DMF Administrator's Guide for IRIX® Systems

007–3681–004

New Features for Release 2.6.3

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

Record of Revision

Version	Description
2.6.1	October 1997 Original printing to support the Data Migration Facility (DMF) release 2.6.1 running under SGI IRIX systems.
2.6.2	August 1998 Reprint with revision to support the Data Migration Facility (DMF) release 2.6.2 running under SGI IRIX systems.
2.6.2.2	December 1998 Reprint with revision to support the Data Migration Facility (DMF) update release 2.6.2.2 running under SGI IRIX systems.
004	September 1999 Reprint with revision to support the Data Migration Facility (DMF) release 2.6.3 running under SGI IRIX systems.

Contents

About This Guide

This publication documents administration of the Data Migration Facility (DMF), release 2.6.3, on SGI systems running the IRIX operating system 6.4.1, 6.5, and later releases.

Related Publications

The following documents contain additional information about DMF that may be helpful:

- *DMF Release and Installation Guide for IRIX Systems*, contains release-specific information about features and describes how to install DMF.

- *DMF Recovery and Troubleshooting Guide for IRIX Systems*, describes how to solve problems with DMF should you encounter them.

To order SGI documentation, go to the SGI Technical Publications Library at `http://techpubs.sgi.com`. Find the title that you want and choose `order` to get the ordering information page for that document.

Conventions

The following conventions are used throughout this document:

Convention	Meaning
`command`	This fixed-space font denotes literal items (such as commands, files, routines, pathnames, signals, messages, programming language structures, and e-mail addresses) and items that appear on the screen.
`manpage` (*x*)	Man page section identifiers appear in parentheses after man page names.
variable	Italic typeface denotes variable entries and words or concepts being defined.
user input	This bold, fixed-space font denotes literal items that the user enters in interactive sessions. Output is shown in nonbold, fixed-space font.

[ ]	Brackets enclose optional portions of a command or directive line.
...	Ellipses indicate that a preceding element can be repeated.

Reader Comments

If you have comments about the technical accuracy, content, or organization of this document, please tell us. Be sure to include the title and part number of the document with your comments.

You can contact us in any of the following ways:

- Send e-mail to the following address:

  `techpubs@sgi.com`

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

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

  `http://techpubs.sgi.com`

- Call the Technical Publications Group, through the Technical Assistance Center, using one of the following numbers:

  For SGI IRIX based operating systems: 1 800 800 4SGI

  For UNICOS or UNICOS/mk based operating systems or Cray Origin 2000 systems: 1 800 950 2729 (toll free from the United States and Canada) or +1 651 683 5600

- Send mail to the following address:

  Technical Publications
  SGI
  1600 Amphitheatre Pkwy.
  Mountain View, California 94043–1351

We value your comments and will respond to them promptly.

Introduction [1]

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

1.1 What Is DMF?

DMF is a hierarchical storage management system for Silicon Graphics environments. Its primary purpose is to preserve the economic value of storage media and stored data. The high I/O bandwidth of these machine environments is sufficient to overrun online disk resources. Consequently, capacity scheduling, in the form of native file system migration, has become an integral part of many computing environments and is a requirement for effective use of Silicon Graphics systems.

In addition to ensuring that adequate disk space is always available, capacity scheduling allows you to maintain a data space that is larger than your online disk resource. Oversubscription requires that the value of stored data be recognized as the same or higher than that of online data; DMF provides this capability. Figure 1 provides a conceptual overview of the data flow between applications and storage media.
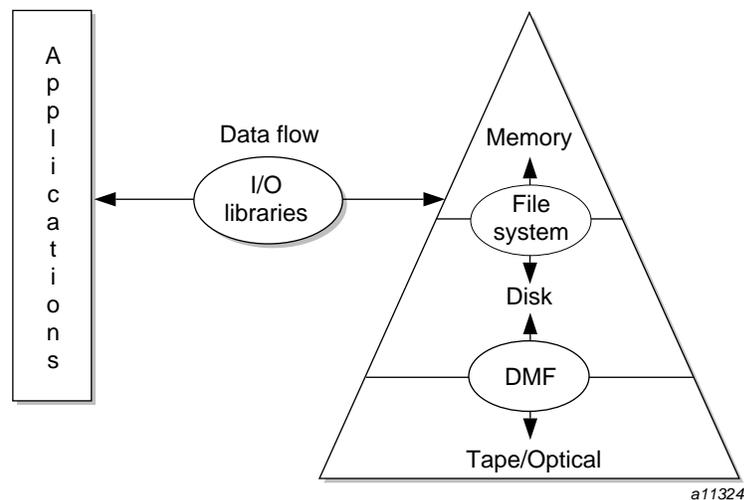
Figure 1. Application Data Flow

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

DMF interoperates with standard data export services such as Network File System (NFS) and File Transfer Protocol (FTP). By combining these services with DMF, as shown in Figure 2, you can configure a Silicon Graphics system as a high-performance file server.
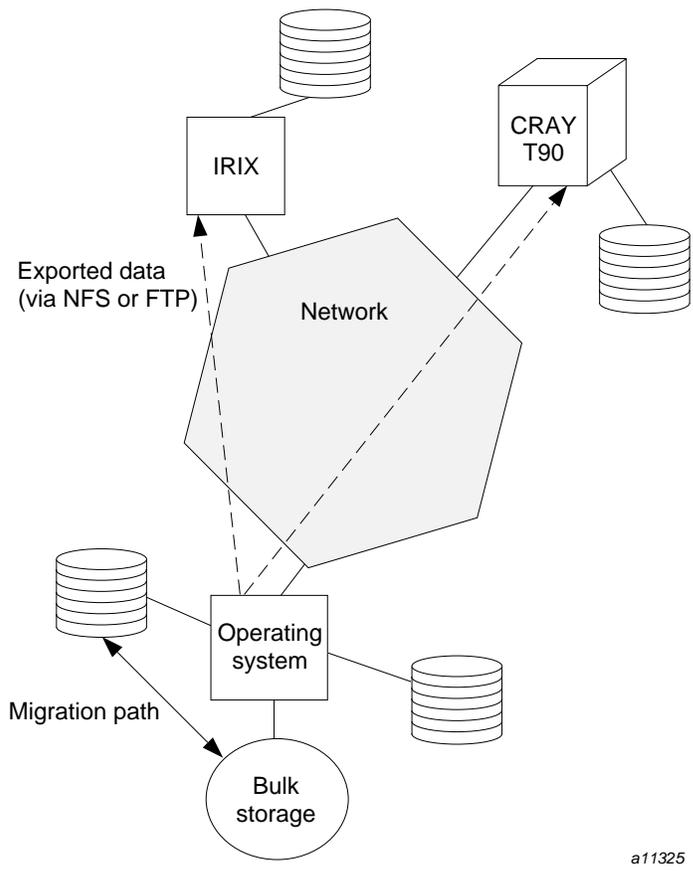
Figure 2. DMF Network Environment

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 file system migrator, DMF manages the capacity of online disk resources by transparently moving file data from disk to offline media. Most commonly, the offline medium is tape, managed by OpenVault or the Tape Management Facility (TMF). However, the offline medium can be any bulk-storage device accessible locally through NFS or FTP.

DMF accomplishes this data migration *transparently*; this means that a user cannot determine, by using POSIX-compliant commands for file system 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 file system.

1.2 How DMF Works

As a DMF administrator, you determine how disk space capacity is handled by selecting which file systems DMF will manage and by specifying the volume of free space that will be maintained on each file system. 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. First, a file is migrated to an offline medium. Once the offline copy is secure, the file is eligible to have its data blocks released (this usually occurs after a minimum space threshold is reached). A file with all offline copies completed is called *fully backed up*. A file that is fully backed up 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 file system 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; offline media management is handled by a daemon-like DMF component called the *media-specific process* (MSP). The MSP manages a pool of media volumes and moves file system data to and from offline media in response to migration requests. This component is designed to make full use of high-capacity, compressible media and to handle a

large volume of transactions. The data-recording format uses blocking and checksumming to ensure the accuracy of the data and to facilitate recovery in the event of media failure.

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

1.3 Ensuring Data Integrity

DMF provides several capabilities that enhance the safety of its operations and ensure the integrity of offline data. For example, you can configure multiple instances of the MSP, with each managing its own pool of media volumes. Therefore, DMF can be configured so that file system data is migrated to multiple offline locations.

DMF stores data that originates in an XFS file system (you can also convert other file servers to IRIX file servers running DMF). Each object stored corresponds to a file in the native file system. When a user deletes a file, the inode for that file is removed from the file system. Deleting a file that has been migrated begins the process of invalidating the offline image of that file. In the tape MSP this eventually creates a gap in the migration medium. To ensure effective use of media, the MSP 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. DMF uses the Raima Data Manager (RDM) as its database engine. This package 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 file system. DMF utilities also allow you to verify the general integrity of the DMF databases themselves.

1.4 DMF Architecture

DMF consists of the DMF daemon and one or more MSPs. The DMF daemon accepts requests from the DMF administrator or from users to migrate file system 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 identifier (called a *bit file identifier*, or *bfid*) for each file that is migrated. The daemon also determines the destination of migration data and forms requests to the appropriate MSP to make offline copies.

The MSP accepts requests from the DMF daemon. For outbound data, the MSP 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.

When running in the IRIX environment, DMF uses the Data Migration API (DMAPI) kernel interface defined by the Data Management Interface Group (DMIG). DMIG is also supported by X/Open, where it is evolving as the XDSM standard.

