particular request. When a request is replayed, DMF reevaluates the parameters to SitePutFile() before calling it.

policy_volgrps Specifies an input parameter that contains the volume

groups and MSPs that have been selected by the policy

statements in the DMF configuration file.

user_priority
user_flags
user_byteranges
user_volgrps

Contains information entered by the user as a dmput parameter (where supported) or as a parameter to one of the following libdmfusr.so subroutines:

DmuPutByPathAsync()
DmuPutByPathSync()
DmuPutByFhandleAsync()
DmuPutByFhandleSync()

operative_priority operative_flags operative_byteranges operative_volgrps

Contains the information that will be used when the request is made to the dmfdaemon. These are all both input and output parameters. You can alter the operative_flags, operative_byteranges, and operative_volgrps values. (Currently, operative_priority is ignored. For compatibility with future releases of DMF, it is recommended that you do not alter the value of this parameter at this time.)

SiteWhen()

The SiteWhen() subroutine provides the opportunity to supply the value for the sitefn variable in when clauses in the following parameters:

AGE_WEIGHT SPACE_WEIGHT SELECT_MSP SELECT_VG

This subroutine is not supported in when clauses associated with a DCM cache.



Caution: If SitePutFile() is implemented, it takes precedence over any when clause being used to control MSP or volume group selection, whether or not SiteWhen() has been implemented.

For example,

```
SELECT_VG tp9840 when uid = archive or sitefn = 6
```

If this subroutine is unavailable, either because it was not implemented or because the sitelib.so library is not accessible, the expression using sitefn is evaluated as being false. Therefore, the example above would be treated as if it were the following:

```
SELECT_VG tp9840 when uid = archive or false
Or:
SELECT_VG tp9840 when uid = archive
```

If a policy stanza contains multiple references to sitefn, it is possible that the subroutine is only called once and the value returned by that call may be used for several substitutions of sitefn. Therefore, a policy that contains the following will not necessarily call the subroutine three times:

```
-1
                        0
                                 when sitefn < 10
AGE_WEIGHT
AGE_WEIGHT
                1
                       .1
SPACE_WEIGHT
                                 when sitefn != 11
                1
                      1e-6
SPACE_WEIGHT
                2
                       1e-9
                                 when sitefn > 19
SPACE_WEIGHT
                3.14 1e-12
```

The subroutine can issue log messages in some circumstances and user messages in others. You can call DmaGetContextFlags() to determine what kind of messages are possible.

The prototype is as follows:

dmacontext Refers to the context established when sitelib.so

was loaded

fstat Specifies the DmuFullstat_t of the file being

evaluated.

fhandle Specifies the DmuFhandle_t of the file being evaluated.

fromtyp Indicates what kind of policy is being evaluated.

Helper Subroutines for sitelib.so

This section describes optional subroutines that may be called from sitelib.so and are present in the processes that load sitelib.so.

DmaConfigStanzaExists()

The DmaConfigStanzaExists() subroutine checks whether a specified stanza exists in the DMF configuration file.

Note: Values in the configuration file may change while DMF is running.

The prototype is as follows:

```
DmaBool_t
```

DmaConfigStanzaExists(

const DmaContext_t dmacontext,
const char *type,
const char *stanza);

dmacontext Specifies the DmaContext_t parameter passed as input

to all site-defined policy subroutines, such as

SitePutFile().

Specifies the type of the stanza being checked.

Specifies the name of the stanza being checked.

For example, if the DMF configuration file contained the following:

```
define /dmf1
```

TYPE filesystem

POLICIES space_policy vg_policy

enddef

Then the following call would return true:

```
DmaConfigStanzaExists(dmacontext, "filesystem","/dmf1");
```

DmaGetConfigBool()

The DmaGetConfigBool() subroutine extracts parameter values of type DmaBool_t from the specified stanza in the DMF configuration file. If there is no such parameter definition or if it exists but with a missing or improper value, then the default is used.

Note: Values in the configuration file may change while DMF is running.

The prototype is as follows:

dmacontext Specifies the DmaContext_t parameter passed as input

to all site-defined policy subroutines, such as

SitePutFile().

stanza Specifies the name of the stanza being searched.

param Specifies the name of the parameter for which

DmaGetConfigBool() is searching.

default_val Specifies the value to use if param is not found in

stanza or if param has a missing or invalid value.

DmaGetConfigFloat()

The DmaGetConfigFloat() subroutine extracts parameter values of type float from the specified stanza in the DMF configuration file. If there is no such parameter definition or if it exists but with a missing or invalid value, the default is used.

Note: Values in the configuration file may change while DMF is running.

The prototype is as follows:

float

DmaGetConfigFloat(

```
const DmaContext_t dmacontext,
const char *stanza,
const char *param,
    float default_val,
    float min,
    float max);
```

The parameters are as follows:

dmacontext Specifies the DmaContext_t parameter passed as input

to all site-defined policy subroutines, such as

SitePutFile().

stanza Specifies the name of the stanza being searched.

param Specifies the name of the parameter for which

DmaGetConfigFloat() is searching.

default_val Specifies the value to use if param is not found in

stanza or if param has a missing or invalid value.

min Defines the minimum valid value.

max Defines the maximum valid value.

DmaGetConfigInt()

The DmaGetConfigInt() subroutine extracts parameter values of type int64_t from the specified stanza in the DMF configuration file. If there is no such parameter definition or if it exists but with a missing or invalid value, then a default value is used.

Note: Values in the configuration file may change while DMF is running.

The prototype is as follows:

```
int64_t
```

DmaGetConfigInt(

```
const DmaContext_t dmacontext,
const char *stanza,
const char *param,
    int64_t default_val,
    int64_t min,
    int64_t max);
```

The parameters are as follows:

dmacontext Specifies the DmaContext_t parameter passed as input

to all site-defined policy subroutines, such as

SitePutFile().

Specifies the name of the stanza being searched.

param Specifies the name of the parameter for which

DmaGetConfigInt() is searching.

default_val Specifies the value to use if param is not found in

stanza or if param has a missing or invalid value.

min Defines the minimum valid value.

max Defines the maximum valid value.

DmaGetConfigList()

The DmaGetConfigList() subroutine returns a pointer to an array of words found in the parameter in the specified stanza. The items value points to a block of memory containing an array of string pointers are well as the strings themselves; the end of the array is marked by a NULL pointer. The block of memory has been allocated by the malloc() subroutine and can be released with the free() subroutine if desired. The caller is responsible for releasing this memory.

Note: Values in the configuration file may change while DMF is running.

The prototype is as follows:

The parameters are as follows:

dmacontext Specifies the DmaContext_t parameter passed as input

to all site-defined policy subroutines, such as

SitePutFile().

stanza Specifies the name of the stanza being searched.

param The name of the parameter for which

DmaGetConfigList() is searching.

items Specifies an output value that points to a block of

memory containing an array of string pointers as well

as the strings themselves; the end of the array is marked by a NULL pointer.

DmaGetConfigStanza()

The DmaGetConfigStanza() subroutine return a pointer to an array of parameters and values for the specified stanza in the DMF configuration file. (That is, it provides the entire stanza, after comments have been removed.) The items value points to a block of memory containing an array of structures with string pointers as well as the strings themselves; the end of the array is marked by a NULL pointer. The block of memory has been allocated by the malloc() subroutine and can be released with the free() subroutine if desired. The caller is responsible for releasing this memory.

Note: Values in the configuration file may change while DMF is running.

The prototype is as follows:

The parameters are as follows:

dmacontext Specifies the DmaContext_t parameter passed as input

to all site-defined policy subroutines, such as

SitePutFile().

stanza Specifies the name of the stanza being searched.

items Specifies an output value that points to a block of

memory containing an array of structures with string

pointers as well as the strings themselves; the end of the array is marked by a NULL pointer.

DmaGetConfigString()

Extracts a string from the specified stanza in the DMF configuration file and returns it. If there is no such parameter definition, the default is used. If the parameter exists but with a missing value, the null string (which is a valid value) is returned.

Note: Values in the configuration file may change while DMF is running.

The prototype is as follows:

void

```
DmaGetConfigString(
```

```
const DmaContext_t dmacontext,
const char *stanza,
const char *param,
const char *default_val,
DmuStringImage_t &result);
```

The parameters are as follows:

dmacontext	specifies the DmaContext_t	parameter	passed as inp	ut
------------	----------------------------	-----------	---------------	----

to all site-defined policy subroutines, such as

SitePutFile().

stanza Specifies the name of the stanza being searched.
param Specifies the name of the parameter for which

DmaGetConfigString() is searching.

default_val Specifies the value to use if param is not found in

stanza. If param is found in stanza but has a

missing value, the null string is returned.

result Specifies an output parameter, containing the result.

DmaGetContextFlags()

The DmaGetContextFlags() determines if a given subroutine can issue log messages or issue user messages.

Note: If DmaFlagContextValid() is not set in the return value, no use should be made of any other bits.

 $\label{lem:decomposition} {\tt DmaGetContextFlags()} \ can \ return \ the \ following \ values, \ which \ may \ be \ OR'd \ together:$

DmaFlagContextValid Indicates that the context is valid.

DmaFlagLogAvail Indicates that DmaSendLogFmtMessage may be called.

DmaFlagMsgAvail Indicates that DmaSendUserFmtMessage may be

called.

The prototype is as follows:

```
uint64_t
```

DmaGetContextFlags(

const DmaContext_t dmacontext);

The parameter is as follows:

dmacontext Specifies the DmaContext_t parameter passed as input

to all site-defined policy subroutines, such as

SitePutFile().

DmaGetCookie()

The DmaGetCookie() subroutine returns the cookie that was stored in dmacontext by a call to DmaSetCookie(). If a NULL value is returned, either the context is invalid or the cookie was not set.

The prototype is as follows:

The parameter is as follows:

dmacontext Specifies the DmaContext_t parameter passed as input

to all site-defined policy subroutines, such as

SitePutFile().

DmaGetDaemonVolGroups()

The DmaGetDaemonVolGroups() subroutine returns the volume groups and MSPs that the dmfdaemon is currently configured to use.

Note: Values in the configuration file may change while DMF is running.

The prototype is as follows:

The parameter is as follows:

dmacontext Specifies the DmaContext_t parameter passed as input

to all site-defined policy subroutines, such as

SitePutFile().

DmaGetProgramIdentity()

The DmaGetProgramIdentity() subroutine returns a pointer to the program DmaIdentity_t object in the dmacontext parameter.

Note: The program Dmaldentity_t object should not be confused with the user Dmaldentity_t object that is returned by "DmaGetUserIdentity()" on page 278. The user identity is usually of much more interest when applying site policies because it defines who is actually making the request as opposed to what process is negotiating the site policies.

007–3681–019

The prototype is as follows:

The parameter is as follows:

dmacontext

Specifies the DmaContext_t parameter passed as input to all site-defined policy subroutines, such as SitePutFile().

DmaGetUserIdentity()

The DmaGetUserIdentity() subroutine returns a pointer to the user DmaIdentity_t object in the dmacontext parameter.

The user DmaIdentity_t object contains as much information as could be reliably gathered regarding the identity of the originator of the request. For example, the user identity in the SitePutFile() policy subroutine would identify the process (such as dmput) that made the original DmuPutByPathSync() libdmfusr call.

If DmaGetUserIdentity() is called from within SiteKernRecall(), it will return the identity of dmfdaemon. The identity of the user who initiated the read request that caused SiteKernRecall() to be called is unknown to DMF.

Within SiteCreateContext(), the user details may be as yet unknown; therefore, DmaGetUserIdentity() may return different values than if it is called with the same context from another site-defined policy subroutine. In most cases, the user identity is determined after the call to SiteCreateContext().

Under certain circumstances, some elements of the Dmaldentity_t structure may be unknown. For example, if a site-defined subroutine is called as a result of a command entered on a client machine running a release prior to DMF 3.1, some elements of the user identity may be unknown.

The prototype is as follows:

The parameter is as follows:

dmacontext Specifies the DmaContext_t parameter passed as input

to all site-defined policy subroutines, such as

SitePutFile().

DmaOpenByHandle()

The DmaOpenByHandle() subroutine opens a file by fhandle, in O_RDONLY mode, and returns the file descriptor.

If there is an error, -1 is returned. It is the caller's responsibility to close the file descriptor.

Note: This subroutine is implemented on IRIX only.

The prototype is as follows:

int

DmaOpenByHandle(

const DmaContext_t dmacontext,
const DmuFhandle_t *dmuhanp);

The parameters are as follows:

dmacontext Specifies the DmaContext_t parameter passed as input

to all site-defined policy subroutines, such as

SitePutFile().

dmuhanp Specifies the DmuFhandle_t of the file to be opened.

DmaSendLogFmtMessage()

The DmaSendLogFmtMessage() subroutine formats and issues log messages, if log messages are possible. The messages will potentially appear in the calling program's log depending upon the DmaLogLevel_t of the message and the log level selected by the administrator in the DMF configuration file. If log messages are not possible, DmaSendLogFmtMessage() silently discards the message.

The prototype is as follows:

const char

The parameters are as follows:

. . .);

dmacontext Specifies the DmaContext_t parameter passed as input

*format,

to all site-defined policy subroutines, such as

SitePutFile().

log_level Specifies the level of the message.

name Specifies a string that is included as part of the log

message.

format Specifies the format for the message that will be printed

in the log. It looks like a printf(3S) format. Do not include \n as part of the message. If you want to print more than one line to the log, make multiple calls to

DmaSendLogFmtMessage().

For example, the following will issue an error message to the calling program's log:

DmaSendUserFmtMessage()

The DmaSendUserFmtMessage() subroutine formats and sends messages to the user, if user messages are possible. The messages will potentially appear as output from commands such as dmput and dmget, depending upon the severity of the message and the level of message verbosity selected by the user. If user messages are not possible, DmaSendUserFmtMessage() silently discards the message.

The prototype is as follows:

```
void
```

DmaSendUserFmtMessage(

The parameters are as follows:

dmacontext Specifies the DmaContext_t parameter passed as input

to all site-defined policy subroutines, such as

SitePutFile().

severity Specifies the severity of the message.

position Specifies a string that can be included in the message.

This string may be set to NULL.

err_no Specifies that if err_no is non-zero, the results of

strerror(err_no) will be included in the message.

format Specifies the format for the message that will be sent to

the user. It looks like a printf(3S) format. It is not necessary to put a \n at the end of the message.

DmaSetCookie()

The DmaSetCookie() subroutine stores a pointer to site-defined subroutine information in dmacontext. This pointer may be retrieved by a call to DmaGetCookie(). The site-defined subroutines are responsible for memory management of the space pointed to by the cookie parameter.

The prototype is as follows:

```
void
DmaSetCookie(
```

const DmaContext_t dmacontext,
 void *cookie);

dmacontext Specifies the DmaContext_t parameter that is passed

as input to all site-defined policy subroutines, such as

SitePutFile().

cookie Specifies a pointer to information that sitelib.so

subroutines want to retain while the dmacontext is

valid.

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 or 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 E-1 on page 285 provides a summary of key differences between the two operating systems as they relate to DMF.

Table E-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 command.	dmscanfs(8) command.
DMF database administration	dmdalter and dmdbase commands.	dmdadm(8) command, which has an administrator interface similar to that of the dmcatadm(8) and dmvoladm(8) commands.

007–3681–019

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.
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.

Third-Party Backup Package Configuration

The following third-party backup packages are known to be DMF-aware:

- "LEGATO NetWorker" on page 287
- "Atempo Time Navigator" on page 288
- "Time Navigator for NDMP" on page 289

LEGATO NetWorker

A DMF-aware Application Specific Module (ASM) called dmfasm is available as a patch against NetWorker 7.1.2 for IRIX.

To use NetWorker to back up DMF-managed filesystems, add each filesystem to the NetWorker client's save set list and enable dmfasm on each filesystem.

Note: Only root can restore migrated files because DMF uses an extended (system) attribute owned by root.

You can enable the dmfasm module by creating a file named .nsr in the root directory of each DMF-managed filesystem. The contents of this file should be the following, which specifies that dmfasm should be used on all files and subdirectories:

+dmfasm:

Note: As of NetWorker 7.1.2, the nwbackup and nwrecover commands do not include dmfasm, and therefore backups and recovers performed with those commands will not be DMF-aware. Only the save, savepnpc, and recover commands use dmfasm.

An alternative method for enabling dmfasm on DMF-managed filesystems is to create a directive resource using nwadmin. For example, with two DMF filesystems /dmfusr1 and /dmfusr2, the directive resource would contain the following:

```
<< /dmfusr1 >> +dmfasm: .
<< /dmfusr2 >> +dmfasm: .
```

007–3681–019

After creating the directive, you must update the NetWorker client's Directive field to use the new directive.

See the NetWorker documentation for more information about ASMs, .nsr files, and directives.

To use DMF's do_predump.sh script with NetWorker, set up the NetWorker client to use a precommand as follows:

- 1. Set the client's Backup command field to savepnpc.
- 2. Create a file named /nsr/res/grpname.res, where grpname is the NetWorker group to which the client belongs. The file should contain the following:

```
type: savepnpc;
precmd: "/usr/lib/dmf/do_predump.sh daemon dump_tasks";
```

where:

- daemon is the name of the dmdaemon object in the DMF configuration file
- *dump_tasks* is the name of the task group specifying parameters related to backups

Note: DMF's DUMP_RETENTION parameter should match the value of the NetWorker client's Retention Policy parameter.

For more information about Networker, see www.legato.com and the Networker manuals.

Atempo Time Navigator

Atempo's Time Navigator is high-performance backup and recovery software designed with intuitive graphical user interfaces (GUIs) to manage data in heterogeneous environments.

Time Navigator is DMF-aware and supports a broad range of servers and client operating systems including SGI IRIX and 64-bit Linux running on Intel Itanium 2 processors. It also supports a wide range of SAN hardware and tape libraries. Time Navigator by default uses Atempo's proprietary Time Navigator protocol for all data transfers.

To make Time Navigator aware of a DMF filesystem, add a line resembling the following to the *full-Time-Navigator-installation-path*/Conf/parameters file, where /dmfusr is the DMF user filesystem:

```
parameter:bapi_fs=/dmfusr
```

You can specify more DMF filesystems by adding a similar line for each DMF filesystem.

Using the Time Navigator GUI, you can define **backup classes** to select which directories you want to back up. You can also vary the granularity for backup and restore, such as file, directory, or class level.

Because Time Navigator is DMF-aware, you can also use NDMP to back up and restore data on DMF filesystems.