Figure 3 illustrates the DMF architecture.
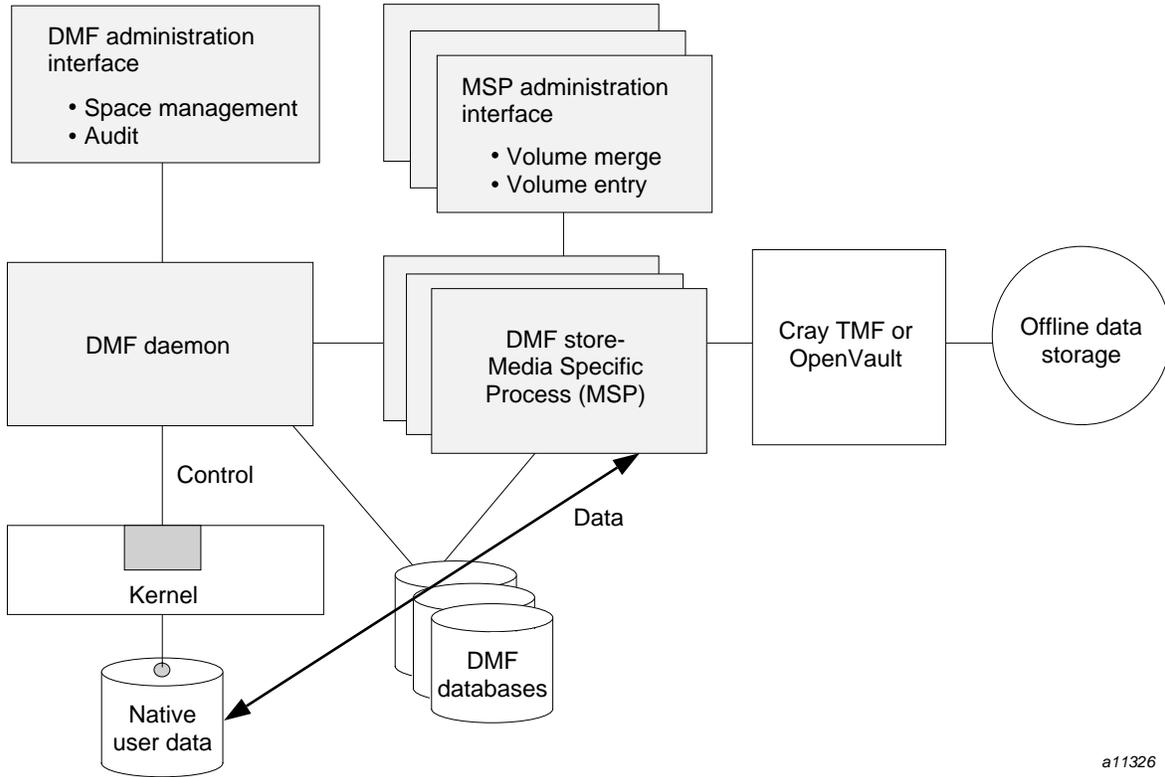
Figure 3. DMF Architecture

1.5 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 file systems or databases is performed.

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

- Total number of files. File identifiers used within DMF are 64-bit, thus providing a capacity of 2**64 files. DMF has been tested on file systems with 20 million inodes. The largest customer installation, on an inode-basis, is

approximately 5 million. The average DMF database size is approximately 1 million entries.

- Total volume of data. Capacity in data volume is limited only by the physical environment and the density of media. The largest customer installation, on the basis of data volume stored, is approximately 300 Tbytes. The average customer is storing 5 to 10 Tbytes.

- 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. The largest data-velocity customer is moving approximately 2.5 Tbytes per day.

- Storage capacity. On IRIX XFS, the largest file is 9 Tbytes.

1.6 DMF Administration

DMF can be configured for a variety of environments including dedicated file servers, lights-out operations and, most frequently, for support of batch and interactive processing in a general-purpose environment with limited disk space.

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

As a DMF administrator, you first need to characterize and determine the size of the environment in which DMF will run. You will want to plan for a certain capacity, both in the number of files and in the volume of data. You will also want to estimate the rate at which you will be moving data between the DMF store and the native file system. You will 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 Section 2.6.1, page 31, and Section 2.9.1, page 50.

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 file system that reached the threshold. Section 3.1, page 72, describes candidate list generation.

- Automated space management. You must decide how much free space to maintain on each managed file system. DMF has the ability to monitor file system capacity and to initiate file migration and the freeing of space when free space falls below the prescribed thresholds. Chapter 3, page 71, provides details about automated space management.

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

  Complex strategies are possible when using multiple MSPs. 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 selection. When setting up a tape MSP, you assign a pool of tapes for use by that MSP. The dmvoladm(8) utility provides management of the tape MSP 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 (the run_merge_tapes.sh task), the media pool is merged on a regular basis in order to reclaim unusable space.

  Recording media eventually becomes unreliable. Sometimes, media transports become misaligned so that a volume written on one cannot be read from another. Two utilities are provided that support management of failing media. The dmatsnf(8) utility is used to scan a DMF volume for flaws, and dmatread(8) is used for recovering data. Additionally, the volume merge process built into the MSP is capable of effectively recovering data from failed media.

  Chapter 6, page 93, provides more information on MSP administration.

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

  - Even though you are running DMF, you will still have to 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 file systems.

The dump utility for your file system (xfsdump and xfsrestore on IRIX systems) works in concert with DMF in that it understands when a file is fully backed up. The dump utilities have an option that allows for dumping only files that are not fully backed up.

You can establish a policy of migrating 100% of DMF-managed file systems, 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. This 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 file systems DMF manages. Much of the work done by DMF ensures that the DMF databases remain aligned with the file systems.

You can configure DMF to automatically examine the consistency and integrity of the DMF daemon and MSP 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 Section 1.8.2, page 11.

1.7 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 disk. The only difference users might notice is a delay in access time.

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) and/or dmls(1) commands

- Test in shell scripts whether a file is online or offline by using the dmattr(1) command

1.8 DMF File Concepts and Terms

DMF regards files as being one of the following:

- *Regular files* are user files residing only on disk

- *Migrating files* are files whose offline copies are in progress

- *Migrated files* can be either of the following:

  - *Dual-state files* are files whose data resides both online and offline

  - *Offline files* are files whose data is no longer on disk

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

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

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.

1.8.1 Migrating a File

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

The DMF daemon keeps a record of all migrated files in its database. The key to each file is its 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 MSP(s) are active, and sends a request to the MSP(s). MSPs copy data to the offline storage media.

When the MSP(s) have completed the offline copy (or copies), the daemon marks the file as fully backed up in its database and changes the file to dual-state. If the user specified the dmput -r option, or if dmfsmon requested that the file's space be released, the daemon releases the data blocks and changes the user file state to offline.

1.8.2 Recalling a Migrated File

When a migrated file must be recalled, a request is made to the DMF daemon. The daemon selects an MSP from its internal list of MSPs and sends that MSP a request to recall a copy of the file. If more than one MSP has a copy, the first MSP in the list is used. (The list of MSPs is created from the configuration file.)

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

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

The soft-delete state allows for the possibility that the file system might be restored after the user has removed a file. When a file system 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 so that copies of the file can be removed from media. For a tape MSP this involves compression (or merging).

1.9 Command Overview

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

1.9.1 Configuration Commands

The configuration file, /etc/dmf/dmbase/host/*hostname*/dmf_config, contains *configuration objects* and associated *configuration parameters* that control the way DMF operates. The *hostname* is the name of the host on which you installed DMF. 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, page 19. The following man pages are related to the configuration file:

Man page	Description
dmf_config(5)	Describes the DMF configuration objects and parameters in detail
dmconfig(8)	This command prints DMF configuration parameters to standard output

1.9.2 DMF Daemon and Related Commands

The DMF daemon, dmdaemon(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 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, dmdaemon 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, page 77, 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 `dmdaemon` and the daemon database:

Command	Description
dmaudit(8)	Reports discrepancies between file systems 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.
dmdaemon(8)	Starts the DMF daemon.
dmdbcheck(8)	Checks the consistency of a database by validating the location and key values associated with each record and key in the data and key files (also an MSP command). If you configure the `run_copy_database.sh` task, this command is executed automatically as part of the task. The consistency check is completed before the DMF databases are saved.
dmdbrecover(8)	Updates the daemon and tape MSP databases with journal entries.
dmdidle(8)	Causes files not yet copied to tape to be flushed to tape, even if this means forcing only a small amount of data to a volume.
dmdstat(8)	Indicates to the caller the current status of `dmdaemon`.
dmdstop(8)	Causes `dmdaemon` to shut down.
dmhdelete(8)	Deletes unused daemon database entries and releases corresponding MSP space. 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 file systems, leaving them as dual-state. This utility is often used to migrate files before running backups of a file system, hence minimizing the size of the dump image.
dmsnap(8)	Copies the DMF daemon and the MSP databases to a specified location. If you configure the `run_copy_database.sh` task, this command is executed automatically as part of the task.
dmversion(8)	Reports the version of DMF that is currently executing.

1.9.3 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 file system through automatic file migration:

Command	Description
dmfsfree(8)	Attempts to bring the free space and migrated space of a file system into compliance with configured values.
dmfsmon(8)	Monitors the free space levels in file systems configured as auto (that is, automated space management is enabled) and lets you maintain a specified level of free space.
dmscanfs(8)	Scans DMF file systems and prints status information to stdout.

See Chapter 3, page 71, for a detailed description of automated space management.

1.9.4 MSP Commands

The DMF tape MSP maintains a database that contains volume (VOL) records and catalog (CAT) records. VOL records contain information about tape

volumes, and CAT records contain information about offline copies of migrated files.

The disk and FTP MSPs allow the use of local or remote disk storage for storing migrated data. They use no special commands, utilities, or databases. For more information, see Section 6.3, page 125, and Section 6.2, page 122.

Two commands manage the CAT and VOL records for the tape MSP:

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

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

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

There are also tools that check for MSP database inconsistencies:

Command	Description
dmatvfy(8)	Verifies the MSP database contents against the dmdaemon(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.

1.9.5 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.
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 file system. This command is mainly used in conjunction with dump and restore commands to return a corrupted file system to a previously known valid state.
dmlockmgr(8)	Invokes the database lock manager. The lock manager is an independent process that communicates with all applications that use the DMF database, mediates record lock requests, and facilitates the automatic transaction recovery mechanism.
dmmove(8)	Moves copies of a migrated file's data to the specified MSPs.
dmmaint(8)	Calls the dmmaint utility, which performs DMF version maintenance and provides interfaces for licensing and initial configuration.
dmov_keyfile(8)	Creates the file of DMF OpenVault keys, ensuring that the contents of the file are semantically correct and have the correct file permissions. This command removes any DMF keys in the file for the OpenVault server system and adds new keys at the front of the file.
dmov_loadtapes(8)	Scans a tape library for volumes not imported into the OpenVault database and allows the user to select a portion of them to be used by an MSP. The selected tapes are imported into the

	OpenVault database, assigned to the DMF application, and added to the MSP's database.
dmov_makecarts(8)	Makes the tapes in one or more MSP databases accessible through OpenVault by importing into the OpenVault database any tapes unknown to it and by registering all volumes to the DMF application not yet so assigned.
dmselect(8)	Selects migrated files based on given criteria. The output of this command can be used as input to dmmove(8).
dmsort(8)	Sorts files of blocked records.
dmxfsrestore(8)	Calls the xfsrestore(1M) command to restore files dumped to tape volumes that were produced by DMF administrative maintenance scripts.

Configuring DMF [2]

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

2.1 Overview of the Configuration Steps

The steps outlined in the following procedure are required to configure DMF.

Procedure 1: Configuration Steps

1. Install DMF, ensuring that FLEXlm licensing is set up correctly. Installation is described in the *DMF Release and Installation Guide for IRIX Systems*.

   Note: You must read Section 2.2 for a description of special configuration issues regarding installation.

2. Ensure that your `PATH` and `MANPATH` environment variables are set to include DMF paths. See Section 2.3, page 24.

3. Invoke `dmmaint`(8) so that you can create or modify your configuration file. Using `dmmaint` is described in the *DMF Release and Installation Guide for IRIX Systems*.

4. Determine how you want to complete periodic maintenance tasks. See Section 2.2.5, page 22.

5. Edit the configuration file to define the base object, daemon object, the objects for daemon maintenance tasks, and objects for automated space management. See Section 2.5, page 26, through Section 2.8, page 39.

6. Define the media-specific process (MSP) objects, the object for MSP maintenance tasks, set up the MSPs, and configure your mounting service. See Section 2.9, page 46, through Section 2.11, page 65.

7. Verify the configuration with the `dmcheck`(8) script. See Section 2.12, page 68.

8. Start DMF. See Section 2.13, page 68.

2.2 Installation Considerations

This section discusses installation considerations that will affect how your system is configured.

2.2.1 File System Mount Options

The Data Management API (DMAPI) is the mechanism within IRIX and the XFS file system 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 DMF installation interface (dmmaint(8)).

> ⚠️ **Caution:** In order for file systems to be managed by DMF, they must be mounted to enable the DMAPI interface. You can do this by using the mount -o dmi command or by declaring parameter 4 in the fstab entry to be dmi. (refer to the man pages for mount or fstab). Failure to enable dmi for DMF-managed file systems will result in a configuration error.

2.2.2 Inode Size Configuration

DMF state information is kept within a file system 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 inode-resident extended attribute and non-resident extended attribute.

You should configure your file systems to ensure that the extended attribute is always inode-resident. This is done with the mkfs_xfs command. Declare the inode size to be 512 bytes using the -i size=512 option. File systems that already exist will have to be dumped, recreated, and restored. This change is not mandatory.

2.2.3 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. If the path value that is returned to the daemon by the MSP can fit into the path field of the daemon's dbrec record, DMF does not require pathseg

records. If the MSP supplies a path value that is longer than the `path` field in the `dbrec`, DMF creates one or more `pathseg` records.

The default size of the path field of the `dbrec` is 34 characters. This size allows the default paths returned by the `dmatmsp`, `dmdskmsp`, and `dmftpmsp` to fit in the `dbrec` path field as long as the user name portion of the `dmftpmsp` or `dmdskmsp` path (*username/bit_file_identifier*) is 8 characters or fewer. In almost all cases, you should not need to reconfigure the daemon database record.

The default size of the path field in the `pathseg` record is 64. For MSP path values which 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 disadvantage of using the default `path` size arises mainly in the tape MSP application in which there is a small amount of wasted space in the daemon's `dbrec` data file. By extending the default path field size to 34 (8 bytes more than the tape MSP requires), there is a resulting 5% wasted space in the daemon's `dbrec` data file. For a 10 MB `dbrec` file, this is 500 Kbytes of wasted space.

For installations that run only the tape MSP and for which the 5% wasted disk space is an important consideration, 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 which might be using the `dmftpmsp` or `dmdskmsp` with a different path-generating algorithm or any other MSP which supplies a path longer than 34 characters to the daemon).

Procedure 2: Daemon Database Record Length Configuration

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

1. If the `dmdaemon` is running, use `dmdstop`(8) to halt processing.

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

   a. cd *HOME_DIR*/daemon (*HOME_DIR* is the value of HOME_DIR returned by the `dmconfig base` command)

   b. dmdump -c . > *textfile* (*textfile* is the name of a file that will contain the text representation of the current database)

       c.   `cp dbrec* pathseg* dmd_db.dbd` *backup_dir* (*backup_dir* is the name of the directory that will hold the old version of the database)

       d.   `rm dbrec* pathseg* dmd_db.dbd`

3. `cd /etc/dmf/dmbase/lib/rdm`

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

5. Edit `dmd_db.ddl` to set the new `path` field lengths for the `dbrec` and/or `pathseg` records. For the most efficient use of disk space for the `dmatmsp`, set the `dbrec path` size to 26.

6. Regenerate the new database definition:

   `/etc/dmf/dmbase/etc/dmddlp -drsx dmd_db.ddl`

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

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

       a.   `cd` *HOME_DIR*`/daemon`

       b.   `dmdadm -u -c "load` *textfile*`"` (*textfile* was created in step 2)

9. If the daemon was running in step 1, restart it by executing `dmdaemon(8)`.

2.2.4 Interprocess Communication Parameters

Ensure that, in the operating system configuration file, the following IPC kernel configuration parameters are set equal to or greater than the default: `MSGMAX`, `MSGMNI`, `MSGSEG`, and `MSGSSZ`. The parameters are described in Appendix A of *IRIX Admin: System Configuration and Operation*, document number 007-2859.

2.2.5 Configuring Automated Maintenance Tasks

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

• Making backups (full or partial) of user file systems to tape

• Making backups of DMF databases to disk

- Removing old log files and old journal files

- Monitoring DMF logs for errors

- Running hard deletes

- Running dmaudit(8)

- Monitoring the status of tapes in tape MSPs

- Merging tapes that have become sparse (and stopping this process at a specified time)

Each of these tasks can be configured in the DMF configuration file through the use of TASK_GROUPS parameters for the DMF daemon and the tape MSP. 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 etc/admin directory in the configured DMF directory (/etc/dmf/dmbase).

The automated tasks are described in Section 2.6.1, page 31, for the daemon tasks and in Section 2.9.1, page 50, for the tape MSP.

provides a summary of the automated maintenance tasks:

Table 1. Automated Maintenance Task Summary

Task	Purpose	Parameters	Related Object Type
run_audit	Audit databases		daemon
run_copy_databases	Backup DMF databases	DATABASE_COPIES	daemon
run_remove_journals	Remove old journal files	JOURNAL_RETENTION	daemon
run_remove_logs	Remove old log files	LOG_RETENTION	daemon

Task	Purpose	Parameters	Related Object Type
`run_full_dump`	Full backup of file system(s) For restores, see `dmxfsrestore(8)`	`DUMP_DEVICE` `DUMP_INVENTORY_COPY` `DUMP_FILE_SYSTEMS` `DUMP_MIGRATE_FIRST` `DUMP_RETENTION` `DUMP_VSNS_USED` `DUMP_TAPES`	`daemon`
`run_hard_deletes`	Hard-delete files	Uses `DUMP_RETENTION`	`daemon`
`run_merge_stop`	Stop tape merges		`msp`
`run_partial_dump`	Partial backup of file system(s)	Uses parameters set for `run_full_dump`	`daemon`
`run_scan_logs`	Scan log files for errors		`daemon`
`run_tape_merge`	Merge sparse tapes	`DATA_LIMIT` `THRESHOLD` `VOLUME_LIMIT`	`msp`
`run_tape_report`	Create tape reports		`msp`

2.3 Setting `PATH` Environment Variables

To use DMF commands and DMF man pages, set your `PATH` and `MANPATH` environment variables. The DMF administrator commands and executables are installed in `/etc/dmf/dmbase/etc`; the user commands are installed in `/etc/dmf/dmbase/bin`. Man pages are installed in `/etc/dmf/dmbase/man`.

Note: If you are not familiar with setting the `MANPATH` environment variable, you should know that some paths are checked even though they are not listed by default. In other words, even though the command `echo $MANPATH` appears to indicate that no variable is defined (in `ksh` it returns no message or in `csh` it returns the message `MANPATH - Undefined variable`), certain paths are still searched for man pages. Setting the `MANPATH` environment variable as described below will supersede the fact that these paths are searched.

If `MANPATH` has not been set, you should read the `man(1)` man page to determined the paths that are checked and then include those paths in the commands below.

The following example uses sh syntax to set and display the DMF PATH environment variables:

```
# PATH=$PATH:/etc/dmf/dmbase/etc:/etc/dmf/dmbase/bin; export PATH
# MANPATH=$MANPATH:/etc/dmf/dmbase/man; export MANPATH
# env | grep PATH
MANPATH=/usr/man:/usr/share/catman:/usr/catman:/usr/local/man:/etc/dmf/dmbase/man
PATH=/usr/sbin:/usr/bsd:/sbin:/usr/bin:/bin:/etc:/usr/etc:/usr/bin/X11:/etc/dmf/dmbase/etc:
/etc/dmf/dmbase/bin
```

The following example uses csh syntax to set and display the DMF PATH environment variables:

```
% setenv PATH ${PATH}:/etc/dmf/dmbase/etc:/etc/dmf/dmbase/bin
% setenv MANPATH ${MANPATH}:/etc/dmf/dmbase/man
% env |grep PATH
MANPATH=/usr/man:/usr/share/catman:/usr/catman:/usr/local/man:/etc/dmf/dmbase/man
PATH=/usr/sbin:/usr/bsd:/sbin:/usr/bin:/bin:/etc:/usr/etc:/usr/bin/X11:/etc/dmf/dmbase/etc:
/etc/dmf/dmbase/bin
```

You can set the user command and man path names in the file /etc/profile for all ksh users and /etc/cshrc for all csh users, or provide a module for users.

2.4 Configuration Objects

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

- The *base object*, which defines path name and file size parameters necessary for DMF operation

- The *daemon object*, which defines parameters necessary for dmdaemon(8) operation

- The *file system object*, which define parameters necessary for migrating files in that file system

- The *policy objects*, which specify parameters to determine MSP selection, automated space management policies, and/or file weight calculations in automatic space management

- The *MSP objects*, which define parameters necessary for that MSP's operation

- The *device objects*, which define parameters for DMF's use of tape devices

- The *taskgroup objects*, which define parameters necessary for automatic completion of specific maintenance tasks

DMF configuration objects and parameters are also defined in the `dmf_config`(5) man page.

2.5 Configuring the Base Object

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

Parameter	Description
TYPE	`base` (type of object).
ADMIN_EMAIL	Email address to which to send output from administrative tasks. The mail may include errors, warnings, and output from any configured tasks. You cannot specify a list of addresses, so if you want multiple recipients for this mail, you must set up an alias.
HOME_DIR	Base path name used to construct directory names for DMF directories in which databases and related files reside. Generally referred to as *HOME_DIR*.
JOURNAL_DIR	Base path name used to construct directory names for DMF directories in which the daemon and tape MSP database journal files will be written. To provide the best chance for database recovery, this directory should be a separate file system and a different physical device from *HOME_DIR*. Generally referred to as *JOURNAL_DIR*.
JOURNAL_SIZE	Maximum size (in bytes) of the database journal file before DMF closes it and starts a new file.
LICENSE_FILE	Full path name of the file containing the FLEXlm license used by DMF. The default is

		/usr/dmf/dmbase/flexlm/license.dat. You should have no need to edit this parameter.
	OV_KEY_FILE	File containing the OpenVault keys used by DMF. It is usually located in *HOME_DIR* and called ovkeys. There is no default. (Use this parameter only if you are using OpenVault as your tape mounting service.)
	OV_SERVER	Name returned by the hostname(1) command on the machine on which the OpenVault server is running. This parameter only applies when OpenVault is used as the mounting service. The default value is the host name of the machine on which you are running.
	SPOOL_DIR	Base path name used to construct the directory names for DMF directories in which DMF log files are kept. Generally referred to as *SPOOL_DIR*.
	TMP_DIR	Base path name used to construct the directory names for DMF directories in which DMF puts temporary files such as pipes. It is also used by scripts for temporary files and is the directory used by default by the tape MSP for caching files if the CACHE_DIR parameter is not defined. Generally referred to as *TMP_DIR*.