To use DMF's do_predump.sh script with Time Navigator, set up Time Navigator to use a precommand as follows:

• In the **Advanced** settings of the backup strategy, specify the following as the preprocessing command:

```
/usr/lib/dmf/do_predump.sh daemon dump_tasks
```

where:

daemon Name of the dmdaemon object in the DMF configuration file

dump_tasks Name of the task group specifying the parameters related to backups

• Ensure that DMF's DUMP_RETENTION parameter matches the retention value of the cartridge pool associated with backing up the DMF filesystem.

For more information about Time Navigator, see www.atempo.com and the Time Navigator manuals.

Time Navigator for NDMP

Time Navigator supports Network Data Management Protocol (NDMP), which is an open network protocol built around client/server technology. NDMP provides a single interface for controlling backup, recovery, and other data transfers between primary and secondary storage devices in a variety of heterogeneous environments.

007–3681–019

NDMP consists of two services allowing hardware and software to communicate regardless of their own characteristics:

- The Data Server reads data from disk and generates an NDMP data stream or reads the NDMP data stream and restores it back to disk.
- The *Tape Server* reads an NDMP data stream and writes it to tape or reads from tape and creates an NDMP data stream.

NDMP allows for library and drive management and enables efficient use of network resources. It also addresses a major problem with network-attached storage (NAS) devices, which usually cannot host backup software because most of them do not run general purpose operating systems and cannot run applications. With Time Navigator, all that is required is an NDMP data server on the NAS system.

Backup using Time Navigator for NDMP is based on the NDMP three-way architecture:

- *Time Navigator NDMP Data Server:* installed on the host on which the data to be backed up resides.
- *Time Navigator NDMP Tape Server:* installed on the host that controls data transfer and is connected to a tape device.
- *Time Navigator Client application for NDMP:* coordinates the Data Server and Tape Server using NDMP. The filesystem or directory to be backed up must be visible to the Time Navigator Client application. If the data to be backed up resides on a different host, you must NFS-mount the required filesystems on this system in order to be able to browse and create the backup class.

Figure F-1 describes the NDMP three-way architecture.

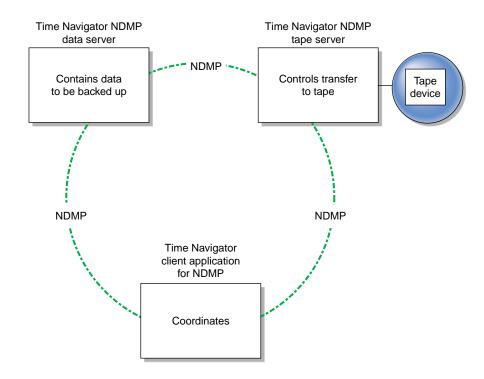


Figure F-1 NDMP Three-way Architecture

Time Navigator bundles the Data Server and the Tape Server in one service called *NDMP Server*, which is installed on both the host acting as the Data Server and the host acting as the Tape Server.

For more information on NDMP, visit www.ndmp.org. For other configurations possible with Time Navigator for NDMP, see the Time Navigator manuals.

007–3681–019

Glossary

accelerated access to first byte

A partial-state file feature capability that allows you to access the beginning of an offline file before the entire file has been recalled.

active database entry

A valid daemon database entry.

See also soft-deleted database entry and hard-deleted database entry

ADMIN EMAIL

The e-mail address to receive output from administrative tasks. See "Configuring the Base Object" on page 42.

AGE_WEIGHT

A floating-point constant and floating-point multiplier to use to calculate the weight given to a file's age. See "File Weighting and MSP or Volume Group Selection Parameters" on page 64.

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.

007-3681-019 293

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.

BFID

See bit-file identifier.

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 bit-file identifier.

BFID-set state

The sum of the states of the components that comprise a bit-file identifier set: the file state of any user file and the state of any database entries (incomplete, complete, soft-deleted, or active).

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.

BUFFERED_IO_SIZE

The size of I/O requests when reading from a filesystem using buffered I/O. See "Configuring Filesystems" on page 58.

CACHE AGE WEIGHT

The floating-point constant and floating-point multiplier to use to calculate the weight given to a file's age. See:

- "File Weighting and MSP or Volume Group Selection Parameters" on page 64
- "DCM Policies" on page 75

CACHE_DIR

The directory in which the volume group stores chunks while merging them from sparse tapes. See "LS Objects" on page 76.

CACHE_SPACE

The amount of disk space (in bytes) that dmatls can use when merging chunks from sparse tapes. See "LS Objects" on page 76.

CACHE_SPACE_WEIGHT

The floating-point constant and floating-point multiplier to use to calculate the weight given to a file's size. See:

- "File Weighting and MSP or Volume Group Selection Parameters" on page 64
- "DCM Policies" on page 75

candidate list

A list that contains an entry for each file in a filesystem eligible for migration, or for a file or range of a files that are eligible to be made offline. This list is ordered from largest file weight (first to be migrated) to smallest. This list is generated and used internally by dmfsmon(8). The dmscanfs(8) command prints similar file status information to standard output.

CAT record

A catalog (CAT) record in the library server database that tracks which migrated files reside on which tape volumes.

CHILD_MAXIMUM

The maximum number of child processes that the media-specific process is allowed to fork. See "Setting Up a Disk MSP" on page 106.

007–3681–019

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.

COMMAND

The binary file to execute in order to initiate a media-specific process or library server. See:

- "LS Objects" on page 76
- "Setting Up an FTP MSP" on page 101
- "Setting Up a Disk MSP" on page 106

complete MSP or volume group daemon-database entry

An entry in the daemon database whose path field contains a key returned by its media-specific process (MSP) or volume group, indicating that the media-specific process or volume group maintains a valid copy of the user file.

compression

The mechanism by which data is reduced as it is written to tape.

configuration object

A series of parameter definitions in the DMF configuration file that controls the way in which DMF operates. By changing the parameters associated with objects, you can modify the behavior of DMF.

configuration parameter

A string in the DMF configuration file that defines a part of a configuration object. By changing the values associated with these parameters, you can modify the behavior of DMF. The parameter serves as the name of the line. Some parameters are reserved words, some are supplied by the site.

daemon

A program that is run automatically by the system for a specific purpose.

daemon database

A database maintained by the DMF daemon. This database contains such information as the bit-file identifier, the media-specific process or volume group name, and media-specific process or volume group key for each copy of a migrated file.

daemon object

The configuration object that defines parameters necessary for dmfdaemon(8) operation.

DASD

See direct-access storage device.

DATABASE_COPIES

One or more directories into which a copy of the DMF databases will be placed. See "Configuring Daemon Maintenance Tasks" on page 50.

data-pointer area

The portion of the inode that points to the file's data blocks.

device object

The configuration object that defines parameters for the DMF backup scripts' use of tape devices other than those defined by a drive group.

DCM

See disk cache manager.

direct-access storage device

An IBM disk drive.

DIRECT_IO_MAXIMUM_SIZE

The maximum size of I/O requests when using O_DIRECT I/O to read from any primary filesystem or when migrating files down the hierarchy from the STORE_DIRECTORY of a disk media-specific process running in DCM mode. See "Configuring the Base Object" on page 42.

DIRECT_IO_SIZE

The size of I/O requests when reading from this filesystem using direct I/O. See "Configuring Filesystems" on page 58.

disk cache

Data on secondary storage.

disk cache manager

The feature that lets you configure the disk media-specific process to manage data on secondary storage, allowing you to further migrate the data to tape as needed.

DG

See drive group.

DMF daemon

The program that accepts requests to migrate data, communicates with the operating system kernel in order to maintain a file's migration state, determines the destination of migrated data, and requests the return of offline copies.

DMF state

See file state.

drive group

(DG) One of the components of an library server. The drive group is responsible for the management of a group of interchangeable tape drives located in the tape library. These drives can be used by multiple volume groups and by non-DMF processes, such as backups and interactive users. However, in the latter cases, the drive group has no management involvement; the mounting service (TMF or OpenVault) is responsible for ensuring that these possibly competing uses of the tape drives do not interfere with each other. The main tasks of the drive group are to monitor tape I/O for errors, to attempt to classify them (as volume, drive, or mounting service problems), and to take preventive action.

DRIVE GROUPS

One or more drive groups containing drives that the library server can use for mounting and unmounting volumes. See "LS Objects" on page 76.

dual-resident file

In DCM, a cache-resident copy of a migrated file that has already been copied to tape, and can therefore be released quickly in order to prevent the cache filling, without any need to first copy it to tape (analogous to a *dual-state file*).

DUALRESIDENCE_TARGET

The percentage of filesystem capacity that DMF maintains as a reserve of dual-state files whose online space can be freed if free space reaches or falls below FREE_SPACE_MINIMUM. See:

- "Automated Space Management Parameters" on page 63
- "DCM Policies" on page 75

dual-state file

A file whose data resides both online and offline.

dual-state filesystem

A filesystem that has the necessary inode space to support dual-state files.

DUMP_DATABASE_COPY

The path to a directory where a snapshot of the DMF databases will be placed when do_predump.sh is run. See "Configuring Daemon Maintenance Tasks" on page 50.

DUMP_DEVICE

The name of the drive group in the configuration file that defines how to mount the tapes that the dump tasks will use. See "Configuring Daemon Maintenance Tasks" on page 50.

DUMP_FILE_SYSTEMS

One or more filesystems to dump. If not specified, the tasks will dump all the DMF-managed user filesystems configured in the configuration file. See "Configuring Daemon Maintenance Tasks" on page 50.

DUMP_FLUSH_DCM_FIRST

Specifies whether or not the dmmigrate command is run before the dumps are done to ensure that all non-dual-resident files in the DCM caches are migrated to tape. See "Configuring Daemon Maintenance Tasks" on page 50.