Warning: Do **not** change the directory names while DMF is running.

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 Section 2.9.3, page 56.

Procedure 3: Base Object Configuration

The following example defines a base object:

```
define  base
        TYPE            base
        ADMIN_EMAIL     root@dmfserver
        HOME_DIR        /dmf/home
```

```
                        TMP_DIR          /tmp/dmf
                        SPOOL_DIR        /dmf/spool/
                        JOURNAL_DIR      /dmf/journals
                        JOURNAL_SIZE     10m
                        LICENSE_FILE     /var/flexlm/dmf_license.dat
                        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 file systems. 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 Section 2.6.1, page 31, 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 email 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 Section 2.2.5, page 22.

3. Configure the file system specified by the HOME_DIR configuration parameter (referred to as *HOME_DIR*) as a separate file system, and restrict its contents to DMF databases and relatively static files such as DMF scripts.

   DMF cannot run if *HOME_DIR* runs out of space, and such an event is more likely to happen if it is simply another directory in /usr.

4. Set TMP_DIR to be any file system 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 file system 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* file system.

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, configure the `OV_KEY_FILE` parameter to be the name of the key file that holds security information for OpenVault. For more information, see Procedure 13, page 56.

9. If you plan to run OpenVault, configure the `OV_SERVER` parameter to the name of the server that runs OpenVault. For more information, see Procedure 13, page 56.

2.6 Configuring the DMF Daemon

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

Parameter	Description
TYPE	`dmdaemon` (type of object)
MESSAGE_LEVEL	Specifies the highest message level number that will be written to the daemon log. It must be an integer between 0 and 6; the higher the number, the more messages written to the log file. The default is 2. For more information on message levels, see Section 2.14, page 68.
MIGRATION_LEVEL	Sets the highest level of migration service allowed on all DMF file systems (you can configure a lower service level for a specific file system). The value can be `none` (no migration), `user` (requests from `dmput`(1) or `dmmigrate`(8) only), or `auto` (automated space management). The default is `auto`.
MOVE_FS	Names the scratch file system used by `dmmove`(8) to move files between MSPs. There is no default.
MSP_NAMES	Names the MSPs used by this daemon. Necessary only if you have more than one MSP.
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 Section 2.6.1, page 31.

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

Procedure 4: Daemon Configuration

The following example defines a daemon object:

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

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

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

2. If you have more than one MSP, ensure that the MOVE_FS parameter is set to a file system that can accept temporary files. This must be the root of a DMAPI file system. 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 none, user, or auto. This value is the highest level you want to allow anywhere in your DMF environment. You can configure a lower level for a specific file system. none means no migration will take place on any DMF file system. user means that users/administrators can perform dmput(1) or dmmigrate(8) commands and no other migration will take place. auto means that you want automated space management on at least one DMF file system. The default is auto. See Section 2.8, page 39, for information about configuring automated space management.

4. Configure MSP_NAMES to be the names of the MSPs to be used by this daemon. You need to specify this parameter if you have more than one MSP. You will use these names when defining the MSP objects and in SELECT_MSP parameters within policies. See Procedure 10, page 49.

5. Configure the TASK_GROUPS parameter to the name(s) of the object(s) 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 file systems. 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 Section 2.6.1 for more information.

2.6.1 Configuring Daemon Maintenance Tasks

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

- Auditing databases (the run_audit.sh task)

- Scanning recent log files for errors (the run_scan_logs.sh task)

- Removing old log files (the run_remove_logs.sh task and the LOG_RETENTION parameter)

- Removing old journal files (the run_remove_journals.sh task and the JOURNAL_RETENTION parameter)

- Backing up DMF databases (the run_copy_databases.sh task and the DATABASE_COPIES parameter)

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

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

- Fully backing up DMF-managed file systems (the run_full_dump.sh task)

- Partially backing up DMF-managed file systems (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 file system 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 file system, and locations for inventory files. Table 1, page 23, provides a summary of automated maintenance tasks.

Procedure 5: 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 would configure a daemon_tasks object:

```
define  daemon_tasks
        TYPE                taskgroup
        RUN_TASK            $ADMINDIR/run_audit.sh every day \
                              at 23:00
#
        RUN_TASK            $ADMINDIR/run_scan_logs.sh at 00:01
#
        RUN_TASK            $ADMINDIR/run_remove_logs.sh every \
                              day at 1:00
        LOG_RETENTION       4w
#
        RUN_TASK            $ADMINDIR/run_remove_journals.sh every \
                              day at 1:00
        JOURNAL_RETENTION   4w
#
        RUN_TASK            $ADMINDIR/run_copy_databases.sh \
                            every day at 3:00 12:00 21:00
        DATABASE_COPIES     /save/dmf_home /alt/dmf_home
enddef
```

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

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

3. Configure the RUN_TASK parameters. DMF substitutes $ADMINDIR in the path with the actual etc/admin directory in the configured DMF directory (/etc/dmf/dmbase). When the task is run, it is given the name of the object that requested the task as the first parameter and the name of the task group (in this case daemon_tasks) as the second parameter. The task itself may use the dmconfig(8) command to obtain further parameters from either of these objects.

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

The *time_expression* defines when a task should be done. It is a schedule expression that has the following form:

[every *n period*] [at *hh*:*mm*[:*ss*] ...] [on *day* ...]

period is one of minute[s], hour[s], day[s], week[s], or month[s].

n is an integer.

day is a day of the month (1 through 31) or day of the week (sunday through saturday).

The following are examples of valid time expressions:

```
at 2:00
every 5 minutes
at 1:00 on tuesday
```

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

a. The run_audit.sh task runs dmaudit. For this task, provide a *time_expression*. If it detects any errors, the run_audit.sh task mails the errors to the email address defined by the ADMIN_EMAIL parameter of the base object (described in Section 2.5, page 26).

b. The run_scan_logs.sh task scans the DMF log files for errors. For this task, provide a *time_expression*. If the task finds any errors, it sends email to the email address defined by the ADMIN_EMAIL parameter of the base object.

c. The run_remove_logs.sh task removes logs that are older than the value you provide by specifying the LOG_RETENTION parameter. You also provide a *time_expression* to specify when you want the run_remove_logs.sh to run. In the example, log files more than 4 weeks old are deleted each day at 1:00 A.M. Valid values for

LOG_RETENTION are a number followed by m[inutes], h[ours], d[ays], or w[eeks].

The run_remove_journals.sh task removes journals that are older than the value you provide by specifying the JOURNAL_RETENTION parameter. You also provide a *time_expression* to specify when you want the run_remove_journal.sh to run. In the example, journal files more than 4 weeks old are deleted each day at 1:00 A.M. Valid values for JOURNAL_RETENTION are a number followed by m[inutes], h[ours], d[ays], or w[eeks].

> **Note:** The run_remove_journals.sh and run_remove_logs.sh tasks are not limited to the daemon logs and journals; they also clear the logs and journals for MSP(s).

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

The task copies a snapshot of the current DMF databases to the directory with the oldest copy. Integrity checks are done on the databases before the copy is saved. If the checks fail, the copy is not saved, and the task sends email to the email address defined by the ADMIN_EMAIL parameter of the base object.

Procedure 6: Configuring the dump_tasks Object

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

Do not change the script names.

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

The following example would configure a dump_tasks object:

```
define  dump_tasks
        TYPE                taskgroup
        RUN_TASK            $ADMINDIR/run_full_dump.sh on \
                              sunday at 00:01
        RUN_TASK            $ADMINDIR/run_partial_dump.sh on \
                              monday tuesday wednesday thursday \
```

```
                                 friday saturday at 00:01
         RUN_TASK                $ADMINDIR/run_hard_deletes.sh
                                  at 23:00
#
         DUMP_TAPES              HOME_DIR/tapes
         DUMP_RETENTION          4w
         DUMP_DEVICE             SILO_2
         DUMP_MIGRATE_FIRST      yes
         DUMP_INVENTORY_COPY     /save/dump_inventory
enddef
```

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

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

3. Configure the RUN_TASK parameters. See step 3 in Procedure 5, page 32,
   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 file
      systems 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 file systems 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 Section 7.3, page 130.

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

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

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

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

The DUMP_DEVICE parameter specifies the name of the device object in the configuration file that defines how to mount the tapes that the dump tasks will use. See Section 2.9.2, page 53, for information about device objects.

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

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

The DUMP_FILE_SYSTEMS parameter specifies one or more file systems to dump. If not specified, the task dumps all the file systems configured in the configuration file. Use this parameter only if your site needs different dump policies (such as different dump times) for different file systems. It is safest not to specify a value for this parameter and therefore dump all file systems configured.

The DUMP_VSNS_USED parameter is optional. It specifies the name of a file to which the tasks that dump the file systems will append the VSN,

one per line, of each volume used by xfsdump. If you don't specify this parameter, the task uses /dev/null as the file name.

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

2.7 Configuring File Systems

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

The filesystem object parameters are as follows:

Parameter	Value
TYPE	filesystem (type of object).
MESSAGE_LEVEL	Specifies the highest message level number that will be written to the automated space management log (autolog). It must be an integer between 0 and 6; the higher the number, the more messages written to the log file. The default is 2. For more information on message levels, see Section 2.14, page 68.
MIGRATION_LEVEL	Sets the level of migration service for the file system. Valid values are none (no migration), user (only user-initiated migration), or auto (automated space management). The migration level actually used for the file system is the lesser of the MIGRATION_LEVEL of the daemon object and this value. The default is auto.
POLICIES	Specifies the names of the configuration objects defining policies for this file system.
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 file systems.

Procedure 7: Configuring `filesystem` Objects

The following example defines a `filesystem` object:

```
define  /c
        TYPE                    filesystem
        MIGRATION_LEVEL         user
        POLICIES                fs_msp
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 file system 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 `none`, `user`, or `auto`. `none` means no migration will take place on this file system. `user` means that users/administrators can perform dmput(1) or dmmigrate(8) commands but no other migration will take place. `auto` means that you want automated space management on this file system.

   The default is `auto`, which means that you do not need to include this line unless you want to specify `user` or `none`. See Section 2.8 and Procedure 8, page 43, 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. The `POLICIES` parameter is used to declare one or more migration policies that will be associated with this file system. Policies are defined with `policy` objects (see Section 2.8). The `POLICIES` parameter is required; there is no default value. A policy can be unique to each DMF-managed file system, or it can be reused numerous times.

2.8 DMF Policies

A `policy` object is used to specify a migration policy. Three types of migration policies can be defined: automated space management, file weighting, and MSP selection.

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

- The POLICIES parameter for a file system must specify one and only one MSP selection policy.

- If the MIGRATION_LEVEL for a file system is auto, the POLICIES parameter for that file system 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 selection) and share that policy among all the file systems. Alternatively, you might create an MSP selection policy for all file systems and a space management policy (including weighting parameters) for all file systems.

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

2.8.1 Automated Space Management Parameters

DMF lets you automatically monitor file systems and migrate data as needed to prevent file systems 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 file system to maintain the level of free space configured (in the configuration file).

Chapter 3, page 71, describes automated space management in more detail.

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

Parameter	Description
TYPE	`policy` (type of object).
FREE_SPACE_MINIMUM	Minimum percentage of free file system space that `dmfsmon` maintains. `dmfsmon` will begin to migrate files when the available free space for the

file system falls below this percentage value. This parameter is required; there is no default.

FREE_SPACE_TARGET — Percentage of free file system space that `dmfsmon` will try to achieve if free space reaches or falls below `FREE_SPACE_MINIMUM`. This parameter is required; there is no default.

MIGRATION_TARGET — Percentage of file system 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 file system is migrated, migrating, or free after it runs to make space available. This parameter is required; there is no default.

FREE_SPACE_DECREMENT — Percentage of file system 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 between 1 and the value of `FREE_SPACE_TARGET`. The default is 2.

FREE_DUALSTATE_FIRST — When set to `on`, `dmfsmon` will free dual-state files before freeing files it will have to migrate first. The default is `off`.

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

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

2.8.2 File Weighting and MSP 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. Each MSP manages its own set of volumes. The MSP selection parameters allow you to direct DMF to migrate files with different characteristics to different MSPs.

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

The policy parameters for file weighting are as follows:

Parameter	Description
AGE_WEIGHT	Specifies a floating point constant and floating point multiplier to use to calculate the weight given to a file's age. AGE_WEIGHT is calculated as *constant* + (*multiplier* * *file_age_in_days*). If DMF cannot locate values for this parameter, it uses a floating point constant of 1 and a floating point multiplier of 1.
SPACE_WEIGHT	Specifies a floating point constant and floating point multiplier to use to calculate the weight given to a file's size. SPACE_WEIGHT is calculated as *constant* + (*multiplier* * *file_disk_space_in_bytes*). If DMF cannot locate values for this parameter, it uses a floating point constant of 0 and a floating point multiplier of 0.

The parameter for MSP selection follows:

Parameter	Description
SELECT_MSP	Specifies the MSP(s) to use for a file. You can list as many MSP names as you have MSP objects defined. A copy of the file will be migrated to each MSP listed. The special MSP name none means that the file will not be migrated. If you

define more than one MSP, separate the names with white space. If no SELECT_MSP parameter applies to a file, it will not be migrated. The parameters are processed in the order they appear in the policy. There is no default.

The file weighting and MSP selection parameters accept an optional *when clause* to restrict the set of files to which that parameter applies.

This clause has the form when *expression*.

expression can include any of the following simple expressions:

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

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

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

The following are examples of valid expressions:

```
space < 10m                (space used is less than 10 million bytes)
uid <= 123                 (file's user ID is less than or equal to 123)
gid = 55                   (file's group ID is 55)
age >= 15                  (file's age is greater than or equal to 15 days)
space > 1g                 (space used is greater than 1 billion bytes)
uid in (10 82-110 200)     (file's user ID is 10, between 82 and 110, or 200)
(gid = 55 or uid <= 123) and age < 5
                           (file's age is greater than 5 days and its
                           group ID is 55 or its user ID is higher than 123)
```

2.8.3 Configuring Policies

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

Procedure 8: Configuring Objects for Automated Space Management

The following example defines a `policy` object for automated space
management:

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

        AGE_WEIGHT 0    0.00    when age < 10
        AGE_WEIGHT 1    0.01    when age < 30
        AGE_WEIGHT 10   0.05    when age < 120
        AGE_WEIGHT 50   0.1

        SPACE_WEIGHT  0   0
enddef
```

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 file
      system 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 file system free space reaches or falls below
      `FREE_SPACE_MINIMUM`. The default is 30.

   b. Configure `FREE_SPACE_TARGET` to an integer percentage of total file
      system 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
      file system space that DMF must maintain as free. DMF will begin to
      migrate files when the available free space for the configured file
      system reaches or falls below this percentage value. The default is 10.

d. Configure `FREE_DUALSTATE_FIRST` to be `on` if you want DMF to free the space used by dual-state files before it migrates and frees regular files. The default is `off`.

4. Configure the age and size weighting factors associated with a file when it is evaluated for migration as follows:

a. The syntax of the `AGE_WEIGHT` parameter is a floating-point constant followed by a floating-point multiplier. The age weight is calculated as follows:

constant + (*multiplier* x *age_in_days*)

Add a `when` clause to select which files should use these values. DMF checks each `AGE_WEIGHT` parameter in turn, in the order they occur in the configuration file. If the `when` clause is present, DMF determines whether the file matches the criteria in the clause. If no clause is present, a match is assumed. If the file matches the criteria, the file weight is calculated from the parameter values. If they do not match, the next instance of that parameter is examined.

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

In the example policy, files that have been accessed or modified within the last 10 days have a weight of zero. File migration likelihood increases with the length of time since last access because the file will have a greater weight. The final line specifies that files which have not been accessed or modified in 120 days or more have a far greater weight than all other files.

b. The syntax of `SPACE_WEIGHT` parameters is a floating-point constant followed by a floating-point multiplier. The space weight is calculated as follows:

constant + (*multiplier* x *file_disk_space_in_bytes*)

In the example policy, the size of the file does not affect migration because all files have `SPACE_WEIGHT` of zero.

A `SPACE_WEIGHT` of `0 0.0` is used if no `SPACE_WEIGHT` applies for a file.

c. Configure negative values to ensure that files are never automatically migrated. For example, you might want to set a minimum age for

migration. The following parameter specifies that files that have been accessed or modified within 1 day are never automatically migrated:

```
AGE_WEIGHT -1    0.0    when age <= 1
```

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

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

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

Procedure 9: Configuring Objects for MSP Selection

The following example defines a `policy` object for MSP 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_MSP cart2
enddef
```

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 name (or names) you specify as the first value of the `SELECT_MSP` parameter is a name you set previously in the `MSP_NAMES` parameter of the `daemon` object. There is no default.

4. Configure MSP selection criteria as follows:

   a. If you want to select an MSP 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
   ```

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

```
SELECT_MSP cart1 cart2  when gid = 22
```
Separate multiple MSP names with a blank space.

c. If you want to ensure that some files are never migrated, you can designate the MSP 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_MSP 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 file system uses 4–Kbyte blocks, the space used by the file is 4096, and it does not match the policy line.

To ensure that files smaller than 4 Kbytes do not migrate, use the following line:

```
SELECT_MSP none when space <= 4096
```

2.9 Setting up Tape MSPs

Each MSP you create must have an object defined in the configuration file.

The tape MSP entry has the following options, listed in the order in which they appear in the sample file:

Option	Description
TYPE	msp (type of object).
COMMAND	Binary file to execute in order to initiate this MSP. For the tape MSP, this value must be dmatmsp.
CACHE_DIR	Directory in which the MSP stores chunks while merging them from sparse tapes. If you do not

specify this parameter, DMF uses the value of `TMP_DIR` from the base object.

CACHE_SPACE	Amount of disk space (in bytes) that `dmatmsp` can use when merging chunks from sparse tapes. During merging, small chunks from sparse tapes are cached on disk before being written to a tape. The default is 0, which causes all files to be merged via sockets. For more information, see Procedure 10, step 5, page 50.
CHILD_MAXIMUM	Maximum number of child processes the MSP is allowed to fork. The maximum value is 100; the default is 4.
DISK_IO_SIZE	Transfer size (in bytes) used when reading from or writing to files within a DMF file system. The value must be between 4096 and 16 million (16m). The default is 65536.
HFREE_TIME	Minimum number of seconds that a tape no longer containing valid data must remain unused before the MSP overwrites it. The default value is 172,800 seconds (2 days), and the minimum allowed value is 0.
	When an MSP removes all data from a tape, it sets the `hfree` (hold free tape) flag bit in the tape's volume (VOL) database entry to prevent that tape from being immediately reused. The next time the MSP scans the database for volumes after HFREE_TIME seconds have passed, the MSP clears the `hfree` flag, allowing the tape to be rewritten. If HFREE_TIME is set to 0, the MSP will never clear `hfree`, so an unused tape will not be reused until you clear its `hfree` flag manually. See the `dmvoladm` man page for a description of how to set and clear the `hfree` flag manually.
MAX_CACHE_FILE	Largest chunk (in bytes) that will be merged using the merge disk cache. Larger files are transferred directly via a socket from the read child to the write child. The default is 25% of the CACHE_SPACE value. Valid values are 0 through the value of CACHE_SPACE.

MAX_CHUNK_SIZE	Specifies that the MSP should break up large files into chunks no larger than this value (specified in bytes) as it writes data to tape. If a file is larger than this size, it is broken up into pieces of the specified size, and, depending on other activity, more than one write child may be used to write the data to tape. If MAX_CHUNK_SIZE is 0 (the default) the MSP only breaks a file into chunks when an end of volume is reached.
MAX_PUT_CHILDREN	Maximum number of write child processes the MSP will schedule. The default and the maximum are the value of CHILD_MAXIMUM; the minimum is 1.
MERGE_CUTOFF	Limit at which the MSP 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 is CHILD_MAXIMUM, which means that, if sparse tapes are available, children will be created until there are CHILD_MAXIMUM children, thus using tape efficiently. However, if any recall requests arrive, they will be started before new merges.
	Setting this number below CHILD_MAXIMUM reserves some tape units for recalls as the expense of merge efficiency. Setting this number above CHILD_MAXIMUM increases the priority of merges relative to recalls.
MESSAGE_LEVEL	Highest message level number that will be written to the MSP log. It must be an integer between 0 and 6; the higher the number, the more messages written to the log file. The default is 2. For more information on message levels, see Section 2.14, page 68.
MIN_TAPES	Minimum number of unused tapes that can exist in the MSP VOL database before operator notification. If the number of unused tapes falls below MIN_TAPES, the operator will be asked to add new tapes. The default is 10; the minimum is 0.

TAPE_TYPE	Specifies the name of a device object that describes how the tapes are accessed and used. There is no default. The device object is described in Section 2.9.2, page 53.
TASK_GROUPS	Names the task groups that contain tasks the MSP should run. They are configured as objects of TYPE taskgroup. There is no default. See Section 2.9.1, page 50, for more information.
TIMEOUT_FLUSH	Minutes after which the MSP will flush files to tape. The default is 120 minutes.

Procedure 10: Configuring Tape MSPs

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

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

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

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

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

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

4. Define a TAPE_TYPE parameter that names the device type object for the MSP. There is no default. Use the value you set here in defining device objects. See Section 2.9.2, page 53.

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

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

   See Section 6.1.2, page 94, for more information on tape zone sizes.

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

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

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

9. Configure the TASK_GROUPS parameter to the name(s) of the object(s) used to define how periodic maintenance tasks are completed. There is no default. See Section 2.9.1, for more information.

2.9.1 Configuring Tape MSP Maintenance Tasks

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

- Creating tape reports (the run_tape_report.sh task)

- Merging sparse tapes (the run_tape_merge.sh task and the THRESHOLD, VOLUME_LIMIT, and DATA_LIMIT parameters)

- Stopping tape merges at a specified time (the run_merge_stop.sh task)

For each of these tasks, you can configure when the task is run. For the second task, 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 logs and journals. These tasks are described in Section 2.6.1, page 31.
>
> Table 1, page 23, provides a summary of automated maintenance tasks.

Procedure 11: Configuring the `msp_tasks` Object

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

Do not change the path names or task names.

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