DUMP_INVENTORY_COPY

The pathnames of one or more directories into which are copied the XFS inventory files for the backed-up filesystems. See "Configuring Daemon Maintenance Tasks" on page 50.

DUMP_MAX_FILESPACE

A maximum disk space used for files to be dumped, which may be larger or smaller than the length of the file. See "Configuring Daemon Maintenance Tasks" on page 50.

DUMP_MIGRATE_FIRST

Specifies whether or not the dmmigrate command is run before the dumps are done to ensure that all migratable files in the DMF-managed user filesystems are migrated, thus reducing the number of tapes needed for the dump and making it run much faster. See "Configuring Daemon Maintenance Tasks" on page 50.

DUMP_RETENTION

The length of time that the backups of the filesystem will be kept before the tapes are reused. See "Configuring Daemon Maintenance Tasks" on page 50.

DUMP_TAPES

The path of a file that contains tape volume serial numbers (VSNs), one per line, for the dump tasks to use. See "Configuring Daemon Maintenance Tasks" on page 50.

DUMP_VSNS_USED

A file in which the VSNs of tapes that are used are written. See "Configuring Daemon Maintenance Tasks" on page 50.

DUMP_XFSDUMP_PARAMS

Passes parameters to the xfsdump program. See "Configuring Daemon Maintenance Tasks" on page 50.

EXPORT_QUEUE

Instructs the daemon to export details of its internal request queue to SPOOL_DIR/daemon_exports every two minutes, for use by dmstat(8) and other utilities. See "Configuring the DMF Daemon" on page 46.

fhandle

See file handle.

file

An inode and its associated data blocks; an empty file has an inode but no data blocks.

file handle

The DMAPI identification for a file. You can use the dmscanfs(8), dmattr(1), and dmfind(1) commands to find file handles.

file state

The migration state of a file as indicated by the dmattr(1) command. A file can be regular (not migrated), migrating, dual-state, offline, partial-state, unmigrating, never-migrated, or have an invalid DMF state.

file tag

A site-assigned 16-bit integer associated with a specific file allowing the file to be identified and acted upon.

```
FREE_DUALRESIDENT_FIRST
```

Specifies whether or not dmfsmon will first free dual-state and partial-state files before freeing files it must migrate. See "DCM Policies" on page 75.

```
FREE_DUALSTATE_FIRST
```

Specifies whether or not dmfsmon will first free dual-state and partial-state files before freeing files it must migrate. See "Automated Space Management Parameters" on page 63.

FREE SPACE DECREMENT

The percentage of filesystem space by which dmfsmon will decrement FREE_SPACE_MINIMUM (if it cannot find enough files to migrate) so that the value is reached. The decrement is applied until a value is found that dmfsmon can achieve. See "Automated Space Management Parameters" on page 63.

FREE_SPACE_MINIMUM

The minimum percentage of free filesystem space that dmfsmon maintains. dmfsmon will begin to migrate files when the available free space for the filesystem falls below this percentage value. See "Automated Space Management Parameters" on page 63.

FREE_SPACE_TARGET

The percentage of free filesystem space that dmfsmon will try to achieve if free space reaches or falls below FREE_SPACE_MINIMUM. See "Automated Space Management Parameters" on page 63.

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.

FTP_DIRECTORY

The directory to use on the remote system. See "Setting Up an FTP MSP" on page 101.

FTP_HOST

The Internet hostname of the remote machine on which files are to be stored. See "Setting Up an FTP MSP" on page 101.

FTP MSP

The daemon-like media-specific process (MSP) that copies data blocks onto alternate media and assigns keys to identify the location of the migrated data using the file transfer protocol (FTP) to transfer to and from disks of another system on the network.

FTP PASSWORD

The file containing the password to use when migrating files to the remote system. This file must be owned by root and be only accessible by root. See "Setting Up an FTP MSP" on page 101.

FTP_USER

The user name to use when migrating files to the remote system. See "Setting Up an FTP MSP" on page 101.

hard-deleted database entry

A media-specific process or volume group database entry that has been removed from the daemon database and whose media-specific process or volume group copy has been discarded. See also *active database entry* and *soft-deleted database entry*.

HOME_DIR

The base pathname used to construct directory names for DMF directories in which databases and related files reside. See "Configuring the Base Object" on page 42.

inode

The portion of a file that contains the bit-file identifier, the state field, and the data pointers.

incomplete MSP or volume group daemon-database entry

An entry in the daemon database for a media-specific process (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.

JOURNAL DIR

The base pathname used to construct directory names for DMF directories in which the daemon and library server database journal files will be written. See "Configuring the Base Object" on page 42.

JOURNAL RETENTION

The length of time to keep journals. See "Configuring Daemon Maintenance Tasks" on page 50.

JOURNAL_SIZE

The maximum size (in bytes) of the database journal file before DMF closes it and starts a new file. See "Configuring the Base Object" on page 42.

library server

(LS) The daemon-like process by which data blocks are copied onto tape and that maintains the location of the migrated data. Each library server has an associated catalog (CAT) and volume (VOL) database. A library server can be configured to contain one or more drive groups. Each drive group contains one or more volume groups. A volume group is responsible for copying data blocks onto alternate media. A volume group is capable of managing a single copy of a user file.

LICENSE_FILE

The full pathname of the file containing the license used by DMF. See "Configuring the Base Object" on page 42.

LOG_RETENTION

The length of time to keep log files. See "Configuring Daemon Maintenance Tasks" on page 50.

LS

See library server

LS_NAMES

The library servers used by the DMF daemon. See "Configuring the DMF Daemon" on page 46.

MAX CACHE FILE

The largest chunk (in bytes) that will be merged using the merge disk cache. See "LS Objects" on page 76.

MAX_MANAGED_REGIONS

The maximum number of managed regions that DMF will assign to a file on a per-filesystem basis. You can set MAX_MANAGED_REGIONS to any number that is less than the actual number of regions that will fit in a filesystem attribute. See "Configuring Filesystems" on page 58.

MAX_VIRTUAL_MEMORY

The maximum number of bytes to which the DMF daemon's virtual memory size is allowed to grow. See "Configuring the DMF Daemon" on page 46.

media-specific process

(MSP) The daemon-like process by which data blocks are copied onto alternate media and that assigns keys to identify the location of the migrated data.

merging

See volume merging.

MESSAGE_LEVEL

The highest message level number that will be written to a log file (the higher the number, the more messages written). See:

- "Configuring the DMF Daemon" on page 46
- "Configuring Filesystems" on page 58
- "LS Objects" on page 76
- "General Message Log File Format" on page 114

migrated file

A file that has a bit-file identifier and whose offline copies are completed. Migrated files can be *dual-state* or *offline*.

migrating file

A file that has a bit-file identifier but whose offline copies are in progress.

007-3681-019 305

MIGRATION_LEVEL

The highest level of migration service allowed. See "Configuring the DMF Daemon" on page 46, "Configuring Filesystems" on page 58.

MIGRATION_TARGET

The percentage of filesystem capacity that DMF maintains as a reserve of dual-state files whose online space can be freed if free space reaches or falls below FREE_SPACE_MINIMUM. See "Automated Space Management Parameters" on page 63.

MIN_DIRECT_SIZE

Determines whether direct or buffered I/O is used when reading from this filesystem (see O_DIRECT in the open(2) man page for a description of direct I/O). See "Configuring Filesystems" on page 58.

MOVE_FS

The scratch filesystem used by dmmove(8) to move files between media-specific processes or volume groups. See "Configuring the DMF Daemon" on page 46.

MSP

See media-specific process.

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 media-specific process or volume group where to locate the copy of the file's data.

MSP_NAMES

Names the media-specific processes used by the DMF daemon. See "Configuring the DMF Daemon" on page 46.

MSP object

The configuration object that defines parameters necessary for that media-specific process's operation. There is one media-specific process object for each media-specific process.

MOUNT_SERVICE

The mounting service to use. Supported values are openvault and tmf. See:

- "Configuring Device Objects" on page 57
- "Drive Group Objects" on page 78

MOUNT_SERVICE_GROUP

The name by which the object'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 OpenVault drivegroup name that would be listed by the ov_drivegroup command. See "Configuring Device Objects" on page 57, "Drive Group Objects" on page 78.

nonmigrated file

A file that does not have a bit-file identifier or any offline copies. See regular file.

offline file

A file whose inode contains a bit-file identifier but whose disk blocks have been removed. The file's data exists elsewhere in copies on alternate media.

offline pointer

In media-specific process and volume group processing, a character string that the media-specific process or volume group returns to the daemon to indicate how a file is to be retrieved.

OpenVault

A storage library management facility that improves how applications can manage, store, and retrieve removable media.

orphan chunk

An unused area in a library server catalog (CAT) database entry resulting from the removal of migrated files.

orphan database entry

An unused database entry resulting from the removal of a migrated file during a period in which the DMF daemon is not running.

OV_ACCESS_MODES

A list of access mode names that control how data is written to tape. See "Configuring Device Objects" on page 57.

OV_INTERCHANGE_MODES

A list of interchange mode names that control how data is written to tape. See "Configuring Device Objects" on page 57.

OV_KEY_FILE

The file containing the OpenVault keys used by DMF. See "Configuring the Base Object" on page 42.