```
define msp_tasks
 TYPE    taskgroup
 RUN_TASK $ADMINDIR/run_tape_report.sh at 00:10
#
 RUN_TASK $ADMINDIR/run_tape_merge.sh on \
                   monday wednesday friday at 2:00
THRESHOLD          50
#VOLUME_LIMIT      20
#DATA_LIMIT        5g
#
RUN_TASK $ADMINDIR/run_merge_stop.sh at 5:00
```

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

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

3. Configure the RUN_TASK parameters. DMF substitutes $ADMINDIR in the path with the actual etc/admin directory in the configured DMF directory (/etc/dmf/dmbase). When the task is run, it is given the name of the object that requested the task as the first parameter and the name of the task group (in this case msp_tasks) as the second parameter. The task itself may use the dmconfig(8) command to obtain further parameters from either of these objects.

The RUN_TASK parameters require that you provide a *time_expression*.

The *time_expression* defines when a task should be done. It is a schedule expression that has the following form:

[every *n* *period*] [at *hh*:*mm*[:*ss*] ...] [on *day* ...]

period is one of minute[s], hour[s], day[s], week[s], or month[s].

n is an integer.

day is a day of the month (1 through 31) or day of the week (sunday through saturday).

The following are examples of valid time expressions:

```
at 2:00
every 5 minutes
at 1:00 on tuesday
```

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

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

b. The run_tape_merge.sh task merges sparse tapes. You can specify the criteria that DMF uses to determine that a tape is sparse, as follows:

    • Use the THRESHOLD parameter to set an integer percentage of active data on a tape. DMF will consider a tape to be sparse when it has less than this percentage of data that is still active.

    • Use the VOLUME_LIMIT parameter to set the maximum number of tape volumes that can be selected for merging at one time.

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

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

2.9.2 Device Objects

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

The following parameters are common to **all** device objects:

Parameter	Description
TYPE	`device` (type of object).
BLOCK_SIZE	Block size used when writing tapes from the beginning. The default depends upon the device, with DMF setting defaults as follows:

AMPEX DIS/DST	1199840
DLT	131072
STK 9840	126796
Other devices	65536

Parameter	Description
LABEL_TYPE	Label type used when writing tapes from the beginning. Possible values are `nl` (no label), `sl` (standard label, for IBM tapes), and `al` (ANSI label). The default is `al`.
MOUNT_SERVICE	Specifies the mounting service to use. Supported values are `openvault` and `tmf`. This parameter is required; there is no default.
POSITIONING	Specifies how the tape should be positioned to a zone; either `skip` or `direct`. `skip` specifies the use of tape mark skipping. `direct` specifies the use of block ID seek capability if the block ID is known. The default is `direct`.
POSITION_RETRY	Level of retry in the event of a failure during zone positioning; one of `none`, `lazy`, or `aggressive`. `lazy` specifies that the MSP will retry if a reasonably fast alternative method of positioning is available. `aggressive` specifies that the MSP may try more costly and time-consuming alternatives. If the MSP is unable to position to a zone, the MSP aborts all recalls for files with data in that zone (however, DMF does not abort them if a copy exists in another MSP). The default is `lazy`.

VERIFY_POSITION	Specifies whether the tape MSP write child should (prior to writing) verify that the tape is correctly positioned and that the tape was properly terminated by the last use. The default is to verify. Specifying `no` or `off` turns verification off; anything else ensures verification.
WRITE_CHECKSUM	Specifies that tape block should be checksummed before writing. If a tape block has a checksum, it is verified when read. The default is `on`.
ZONE_SIZE	Specifies approximately how much data the write child should put in a zone. The write child adds files and chunks to a zone until the data written equals or exceeds this value, at which time it writes a tape mark and updates the database. Smaller values allow faster recalls and better recoverability but poorer write performance. The MSP also uses zone size to determine when to start write children. The default is 50 MB.

2.9.2.1 Device Objects for OpenVault As Mounting Service

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

Parameter	Description
OV_ACCESS_MODES	Specifies a list of access mode names that control how data is written to tape. The default value is `readwrite` when migrating and `readonly` when recalling. This parameter is optional.
OV_INTERCHANGE_MODES	Specifies a list of interchange mode names that control how data is written to tape. This can be used to control whether the device compresses data as it is written. This optional parameter is applied when a tape is mounted or rewritten.

Examples of the use of these parameters are provided in Procedure 13, page 56.

OpenVault requires several configuration steps in addition to configuring the device object. They are described in Section 2.9.3, page 56.

2.9.2.2 Device Objects for TMF as Mounting Service

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

Parameter	Description
TMF_TMMNT_OPTIONS	Specifies command options that should be added to the tmmnt command when mounting a tape.
	DMF uses the -Z option to tmmnt, so options controlling block size and label parameters are ignored. Use the BLOCK_SIZE and LABEL_TYPE device parameters instead.
	Use -g if the group name is different from the device object's name. Use -i to request compression.

Procedure 12: Configuring Devices for TMF

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

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

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

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

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

3. Configure the MOUNT_SERVICE to be tmf.

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

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

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

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

2.9.3 Using OpenVault for Tape MSPs

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

Procedure 13: Configuring DMF to Use OpenVault

The following procedure describes how to make OpenVault and DMF work together. The OpenVault setup script can be used to enable the DMF application. See the *OpenVault Operator's and Administrator's Guide* for a description of this script.

Note: The example 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. To do this, use the setup script, menu item 1, submenu 5.

2. Add the DMF application as a valid user to appropriate drive groups. It is preferable that you use the OpenVault setup script, menu item 2, submenu 7. If for some reason you cannot use the setup 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 tape groups. It is preferable that you use the OpenVault `setup` script, menu item 2, submenu 8. You can enter the command manually, as follows:

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

4. Configure the `base` object for use with OpenVault:

```
define  base
        TYPE            base
        HOME_DIR        /dmf/home
.
.
.
        OV_KEY_FILE     /dmf/home/ovkeys
        OV_SERVER       hostname
enddef
```

   a. Configure the `OV_KEY_FILE` parameter name of the key file that holds security information for OpenVault. It is usually located in *HOME_DIR* and called `ovkeys`.

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

5. Use the `dmov_keyfile`(8) command to create the file defined by the `OV_KEY_FILE` parameter. This command will prompt you for the privileged and unprivileged keys that you defined in step 1.

6. Configure the `device` object for use with OpenVault:

```
define  timber
        TYPE                   device
        MOUNT_SERVICE          openvault
        OV_ACCESS_MODES        readwrite
        OV_INTERCHANGE_MODES   compression
        ZONE_SIZE              200m
enddef
```

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

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

c. Configure the `MOUNT_SERVICE` parameter to be `openvault`.

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

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

e. Configure the `OV_INTERCHANGE_MODES` parameter to be a list of interchange mode names that control how data is written to tape. This can be used to control whether the device compresses data as it is written. This parameter is optional.

To specify that you want data compressed, use
`OV_INTERCHANGE_MODES     compression`

To force all tapes to be written as DLT4000, use
`OV_INTERCHANGE_MODES     DLT4000`

This parameter is applied when a tape is first used or rewritten.

f. Configure other parameters relevant to your site. The example sets the `ZONE_SIZE` parameter to 200 MB. The target zone size is a major factor in determining how much data is written before writing a tape mark and updating the MSP database. Here, the tapes used by the `SILO_2` MSP will, in general, have more data written in a zone than DMF uses as a default. Smaller values allow faster recalls and better recovery, but they cause poorer write performance than larger values. The default is 50 MB. See Section 6.1.2, page 94, for more information on how tape zone sizes are determined.

7. Make the appropriate cartridges accessible to the MSPs by assigning the cartridges to the DMF application in OpenVault. To do this, you must know the following:

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

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

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

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

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

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

    ```
    dmov_makecarts -g cartgroup -t carttype msp
    ```

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 an MSP database by entering the following:

    ```
    dmov_loadtapes -l library -g cartgroup -t carttype msp
    ```

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

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.

2.9.4 MSP Database Records

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

You can follow the steps in Procedure 14 for all the tape MSPs you have defined.

⚠

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

Procedure 14: Creating MSP Database Records

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

1. If you have not yet done so, set your `PATH` environment variable to include `/etc/dmf/dmbase/etc`. (See Section 2.3, page 24.)

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

```
% dmvoladm -m cart1
dmvoladm: at rdm_open - created database atmsp_db
adm: 1>
```

The response is an informational message indicating that `dmvoladm` could not open an existing MSP databases, so it is creating a new and empty one. You should get this message the first time you use `dmvoladm` for an MSP, but never again. The next line is the prompt for `dmvoladm` directives.

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

   After the prompt, enter the following directive:

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

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

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

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

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

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

6. Issue the dmvoladm quit directive to complete setting up the MSP.