OV_SERVER

The name returned by the hostname(1) command on the machine on which the OpenVault server is running. See "Configuring the Base Object" on page 42.

parameter

See configuration parameter.

partial-state file

A file that has more than one region. DMF allows a file to include up to four distinct file regions. See also *region*.

partial-state file online retention

A partial-state file feature capability that allows you to keep a specific region of a file online while freeing the rest of it (for example, if you wanted to keep just the beginning of a file online). See also *partial-state file*.

partial-state file recall

A partial-state file feature capability that allows you to recall a specific region of a file without recalling the entire file. For more information, see the dmput(1) and dmget(1) man pages. See also *partial-state file*.

PARTIAL_STATE_FILES

Enables or disables the DMF daemon's ability to produce partial-state files. See "Configuring the DMF Daemon" on page 46.

POLICIES

The names of the configuration objects defining policies for this filesystem. See:

- "Configuring Filesystems" on page 58
- "Setting Up a Disk MSP in DCM Mode" on page 110

policy

Rules that tell DMF how to determine media-specific process or volume group selection, automated space-management policies, and/or file weight calculations

policy object

The configuration object that specifies parameters to determine media-specific process or volume group selection, automated space management policies, and/or file weight calculations in automatic space management.

raw time

The time in seconds since January 1, 1970.

recall

To request that a migrated file's data be moved back (unmigrated) onto the filesystem disk, either by explicitly entering the dmget(1) command or by executing another command that will open the file, such as the vi(1) command.

RECALL_NOTIFICATION_RATE

The approximate rate, in seconds, at which regions of a file being recalled are put online. This allows for access to part of a file before the entire file is recalled. See "Configuring the DMF Daemon" on page 46.

region

A contiguous range of bytes that have the same residency state. The states can be DUALSTATE, OFFLINE, MIGRATING, or UNMIGRATING.

regular file

DMF considers a regular file to be one with no bit-file identifier and no offline copies.

removable media

A tape cartridge, a tape reel, an optical disc, a digital linear tape, a removable magnetic disk, or a videotape.

RUN_TASK

A DMF maintenance command to be executed. See:

- "Automated Maintenance Tasks" on page 36
- "Configuring Daemon Maintenance Tasks" on page 50

SCAN FAST

The parameter that specifies whether or not DMF will use fast scanning. If set to no, the dmscanfs command will use its recursive option, which is much slower but results in pathnames being included in the output file. See "Configuring Daemon Maintenance Tasks" on page 50.

SELECT_LOWER_VG

The volume groups to use for migrating a file. (It is not used for defining which volume group to use for recalls; for that, see the definitions of the LS_NAMES, MSP_NAMES, DRIVE_GROUPS, and VOLUME_GROUPS parameters.) See:

- "File Weighting and MSP or Volume Group Selection Parameters" on page 64
- "DCM Policies" on page 75

SELECT MSP

The media-specific processes to use for migrating a file. See "File Weighting and MSP or Volume Group Selection Parameters" on page 64.

SELECT_VG

The volume groups to use for migrating a file. See "File Weighting and MSP or Volume Group Selection Parameters" on page 64.

site-defined policy

A site-specific library of C++ functions that DMF will consult when making decisions about its operation.

snapshot

The information about all bit-file identifier sets that is collected and analyzed by dmaudit(8). The snapshot analysis is available from the report function.

soft-deleted database entry

A daemon database entry for which the media-specific process 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.

SPACE_WEIGHT

The floating-point constant and floating-point multiplier to use to calculate the weight given to a file's size. See "File Weighting and MSP or Volume Group Selection Parameters" on page 64.

sparse tape

A tape containing only a small amount of active information.

special file

A device file in UNIX or Linux. (DMF never migrates special files.)

SPOOL_DIR

The base pathname used to construct the directory names for DMF directories in which DMF log files are kept. See "Configuring the Base Object" on page 42.

state field

The field in the inode that shows the current migration state of a file.

STORE_DIRECTORY

The directory used to hold files for a media-specific process. See "Setting Up a Disk MSP" on page 106.

tape block

See block.

tape drive

A hardware device that reads and writes data to tape.

tape chunk

See chunk.

tape merging

See volume merging.

task

A process initiated by the DMF event mechanism. Configuration tasks that allow certain recurring administrative duties to be automated are defined with configuration file parameters.

TASK_GROUPS

The task groups containing tasks that the daemon or library server should run. See:

- "Configuring the DMF Daemon" on page 46
- "Configuring Daemon Maintenance Tasks" on page 50
- "Configuring Filesystems" on page 58
- "LS Objects" on page 76

THRESHOLD

The percentage of active data on a tape. DMF will consider a tape to be sparse when it has less than this percentage of data that is still active. See "Configuring Daemon Maintenance Tasks" on page 50.

TMF_TMMNT_OPTIONS

Command options that should be added to the tmmnt command when mounting a tape. See "Configuring Device Objects" on page 57.

TMP_DIR

The base pathname used to construct the directory names for DMF directories in which DMF puts temporary files such as pipes. See "Configuring the Base Object" on page 42.

TYPE

The required name for the object. See:

- "Configuring the Base Object" on page 42
- "Configuring the DMF Daemon" on page 46
- "Configuring Device Objects" on page 57
- "Configuring Filesystems" on page 58
- "Automated Space Management Parameters" on page 63
- "LS Objects" on page 76

VOLUME_GROUPS

The volume groups containing volumes that can be mounted on any of the drives within this drive group. See "Drive Group Objects" on page 78.

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 bit-file-identifier (BFID) set state that consists of one or more soft-deleted daemon database entries, either incomplete or complete. There is no user file.

voiding the BFID

The process of removing the bit-file identifier (BFID) from the user file inode and soft-deleting all associated database entries.

VOL record

A volume (VOL) record in the library server database that contains information about a tape volumes that exists in the pool of tapes used by the library server.

volume group

(VG) 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 library server for copying active data from volumes that contain largely obsolete data to volumes that contain mostly active data. Also known as *tape merging*.

WATCHER

The resource watcher that the library server should run. See "LS Objects" on page 76.

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

 \mathbf{A}

A	relationship of targets, 124 selection of migration candidates, 123
absolute block positioning, 11	automounters, 11
accelerated access to first byte, 28	
ADMIN_EMAIL, 43	В
\$ADMINDIR directory	D
daemon maintenance tasks, 52	backup package configuration, 196, 287
MSP maintenance tasks, 98	backup package configuration, 190, 207 backups
administrative tasks	DMF and backup products, 191
automated maintenance tasks, 36	of daemon database, 54
daemon configuration, 50	base object
filesystem backups, 16, 54	configuration, 42
maintenance and recovery, 189	terminology, 41
overview, 14	batch processing, 14
tape management, 97	bfid, 136, 286
age expression, 66	bit-file identifier (BFID), 12
AGE_WEIGHT, 65, 267	BLOCK_SIZE, 78
ALGORITHM, 87	blocks, 145
allocation group, 10	blocksize keyword, 166
ALLOCATION_GROUP, 84	BUFFERED_IO_SIZE, 58, 110
ALLOCATION_MAXIMUM, 84	DOTTERED_IO_SIZE, 30, 110
ALLOCATION_MINIMUM, 84	
AMPEX DIS/DST, 78	С
application data flow, 2	C
architecture, 12	CACHE_DIR, 76, 153
Atempo Time Navigator, 288	CACHE_SPACE, 77, 153
autolog log file, 114, 125	CANCEL message, 178
automated maintenance tasks	candidate list
daemon configuration, 50	creation, 121
overview, 36	generation, 122
automated space management	terminology, 15
administration duties, 15	candidates for migration
candidate list generation, 122	file exclusion, 122
commands overview, 22	file selection, 123
configuration, 63	relationship of space management targets, 124
file exclusion, 122	capacity of DMF, 13
log file, 114, 125	r,

007-3681-019 315

CAT database	FREE_SPACE_MINIMUM, 123
backup, 202	FREE_SPACE_TARGET, 123
message format comparison, 207, 208	FTP MSP, 101
message interpretation, 209	HOME_DIR, 145
CAT records	initial, 40
dmatls database, 144	JOURNAL_DIR, 137, 145, 149
LS database directories, 147	JOURNAL_SIZE, 137, 150
terminology, 23	LS set up, 100
catalog records	MIGRATION_TARGET, 123
See "CAT records", 23	MSP or volume group selection, 65, 73
cflags, 158	msp_tasks object, 97
checkage, 132	objects, 20, 41
checktime, 132, 136	OpenVault mounting service, 92
CHILD_MAXIMUM, 101, 107	overview, 29
chkconfig, 40	parameters, 20
chunkdata, 158	policy object, 62
chunklength, 158	requirements, 31
chunknumber, 158	space management parameters, 123
chunkoffset, 158	SPOOL_DIR, 125, 136, 145
chunkpos, 158	verifying, 113
chunks, 145	Configure button, 38
chunksleft keyword, 166	context manipulation subroutines, 232
CIFS, 3	converting from an IRIX DMF to a Linux DMF, 182
client and server subsystems, 3	copy file requests, 236
client-only user commands, 5	count directive, 130, 155, 163
clients supported, 4	cpio file recall, 192
COMMAND, 77, 101, 107	create directive, 130, 155, 163
commands, 17, 20	customizable policies
Common Internet File System (CIFS), 3	See "site-defined policies", 27
configuration	customizing DMF, 26
automated space management, 63	CXFS, 3
base object, 42	
binary file installation, 30	
command overview, 20	D
daemon object configuration, 46	
daemon_tasks, 51	daemon
DCM, 110	commands overview, 20
disk MSP, 106	configuration parameters, 46
dump_tasks, 54	configuring automated maintenance tasks, 50
file weighting, 64, 70	dmd_db.dbd, 203
filesystem object configuration, 58	log file, 114
FREE SPACE DECREMENT, 124	

logs and journals, 136	configuration, 110
object, 41, 46	disk MSP and, 180
processing, 127	filesystems and, 200
shutdown, 128	terminology, 7
startup, 127	delete directive, 130, 155, 164
daemon database	deleteage, 132
automated verification task, 53	deletetime, 132, 136
automating copying for reliability, 54	Dependencies button, 39
backup, 202	dev/inode (UNICOS difference), 286
directory location, 129	device object, 41, 57
dmdadm and, 129	DIRECT_IO_MAXIMUM_SIZE, 43
message format comparison, 207, 208	DIRECT_IO_SIZE, 58, 111
record length, 33, 34	directories
recovery, 203, 204	not migrated by DMF, 18
selection, 202	directory structure prior to DMF 2.8, 283
daemon_tasks, 50, 51	disk cache manager
data integrity	See "DCM", 180
administrative tasks and, 16	disk MSP, 178
copying filesystem data, 54	configuration, 106
overview, 11	log files, 180
data reliability	request processing, 179
administrative tasks and, 16	terminology, 7
copying daemon database, 54	verification, 181
copying filesystem data, 54	disk space capacity, 6
Data Server for NDMP, 290	distributed commands, 213
DATA_LIMIT, 99	DLT, 78
database journal files, 139	DmaConfigStanzaExists(), 269
DATABASE_COPIES, 54	DmaContext_t, 259
databases	DmaFrom_t, 260
daemon, 203	DmaGetConfigBool(), 270
dmcatadm message interpretation, 209	DmaGetConfigFloat(), 271
dmvoladm message interpretation, 211	DmaGetConfigInt(), 272
example of recovery, 204	DmaGetConfigList(), 273
LS recovery, 203	DmaGetConfigStanza(), 274
message format for comparisons, 207, 208	DmaGetConfigString(), 275
See "daemon database", 33	DmaGetContextFlags(), 276
dataleft keyword, 167	DmaGetCookie(), 276
datalimit, 169	DmaGetDaemonVolGroups(), 277
datawritten keyword, 167	DmaGetProgramIdentity(), 277
dbrec.dat file, 203	DmaGetUserIdentity(), 278
dbrec.keys file, 203	DmaIdentity_t, 260
DCM	•

DmaLogLevel_t, 262	format keyword, 134
DmaOpenByHandle(), 279	format keywords, 132
DMAPI requirement, 5	limit keywords, 134
DmaRealm_t, 262	selection expression, 131
DmaRecallType_t, 262	summary, 21
DmaSendLogFmtMessage(), 279	text field order, 136
DmaSendUserFmtMessage(), 280	dmdalter (UNICOS difference), 285
DmaSetCookie(), 281	dmdate, 24
dmatls	dmdbase (UNICOS difference), 285
journal files, 149	dmdbcheck, 16, 21, 23
library server terminology, 7	dmdbrecover, 21, 203
log files, 150	dmdidle, 21
LS operations, 144	dmdlog log file, 114, 127, 136
dmatrc, 11, 144	dmdskfree, 24
dmatread, 23, 144, 174	dmdskmsp, 7, 178
dmatsnf, 23, 144, 175	dmdskvfy, 24, 181
dmattr, 5, 17	dmdstat, 21
dmatvfy, 23	dmdstop, 21, 114, 128
dmatwc, 11, 144	dmdu, 17
dmaudit	dmdump
summary, 21	summary, 24
verifymsp, 175	text field order, 173
dmcatadm	dmdumpj, 24
directives, 155	DMF user library
example of list directive, 161	See "user library (libdmfusr.so)", 213
field keywords, 158	DMF-aware backup packages, 196, 287
interface, 154	dmf.conf, 20
keywords, 157	dmfasm, 287
limit keywords, 160	dmfdaemon, 21, 127
summary, 23	dmfill, 24, 202
text field order, 162	dmfind, 5, 17, 286
dmcheck, 21	dmfsfree, 22, 121
dmclripc, 24	dmfsmon, 22, 63, 121-123
dmcollect, 24	dmftpmsp, 7, 101, 176
dmconfig, 20	dmget, 5, 17
dmcopy, 5, 17	dmhdelete, 19, 21
dmd_db journal file, 137	dmhit (UNICOS difference), 285
dmd_db.dbd, 203	dmlocklog log file, 114
dmdadm	dmlockmgr, 25, 139
directives, 129, 130	abort, 141
example of list directive, 135	communication and log files, 139
field keywords, 132	-

database journal files, 139	DmuCopyAsync(), 236
interprocess communication, 140	DmuCopyRange_t, 224
log file, 114	DmuCopyRanges_t, 225
transaction log files, 139, 141	DmuCopySync(), 236
dmls, 5, 17, 286	DmuCreateContext(), 233
dmmaint, 38, 283	DmuDestroyContext(), 235
Configure button, 38	DmuErrHandler_f, 226
Dependencies button, 39	DmuErrInfo_t, 226
Inspect button, 39	DmuError_t, 227
License Info button, 40	DmuEvents_t, 227
News button, 39	DmuFhandle_t, 227
summary, 25	DmuFullRegbuf_t, 228
tasks, 29	DmuFullstat_t, 228
Update License button, 40	DmuFullstatByFhandleAsync(), 238
dmmaint utility, 6	DmuFullstatByFhandleSync(), 238
dmmigall (UNICOS difference), 286	DmuFullstatByPathAsync(), 238
dmmigrate, 192	DmuFullstatByPathSync(), 238
file backup, 192	DmuFullstatCompletion(), 249
summary, 22	DmuGetByFhandleAsync(), 243
dmmove, 25, 181	DmuGetByFhandleSync(), 243
scratch filesystem location	DmuGetByPathAsync(), 243
MOVE_FS, 47	DmuGetByPathSync(), 243
dmmvtree, 25	DmuGetNextReply(), 250
dmofrq (UNICOS difference), 285	DmuGetThisReply(), 251
dmov_keyfile, 25, 94	DmuPutByFhandleAsync(), 240, 265
dmov_loadtapes, 25, 96	DmuPutByFhandleSync(), 240, 265
dmov_makecarts, 25, 95	DmuPutByPathAsync(), 240, 265
dmput, 5, 17	DmuPutByPathSync(), 240, 265
dmscanfs, 22, 123, 285	DmuRegion_t, 229
dmselect, 25, 181	DmuRegionbuf_t, 229
dmsnap, 22	DmuReplyOrder_t, 230
dmsnap (UNICOS difference), 286	DmuReplyType_t, 230
dmsort, 25	DmuSettagByFhandleAsync(), 245
dmstat, 26	DmuSettagByFhandleSync(), 245
dmtag, 17	DmuSettagByPathAsync(), 245
DmuAllErrors_t, 218	DmuSettagByPathSync(), 245
DmuAttr_t, 219	DmuSeverity_t, 231
DmuAwaitReplies(), 248	DmuVolGroup_t, 231
DmuByteRange_t, 220	DmuVolGroups_t, 231
DmuByteRanges_t, 220	dmversion, 22
DmuChangedDirectory(), 234	dmvoladm
DmuCompletion_t, 224	

directives, 163	Time Navigator, 289
examples of list directive, 170	DUMP_TAPES, 57
field keywords, 166	dump_tasks, 54
format keywords, 169	dump_tasks object, 50
limit keywords, 169	DUMP_VSNS_ÚSED, 57
select directive, 153	DUMP_XFSDUMP_PARAMS, 57
summary, 23	Dynamic Shared Object library, 10
text field order, 173	, , ,
dmxfsrestore, 26	
do_predump.sh	E
NetWorker, 288	
snapshot location, 55	entries keyword, 160
summary, 197	eotblockid keyword, 167
Time Navigator, 289	eotchunk keyword, 167
drive group	eotpos keyword, 167
object, 41, 78	eotzone keyword, 167
OpenVault and, 92	error reports and tapes, 97
terminology, 9	/etc/dmf/dmbase, 283
TMF tapes and, 96	/etc/dmf/dmf.conf, 115
DRIVE_GROUPS, 77	EXPORT_QUEUE, 46
DRIVE_MAXIMUM, 79, 85	extended attribute structure, 32
DRIVE_SCHEDULER, 79	
DRIVES_TO_DOWN, 79	
DSK_BUFSIZE, 107	F
DSO, 10	
dual-resident state, 181	fhandle, 286
dual-state file	file concepts, 18
file migration and, 6	file handle (UNICOS difference), 286
terminology, 18	file migration
xfsdump and, 192	automated selection of candidates, 123
dump directive, 130, 155, 164	excluding files from, 122
dump utilities, 16	overview, 18
DUMP_DATABASE_COPY, 55, 197	real-time partitions and, 125
DUMP_DEVICE, 55	relationship of space management targets, 124
DUMP_FILE_SYSTEMS, 56, 197	See "migration", 73
DUMP_FLUSH_DCM_FIRST, 56, 197	terminology, 6
DUMP_INVENTORY_COPY, 56	weighting of files, 64
DUMP_MAX_FILESPACE, 56	file recall, 19
DUMP_MIGRATE_FIRST, 56, 197	file request subroutines, 235
DUMP_RETENTION	file tagging, 26
NetWorker, 288	file weighting, 64, 70
summary, 57	