```
adm:3> quit
```

2.10 Setting up FTP MSPs

To enable a file transfer protocol (FTP) MSP, include a name for it on the MSP_NAMES parameter in the daemon object and define an msp object for it in the DMF configuration file.

DMF has the capability to use an FTP MSP to convert a non-DMF file server 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 file server conversion.

An FTP MSP object has the following options (defaults are provided here or in Procedure 16, page 67):

Parameter	Description
TYPE	msp (type of object).
COMMAND	Binary file to execute in order to initiate this MSP. For the FTP MSP, this value must be dmftpmsp.
CHILD_MAXIMUM	Maximum number of child processes the MSP is allowed to fork. The default is 4; the maximum is 100.
DISK_IO_SIZE	Transfer size (in bytes) used when reading from or writing to files within a DMF file system. The value must be between 4096 and 16 million (16m). The default is 65536.
FTP_ACCOUNT	Account ID to use when migrating files to the remote system.
FTP_COMMAND	Additional commands to send to the remote system. There may be more than one instance of this parameter.

FTP_DIRECTORY	Directory to use on the remote system.
FTP_HOST	Internet host name of the remote machine on which files are to be stored.
FTP_PASSWORD	File containing the password to use when migrating files to the remote system. This file must be owned by root and be only accessible by root.
FTP_PORT	Port number of the FTP server on the remote system. The default value is the value configured for ftp in the services file.
FTP_USER	User name to use when migrating files to the remote system.
GUARANTEED_DELETES	Number of child processes that are guaranteed to be available for processing delete requests. If CHILD_MAXIMUM is nonzero, its value must be greater than the sum of GUARANTEED_DELETES and GUARANTEED_GETS. The default is 1.
GUARANTEED_GETS	Number of child processes that are guaranteed to be available for processing dmget(1) requests. If CHILD_MAXIMUM is nonzero, its value must be greater than the sum of GUARANTEED_DELETES and GUARANTEED_GETS. The default is 1.
IMPORT_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.
IMPORT_DELETE	Specifies if the MSP should honor hard-delete requests from the DMF daemon. 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.

NAME_FORMAT	Remote file name template that creates names for files stored on remote machines. The default is *username/bfid* (the bfid is the full bfid in hexadecimal).
MESSAGE_LEVEL	Specifies the highest message level number that will be written to the MSP log. It must be an integer between 0 and 6; the higher the number, the more messages written to the log file. The default is 2. For more information on message levels, see Section 2.14, page 68.
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.
TASK_GROUPS	Names the task groups that contain tasks the MSP should run. They are configured as objects of TYPE taskgroup. There is no default. Currently there are tasks defined only for the tape MSP.

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

If CHILD_MAXIMUM is non-zero, 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_config(5) man page.

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

The default value for NAME_FORMAT creates a unique file name and a subdirectory on the remote machine. The subdirectory is named after the file's owner at the time of migration. 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. The unique file name is the encoded bit-file identifier (bfid) of the file.

Possible substitutes you may specify to create the NAME_FORMAT file name are as follows:

%1 substitutes for the first 32 bits of the bfid in hexadecimal

%2 substitutes for the second 32 bits of the bfid in hexadecimal

%3 substitutes for the third 32 bits of the bfid in hexadecimal

%4 substitutes for the fourth 32 bits of the bfid in hexadecimal

%b substitutes for the full bfid in hexadecimal

%u substitutes for the user name of the file owner

%U substitutes for the user ID of the file owner

%g substitutes for the group name of the file

%G substitutes for the group ID of the file

%% substitutes for the literal % character

The %1, %2, %3, %4, and %b substitutions generate uppercase hexadecimal numbers. The NAME_FORMAT must include either %b or %2, %3, %4 in some combination.

Procedure 15: Configuring the `ftp` Object

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
```

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

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

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

3. Ensure that COMMAND is set to 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.

2.11 Setting up Disk MSPs

To enable a disk MSP, include a name for it on the MSP_NAMES parameter in the daemon object and define an msp object for it in the DMF configuration file.

As with the FTP MSP, you can use a disk MSP to convert a non-DMF file server 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 file server conversion.

A disk MSP object has the following options:

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

DISK_IO_SIZE	Transfer size (in bytes) used when reading from or writing to files within a DMF file system. The value must be between 4096 and 16 million (16m). The default is 65536.
GUARANTEED_DELETES	Number of child processes that are guaranteed to be available for processing delete requests. The default is 1.
GUARANTEED_GETS	Number of child processes that are guaranteed to be available for processing dmget(1) requests. The default is 1.
IMPORT_DELETE	Applies only if IMPORT_ONLY is set to on. Set IMPORT_DELETE to on if you wish files to be deleted in STORE_DIRECTORY when hard deletes are processed.
IMPORT_ONLY	MSP is used for importing only. Set this parameter on when the data is stored as a bit-for-bit copy of the file and needs to be available to DMF as part of a conversion. The MSP will not accept dmput(1) requests when this parameter is enabled. The MSP will, by default, ignore hard delete requests when this parameter is enabled.
MESSAGE_LEVEL	Specifies the highest message level number that will be written to the MSP log. It must be an integer between 0 and 6; the higher the number, the more messages written to the log file. The default is 2. For more information on message levels, see Section 2.14, page 68.
NAME_FORMAT	Template that creates names for files in STORE_DIRECTORY. The default is *username*/*bfid* (the bfid is the full bfid in hexadecimal).
STORE_DIRECTORY	Specifies the directory used to store files for this MSP.
TASK_GROUPS	Names the task groups that contain tasks the MSP should run. They are configured as objects of

TYPE taskgroup. There is no default. Currently
there are tasks defined only for the tape MSP.

The default value for NAME_FORMAT creates a unique file name and a
subdirectory in the STORE_DIRECTORY. The subdirectory is named after the
file's owner at the time of migration. The unique file name is the encoded
bit-file identifier of the file.

Procedure 16: Configuring the `dsk` Object

The following example describes setting up a disk MSP:

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

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
   parameter of the daemon object. There is no default.

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

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

4. Set the CHILD_MAXIMUM parameter to the maximum number of child
   processes you want this MSP to be able to fork. The default is 4. The
   example allows 8.

5. Set the GUARANTEED_DELETES parameter to the number of child processes
   that are guaranteed to be available for processing delete requests. The
   default is 1. The example allows 3.

6. Set the GUARANTEED_GETS parameter to the number of child processes that
   are guaranteed to be available for processing dmget requests. The default is
   1. The example allows 3.

7. Set the STORE_DIRECTORY to the directory where files will be stored. This
   parameter is required; there is no default.

2.12 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.

2.13 Initializing DMF

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

The dmmaint(8) utility sets up system startup and shutdown scripts to start and stop DMF. You can start the DMF daemon manually by executing the dmdaemon command and stop it by executing the dmdstop(8) command.

After dmdaemon is activated, the dmget(1) and dmput(1) user commands can be used to manage file system space manually.

2.14 General Message Log File Format

The dmdaemon, dmlockmgr, dmfsmon, and MSP processes all create message files that are used to track various DMF events. These DMF message log files use the same general naming convention and message format. The message log file names are created using the extension *.yyyymmdd*, which represents the year, month, and day of log file creation.

Each line in a message log file begins with the time the message was issued, an optional message level, the process ID number, and the name of the program that issued the message.

The optional message level is described below. The remainder of the line contains informative or diagnostic information. The following sections provide details about each of these log files:

• See Section 4.3, page 85, for information about dmdaemon and dmdlog.*yyyymmdd*

- See Section 5.1, page 89, for information about `dmlockmgr` and `dmlocklog.`*yyyymmdd*

- See Section 3.3, page 75, for information about `dmfsmon` and `autolog.`*yyyymmdd*

- See Section 6.1.6, page 99, and Section 6.2.2, page 124, for information about `dmatmsp`, `dmdskmsp`, and `dmftpmsp` and `msplog.`*yyyymmdd*

- See Chapter 7, page 129, 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 as follows; the corresponding message level in the configuration file is also provided:

Table 2. DMF Log File Message Types

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

Automated Space Management [3]

The DMF file system monitor, `dmfsmon`(8), is a daemon that monitors the free space levels in file systems configured as `auto` (that is, automated space management is enabled) and lets you maintain a specified level of free space. When the free space in one of the file systems falls below the free-space minimum, `dmfsmon` invokes `dmfsfree`(8).

The `dmfsfree` command attempts to bring the free space and migrated space of a file system into compliance with configured values. The `dmfsmon` command uses `dmfsfree` to bring the free space and migrated space into compliance with configured values. `dmfsfree` may also be invoked by system administrators.

When the free space in one of the file systems falls below its free-space minimum, the `dmfsfree` command performs the following steps:

* Scans the file system for files that can be migrated and freed. Each of these candidates is assigned a file weight. This information is used to create a list, called a *candidate list*, that contains an entry for each file and is ordered by file weight (largest to smallest).

* Selects enough candidates to bring the free space back up to the desired level. Files are selected in order from largest file weight to smallest.

* Selects enough non-migrated files from the candidate list to achieve the *migration target*, which is the percentage of file system space you want to have as free space **and** space occupied by migrated but online files. Files are selected from the candidate list in order from largest file weight to smallest.

The `dmfsmon` daemon should be running whenever DMF is active. You control automated space management by setting the file system and policy configuration parameters in the DMF configuration file. The configuration parameters specify targets for migration and free-space as well as one or more policies for file weighting. Only file systems configured as `MIGRATION_LEVEL` `auto` in the configuration file are included in the space-management process. Section 2.8, page 39, describes how to configure automated space management.

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

The following sections describe space management and associated processes.

3.1 Generating the Candidate List

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

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

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

The candidate list is ordered by total file weight (largest to smallest). You can configure the weighting parameters to have a negative value and ensure that certain files are never automatically migrated.

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

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

3.2 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 file system in the DMF configuration file as described in Section 2.8, page 39.

The most essential parameters are as follows:

- FREE_SPACE_MINIMUM specifies the minimum percentage of file system space that must be free. When this value is reached, dmfsmon will take action to migrate and free enough files to bring the file system into compliance. For example, setting this parameter to 10 indicates that when less than 10% of the file system 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_config(5) man page.

- FREE_SPACE_TARGET specifies the percentage of free file system space the
  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 file system is less than 10% free and
  migrates and frees files until 15% of the file system is available.

- MIGRATION_TARGET specifies the percentage of file system 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 file system free
  space reaches or falls below FREE_SPACE_MINIMUM.