filesize keyword, 158	gid expression, 66
filesystem	GUARANTEED_DELETES, 102, 107
configuration, 58	GUARANTEED_GETS, 102, 107
conversion, 107	
DCM and, 200	
dmdskmsp, 107	Н
dmftpmsp, 102	
mount options, 32	hard-deleted files
filesystem object, 41, 58	defined, 191
FINISH message, 178	maintenance/recovery, 190
flag keywords, 169	terminology, 19
FLEXIm license	hardware and software requirements, 4
requirements, 6	help directive, 130, 155, 164
FLUSHALL message, 178	helper subroutines for sitelib.so, 269
format keyword, 134, 160	herr, 169
free space management, 6	hflags, 169
FREE_DUALSTATE_FIRST, 63	hflags keyword, 167
FREE_SPACE_DECREMENT, 63, 124	hfree, 170
FREE_SPACE_MINIMUM, 63, 123	HFREE_TIME, 85
FREE_SPACE_TARGET, 64, 123	hfull, 170
fstab, 5	hierarchical storage management, 1
FTP, 3	hlock, 170
FTP MSP, 101, 176	hoa , 170
log files, 177	HOME_DIR, 31, 43, 45, 145
messages, 178	HP ULTRIUM, 78
request processing, 176	hro, 170
terminology, 7	hsite1, 170
FTP_ACCOUNT, 101	hsparse, 170
FTP_COMMAND, 102	HTML_REFRESH, 89
FTP_DIRECTORY, 102	hvfy, 170
FTP_HOST, 102	
FTP_PASSWORD, 102	
FTP_PORT, 102	I
FTP_USER, 102	
fullstat requests, 238	IBM 03590, 78
fully migrated, 7	IBM ULT3580, 78
fully migrated file, 6	IBM ULTRIUM, 78
	IMPORT_DELETE, 102, 107
	IMPORT_ONLY, 102, 107
G	initial configuration, 40
	inode and DMF, 18
get file requests, 243	

inode size, 32	configuration, 43
Inspect button, 39	License Info button, 40
inst, 113	LICENSE_FILE, 43
installation of binary files, 30	licensing
interprocess communication (IPC), 35, 139, 140	overview, 6
introduction to DMF, 1	requirements, 6
IRIX DMF conversion to Linux, 182	lights-out operations, 14
IRIX version, 4	limit keywords
,	dmcatadm, 160
	dmvoladm command, 169
I	Linux version, 4, 5
,	list directive, 130, 155, 164
journal files	LK license, 6
configuring automated task for retaining, 53	load directive, 130, 155, 164
database, 139	lock manager, 139
dmfdaemon, 136	log files, 139, 141
LS, 149	automated space management, 125
retaining, 189	automated task for retaining, 53
summary, 16	disk MSP, 180
JOURNAL_DIR, 31, 43, 46, 137, 145	dmfdaemon, 136
JOURNAL_RETENTION, 53	FTP MSP, 177
JOURNAL_SIZE, 43, 137, 150	general format, 114
- , , ,	LS, 150
	retaining, 189
L	LOG_RETENTION, 53
	LS, 144
label keyword, 167	architecture, 9
LABEL_TYPE, 80	CAT database records, 147
LEGATO NetWorker, 287	CAT database tape records, 147
Leopard version, 4	commands, 22
libdmfadm.H, 259	configuration example, 89
libdmfcom.H, 259	database recovery, 204
libdmfusr.so, 5, 26, 27	database recovery example, 204
See "user library (libdmfusr.so)", 213	description, 143
libraries	directories, 145
sitelib.so, 255	dmatsnf, 175
library server	dmaudit verifymsp, 175
See "LS", 76	dmcatadm, 154
library versioning, 216	dmvoladm, 163
libsrv_db journal file, 149	drive scheduling, 186
libsrv_db.dbd, 148, 149, 203	error analysis and avoidance, 184
license	•

journals, 149	See "MSP", 7
log files, 150	MERGE_CUTOFF, 86, 154
object, 41	merging tapes, 99, 100
objects, 76	MESSAGE_LEVEL, 47, 60, 77, 103, 107
process, 9	messages
setup, 76	CAT database, 207, 208
status monitoring, 187	daemon database, 207, 208
tape operations, 144	FTP MSP, 178
terminology, 7	interpretation for dmcatadm, 209
VOL database records, 148	interpretation for dmvoladm, 211
VOL database records for tape, 148	log file, 114
volume merging, 153	VOL database, 208
LS database	migrated data movement between MSPs, 181
CAT records, 144	migrated file
VOL records, 148	recalling, 19
LS_NAMES, 47	terminology, 18
	migrating file, 18
	migration
M	MSP or volume group, 73
	MSP or volume group selection, 65
Mac OS X version, 4	weighting of files, 64, 70
maintenance and recovery	migration candidates
automated, 36	file exclusion, 122
cleaning up journal files, 189	file selection, 123
cleaning up log files, 189	relationship of space management targets, 124
database backup, 202–204	migration of files
dmfill, 202	overview, 18
example, 204	migration target, 121
hard-deletes, 190	MIGRATION_LEVEL, 47, 60, 108
LS database, 203, 204	MIGRATION_TARGET, 64, 123
soft-deletes, 190	MIN_DIRECT_SIZE, 60, 111
maintenance tasks	MIN_VOLUMES, 86
daemon configuration, 50	MODULE_PATH, 88
maintenance utility, 38	mount, 5
MAX_CACHE_FILE, 77, 154	DMF-managed filesystems, 32
MAX_CHUNK_SIZE, 85	MOUNT_SERVICE, 58, 80
MAX_MS_RESTARTS, 80	MOUNT_SERVICE_GROUP, 58, 80
MAX_PUT_CHILDREN, 85	MOUNT_TIMEOUT, 80
MAX_VIRTUAL_MEMORY, 48	mounting services
media concepts, 145	support for, 32
media transports, 11	MOVE_FS, 31, 47
media-specific processes	

MSG_DELAY, 80	mspkey, 133, 136
MSGMAX, 35	msplog file, 150, 180
configuring, 35	dmatls, 151
msgop, 35	LS statistics messages, 151
MSGSEG	message format, 114
configuring, 35	mspname, 132, 136
MSGSSZ	MVS_UNIT, 103
configuring, 35	_ ,
MSP	
commands, 22	N
description, 143	
disk, 178	n-tier capability, 2
dmcatadm message interpretation, 209	NAME_FORMAT, 103, 108
dmfdaemon, 144	NDMP, 289
dmvoladm message interpretation, 211	Network Data Management Protocol (NDMP), 289
FTP, 176	network file system (NFS), 3
log files, 114	NetWorker, 287
and automated maintenance tasks, 53	News button, 39
message format, 207, 208	NFS, 3
moving migrated data between MSPs, 181	nwbackup, 287
objects, 41	nwrecover, 287
tape pool	itwiecovei, 207
configuring automated task to report status, 98	0
•	O
terminology, 7	abjects 41
MSP log files	objects, 41
and automated maintenance tasks, 97	offline data management
MSP or volume group	overview, 15
configuration, 73	offline file, 6, 7, 18
selection for migrating files, 65	OpenVault, 3
MSP types, 8	enhancements, 32
MSP/LS	OpenVault for drive groups, 92
dmatread, 174	OpenVault mounting service, 43
tape	configuration, 92
setup, 100	OV_SERVER, 43
MSP/LS database	origage, 133
CAT records, 147	origdevice, 133, 136
VOL records, 144	originode, 133, 136
MSP_NAMES, 47	origname, 133, 136
msp_tasks, 97	origsize, 133, 136
msp_tasks object	origtime, 133, 136
configuration, 97	

origuid, 133, 136	quit directive, 130, 155, 164
OV_ACCESS_MODES, 58, 81	1
OV_INTERCHANGE_MODES, 58, 81	
OV_KEY_FILE, 43, 94	R
OV_SERVER, 43	
overhead of DMF, 13	RDM
oversubscription, 1	lock manager, 139
•	RDM lock manager, 139
	READ_IDLE_DELAY, 82
P	READ_TIME, 86
	readage, 158
parameter table, 115	readcount, 158
partial-state file, 18, 27	readdate, 158
enable/disable feature, 48	Readme file, 39
partial-state file online retention, 28	recall
partial-state file recall, 28	migrated files, 19
PARTIAL_STATE_FILES, 48	RECALL_NOTIFICATION_RATE, 48
PARTIAL_STATE_FILES parameter, 29	record length
pathseg.dat file, 203	daemon database, 33, 34
pathseg.keys file, 203	recordlimit, 134, 160, 169
PENALTY, 88	recordorder, 134, 160, 169
periodic maintenance tasks	recover command, 287
daemon configuration, 50	recovery
pipes (not migrated by DMF), 18	daemon database, 203, 204
POLICIES, 61	LS database, 203, 204
dmdskmsp, 109	Red Hat Linux version, 4
policies (site-defined), 26	region, 27
policy object, 41	regular file, 18
configuration, 62	REINSTATE_DRIVE_DELAY, 82
POSITION_RETRY, 81	REINSTATE_VOLUME_DELAY, 82
POSITIONING, 81	reliability
PRIORITY_PERIOD	copying daemon database
dmdskmsp, 109	configuring automated tasks, 54
private filesystem of DMF and backups, 201	repair directive, 164
ProPack version, 4	request completion subroutines, 247
put file requests, 240	request processing
PUTS_TIME, 86	disk MSP, 179
	FTP MSP, 176
	requirements, 4
Q	resource scheduler, 10, 87, 88
	resource scheduler algorithm, 10
QUANTUM, 78	

resource scheduler object, 42	S
resource watcher, 10, 89	
resource watcher object, 42	sample_sitelib.C, 256
retention of journal files, 53	sample_sitelib.mk, 256
retention of log files, 53	save command, 287
Retention Policy parameter, 288	savepnpc command, 287
robotic library, 7	SEAGATE ULTRIUM, 78
rpm, 113	select directive, 164
run_audit.sh, 50	select system call
run_audit.sh task	dmfdaemon, 128
configuration, 53	SELECT_MSP, 65, 267
run_copy_databases.sh, 16	SELECT_VG, 267
run_copy_databases.sh task, 50, 54	selection expression, 165
run_daily_report.sh, 50	serial ATA, 2
run_filesystem_scan.sh, 50, 52	Server Message Block (SMB), 3
run_full_dump.sh, 16	set directive, 130, 155, 164
run_full_dump.sh task, 50	settag file requests, 245
configuration, 55	SGI InfiniteStorage Software Platform, 29
run_hard_delete.sh, 17	SGI ProPack for Linux version, 4
run_hard_deletes.sh task, 51	shutdown, 113, 114, 141
configuration, 55	silo, 7
run_merge_stop.sh task	site-defined policies, 26, 27
configuration, 100	site-defined policy, 255, 281
run_partial_dump.sh, 16	considerations, 258
run_partial_dump.sh task, 50	DmaConfigStanzaExists(), 269
configuration, 55	DmaGetConfigBool(), 270
run_remove_journals.sh, 17, 53	DmaGetConfigFloat(), 271
run_remove_journals.sh task, 50	DmaGetConfigInt(), 272
and MSP logs, 53, 97	DmaGetConfigList(), 273
run_remove_logs.sh, 17	DmaGetConfigStanza(), 274
run_remove_logs.sh task, 50, 53	DmaGetConfigString(), 275
and MSP logs, 97	DmaGetContextFlags(), 276
run_scan_logs.sh task, 50	DmaGetCookie(), 276
configuration, 53	DmaGetDaemonVolGroups(), 277
run_tape_merge.sh, 97	DmaGetProgramIdentity(), 277
run_tape_merge.sh task	DmaGetUserIdentity(), 278
configuration, 99	DmaOpenByHandle(), 279
run_tape_report.sh, 97	DmaSendLogFmtMessage(), 279
run_tape_report.sh task	DmaSendUserFmtMessage(), 280
configuration, 98	getting started, 256
run_tape_stop.sh, 97	sitelib.so data types, 262
RUN_TASK, 78, 82, 87	

DmaContext_t, 259	space management and the DCM, 125
DmaFrom_t, 260	SPACE_WEIGHT, 65, 267
DmaIdentity_t, 260	sparse tapes, 15
DmaLogLevel_t, 262	configuration of automated merging, 99
DmaRealm_t, 262	stopping automatically, 100
DmaRecallType_t, 262	merging, 97, 153
SiteFncMap_t, 263	special files (not migrated by DMF), 18
sitelib.so subroutines	SPOOL_DIR, 31, 44, 125, 136, 145
SiteCreateContext(), 263	stdin, stdout, stderr and sitelib.so, 258
SiteDestroyContext(), 263	STK, 78
SiteKernRecall(), 264	storage used by an MSP, 200
SitePutFile(), 265	STORE_DIRECTORY, 180
SiteWhen(), 267	dmdskmsp, 109
SiteCreateContext() sitelib.so subroutine, 263	support
SiteDestroyContext() sitelib.so subroutine, 263	mounting services, 32
sitefn, 66	SUSE Linux Enterprise Server (SLES) version, 5
SiteFncMap, 257	•
SiteFncMap_t, 263	
SiteKernRecall() sitelib.so subroutine, 264	T
SITELIB parameter, 257	
sitelib.readme, 27	tape activity
sitelib.so	automated task, 98
See "site-defined policy", 255	tape maintenance task configuration, 97
SitePutFile() sitelib.so subroutine, 265	tape management
sitetag, 67	error reports, 97
SiteWhen() sitelib.so subroutine, 267	merging sparse tapes, 97, 153
size, 67	msp_tasks object, 98
SLES version, 5	Tape Management Facility (TMF), 3
small files and DMF, 199	tape merging
SMB, 3	See "volume merging", 153
snapshot, 198	tape mounting, 32
.so file, 10	tape MSP/LS
soft-deleted files, 19	dmatread, 174
definition, 191	tape reports
maintenance/recovery, 190	automated task, 98
softdeleted, 67	Tape Server for NDMP, 290
Solaris Version, 4	tapesize keyword, 167
SONY SDX, 78	tar file recall, 192
SONY SDZ, 78	task, 14
space, 67	task group
space management	objects, 41
commands overview. 22	

TASK_GROUP, 87	Update License button, 40
TASK_GROUPS, 48, 61, 78, 82, 104	updateage, 133
dmdskmsp, 109	updatetime, 133, 136
text field order	user interface commands, 17
dmvoladm, 173	user library (libdmfusr.so)
third-party backup package configuration, 196, 287	distributed commands, 213
THRESHOLD, 99	IRIX considerations, 216
threshold keyword, 167	libdmfusr.so.2 data types
Tiger version, 4	DmuAllErrors_t, 218
Time Navigator, 288	DmuByteRange_t, 220
time_expression configuration	DmuByteRanges_t, 220
daemon maintenance tasks, 52	DmuCompletion_t, 224
MSP maintenance tasks, 98	DmuCopyRange_t, 224
TIMEOUT_FLUSH, 87	DmuCopyRanges_t, 225
TMF, 3	DmuErrHandler_f, 226
enhancements, 32	DmuErrInfo_t, 226
TMF tapes, 96	DmuError_t, 227
TMF_TMMNT_OPTIONS, 58, 82	DmuEvents_t, 227
TMP_DIR, 31, 44	DmuFhandle_t, 227
tpcrdm.dat file, 148, 203	DmuFullRegbuf_t, 228
tpcrdm.key1.keys file, 148, 203	DmuFullstat_t, 228
tpcrdm.key2.keys file, 148, 203	DmuRegion_t, 229
tpvrdm.dat, 149	DmuRegionbuf_t, 229
tpvrdm.dat file, 203	DmuReplyOrder_t, 230
tpvrdm.vsn.keys, 149	DmuReplyType_t, 230
tpvrdm.vsn.keys file, 203	DmuSeverity_t, 231
transaction processing, 11	DmuVolGroup_t, 231
transports, 11	DmuVolGroups_t, 231
TYPE, 42, 46, 58, 63, 76, 78, 84, 87, 89, 101	libdmfusr.so.2 data types DmuAttr_t, 219
	library versioning, 216
	sitelib.so and, 258
U	user-accessible API subroutines for
	libdmfusr.so.2, 232
UDB (UNICOS difference), 285	context manipulation subroutines, 232
uid , 67	copy file requests, 236
ULTRIUM, 78	DmuAwaitReplies(), 248
UNICOS differences, 285	DmuChangedDirectory(), 234
UNIX special files (not migrated by DMF), 18	DmuCopyAsync(), 236
unmigrating file, 18	DmuCopySync(), 236
upage keyword, 168	DmuCreateContext(), 233
update directive, 130, 155, 164	DmuDestroyContext(), 235
update keyword, 168	

DmuFullstatByFhandleAsync(), 238	message format comparison, 208
DmuFullstatByFhandleSync(), 238	message interpretation, 211
DmuFullstatByPathAsync(), 238	VOL database records, 144, 149
DmuFullstatByPathSync(), 238	LS, 148
DmuFullstatCompletion(), 249	VOL records, 22
DmuGetByFhandleAsync(), 243	volgrp, 159
DmuGetByFhandleSync(), 243	volgrp keyword, 168
DmuGetByPathAsync(), 243	volume group, 10
DmuGetByPathSync(), 243	objects, 41
DmuGetNextReply(), 250	volume group objects, 84
DmuGetThisReply(), 251	volume merging, 11
DmuPutByFhandleAsync(), 240	configuration of automated task, 99
DmuPutByFhandleSync(), 240	stopping automatically, 100
DmuPutByPathAsync(), 240	LS, 153
DmuPutByPathSync(), 240	volume records, 22
DmuSettagByFhandleAsync(), 245	volume-to-volume merging, 153
DmuSettagByFhandleSync(), 245	VOLUME_GROUPS, 83
DmuSettagByPathAsync(), 245	VOLUME_LIMIT, 99
DmuSettagByPathSync(), 245	vsn, 159
file request subroutines, 235	vsnlist expression, 165
fullstat requests, 238	
get file requests, 243	
put file requests, 240	W
request completion subroutines, 247	
settag file requests, 245	WATCHER, 78
/usr/dmf/dmbase, 283	WEIGHT, 88
/usr/share/doc/dmf-*/info/sample, 256	weighting
	of files for migration, 64, 70
	wfage keyword, 168
<i>I</i>	wfdate keyword, 168
	when clause, 66
verification	WRITE_CHECKSUM, 83, 104
of daemon database integrity, 53	dmdskmsp, 109
verify directive, 155, 164	writeage, 159
verify disk MSPs, 181	writedate, 159
VERIFY_POSITION, 83	
version keyword, 168	
vgnames, 160	X
virtual memory size maximum, 48	
vista.taf file, 142	XFS, 3
VOL database	xfsdump, 192
backup, 202	

xfsrestore, 192 XVM snapshot, 198 zoneblockid, 159 zonenumber, 159 zonepos, 159 zones, 146

 \mathbf{Z}

ZONE_SIZE, 87