When dmfsmon detects that the free space on a file system 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 4 shows the relationship of automated space management migration
targets to each other. Migration events occur when file activity causes free file
system 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:
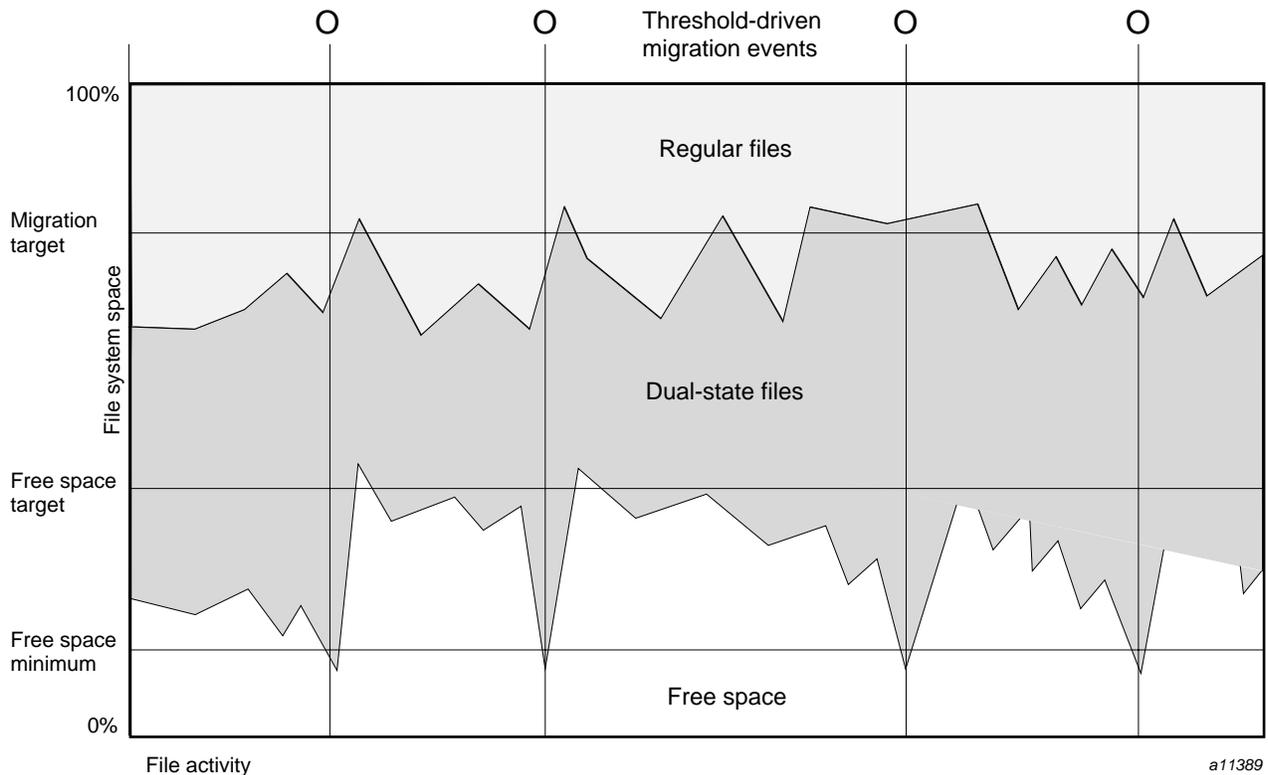
Figure 4. Relationship of Automated Space Management Targets

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 file system 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 file system 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.

3.3 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* (*SPOOL_DIR* is specified by the `SPOOL_DIR`
configuration parameter; see Section 2.5, page 26). The space-management
commands create the *daemon_name* subdirectory in *SPOOL_DIR* if it does not
already exist. The full path name 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:

- The name of the file system being processed

- The number of files selected for migration and freeing

- The number of disk blocks that were migrated and freed

- The names of any other DMF commands executed

- The command's success or failure in meeting the migration and free-space
  targets

The following excerpt show the format of an `autolog` file:

```
11:44:55-V 26968-dmfsmon /dmi - free_space=5.44, minimum=5
11:46:55-V 26968-dmfsmon /dmi - free_space=5.12, minimum=5
11:47:35-I 26968-dmfsmon Started 15135 for execution on /dmi
11:48:55-V 26968-dmfsmon /dmi - free_space=4.79, minimum=5
```

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

The DMF Daemon [4]

The DMF daemon, `dmdaemon`(8), is the core component of DMF. The daemon passes messages between commands, the MSPs, 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 `dmdaemon` command, as follows (assuming your `PATH` environment variable includes `/etc/dmf/dmbase/etc`):

`dmdaemon`

Typically, `dmdaemon` should be called as part of the normal system startup procedure by using a direct call in a system startup script in the `/etc/rc2.d` directory.

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

You can restart the daemon by using the `dmdaemon` command.

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

4.1 Daemon Processing

After initialization, `dmdaemon` performs the following steps:

1. Isolates itself as a daemon process.

2. Checks for the existence of other `dmdaemon` processes. If another `dmdaemon` exists, the newer one terminates immediately.

3. Initializes the `dmdaemon` log.

4. Opens the daemon database.

5. Initializes the daemon request socket.

6. Initiates the MSPs.

7. Enters its main request processing.

The daemon uses log files and journal files as described in Section 4.3.

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 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 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 is received or the MSP 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 `dmdaemon` 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, or MSP problem.

4.2 DMF Daemon Database and `dmdadm`

The DMF daemon maintains a database that resides in the directory *HOME_DIR*/*daemon_name* (*HOME_DIR* is specified by the HOME_DIR configuration parameter). This database contains information about the offline copies of a given file, as well as some information about the original file. The

database also contains the bit file identifier (bfid), which is assigned when the file is first migrated.

Other information maintained on a per-entry basis includes the following:

- File size (in bytes)

- MSP name and MSP recall path

- Date and time information, including the following:

  – Time at which the database record was created

  – Time at which the database record was last updated

  – A check time for use by the administrator

  – A soft-delete time, indicating when the entry was soft-deleted

- Original device and inode number

- Base portion of the original file name, if known

The dmdadm(8) command provides maintenance services for the daemon database.

dmdadm executes directives from stdin or from the command line when you use the -c option. All directives start with a directive name followed by one or more parameters. Parameters may be positional or keyword-value pairs, depending on the command. White space separates the directive name, keywords, and values.

When you are inside the dmdadm interface (that is, when you see the adm *command_number* > prompt), the command has a 30–minute timeout associated with it. If you do not enter a response within 30 minutes of the prompt having been displayed, the dmdadm session terminates with a descriptive message. This behavior on all the database administrative commands limits the amount of time that an administrator can lock the daemon and MSP databases from updates.

4.2.1 dmdadm Directives

The dmdadm directives are as follows:

Directive	Description
count	Displays the number of records that match the expression provided.

create	Deletes an existing database record.
dump	Prints the specified database records to standard out in ASCII; each database field is separated by the pipe character (|).
help	Displays help.
list	Shows the fields of selected database records. You may specify which fields are shown.
load	Applies records to the database obtained from running the dump directive.
quit	Stops program execution after flushing any changed database records to disk. The abbreviation q and the string exit produce the same effect.
set	Specifies the fields to be shown in subsequent list directives.
update	Modifies an existing database record.

The syntax for the dmdadm directives is summarized as follows:

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

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

- A bfid or range of bfids

- The keyword all

- A period (.), which recalls the previous selection

- An expression involving any of the above, field value comparisons, and, or, or parentheses.

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

The syntax for *selection* is as follows:

```
selection      ::=     or-expr
or-expr        ::=     and-expr [ or or-expr   ]
and-expr       ::=     nested-expr [ and and-expr   ]
nested-expr    ::=     comparison | ( expression )
comparison     ::=     bfid-range | field-keyword op   field-value
op             ::=     < | > | = | >= | <=
bfid-range     ::=     bfid [ – bfid] | [bfid – [bfid]] | key-macro
key-macro      ::=     all
field-keyword  ::=     name or abbreviation of the record field
field-value    ::=     appropriate value for the field
bfid           ::=     character representation of the bit file identifier
```

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

```
305c74b200000010-305c74b200000029
7fffffff000f4411-
-305c74b2000004c8
all
origsize>1m
. and origage<7d
mspkey 456 to origuid 2570
```

4.2.2 `dmdadm` Field and Format Keywords

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

Keyword	Description
checkage (ca)	The time at which the database record was last checked; the same as checktime, except that it is specified as *age*. Valid only in unsafe (-u) mode.
checktime (ct)	The time at which the database record was last checked; an integer that reflects raw UNIX time. Valid only in unsafe (-u) mode.
deleteage (da)	The time at which the database record was soft-deleted; the same as deletetime, except

	that it is specified as *age*. Valid only in unsafe (-u) mode.
deletetime (dt)	The time at which the database record was soft-deleted; an integer that reflects raw UNIX time. Valid only in unsafe (-u) mode.
mspname (mn)	The name of the MSP 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 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 *age*.
origdevice (od)	Original device number of the file; an integer.
originode (oi)	Original inode number of the file; an integer.
origname (on)	Base portion of the original file name; a string of up to 14 characters.
origsize (os)	Original size of the file; an integer.
origtime (ot)	Time at which the database record was created; an integer that reflects raw UNIX time.
origuid (ou)	Original user ID of the database record; an integer.
updateage (ua)	Time at which the database record was last updated; the same as updatetime, except that it is specified as *age*.
updatetime (ut)	Time at which the database record was last updated; an integer that reflects raw UNIX time.

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

The age field keywords (`checkage`, `deleteage`, `origage`, and `updateage`) let you express time as *age*. *age* is a string in a form such as `8w12d7h16m20s`, meaning 8 weeks, 12 days, 7 hours, 16 minutes, and 20 seconds old. The age keywords display their value as an integer followed by `w`, `d`, `h`, `m`, or `s` (weeks, days, hours, minutes, and seconds, respectively). The comparison > used with the age keywords means older than the value given (that is, `>5d` is older than 5 days).

The *limit* keywords restrict the records acted upon:

Keyword	Description
`recordlimit (rl)`	Limits the number of records acted upon to the value that you specify; an integer.
`recordorder (ro)`	Specifies the order that records are scanned; may be either `bfid` or `data`. `bfid` specifies that the records are scanned in bfid order. `data` specifies that the records are scanned in the order in which they are found in the database data file. `data` is more efficient for large databases, although it is essentially unordered.

The *format* keyword selects a format to use for the display. If, for example, you want to display fields in a different order than the default or want to include fields that are not included in the default display, you specify them with the `format` keyword. Values for `format` can be `default`, `keyword`, or a list of field keywords enclosed in quotation marks.

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

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

```
adm 3>list all recordlimit 20
          BFID      ORIG       ORIG   ORIG MSP       MSP
                    UID        SIZE    AGE NAME       KEY
-------------------------------------------------------------------
305c74b200000010    20934    69140480  537d silo1      88b49f
305c74b200000013    26444      279290  537d silo1      88b4a2
305c74b200000014    10634       67000  537d silo1      88b4a3
305c74b200000016    10634   284356608  537d silo1      88b4a5
305c74b200000018    10634     1986560  537d silo1      88b4a7
305c74b20000001b    26444      232681  537d silo1      88b4aa
305c74b20000001c    10015     7533688  537d silo1      88b4ab
305c74b200000022     8964    23194990  537d silo1      88b4b1
305c74b200000023     1294   133562368  537d silo1      88b4b2
305c74b200000024    10634       67000  537d silo1      88b4b3
305c74b200000025    10634   284356608  537d silo1      88b4b4
305c74b200000026    10634     1986560  537d silo1      88b4b5
305c74b200000027     1294     1114112  537d silo1      88b4b6
305c74b200000028    10634       25270  537d silo1      88b4b7
305c74b200000029     1294    65077248  537d silo1      88b4b8
305c74b20000002b     9244     2740120  537d silo1      88b4ba
305c74b200000064     9335        9272  537d silo1      88b4f3
305c74b200000065     9335       10154  537d silo1      88b4f4
305c74b200000066     9335        4624  537d silo1      88b4f5
305c74b200000067     9335       10155  537d silo1      88b4f6
adm 4>
```

The following example displays the number of records in the database that are associated with user ID 11789 and that were updated during the last five days:

```
adm 3>count origuid=11789 and updateage<5d
72 records found.
```

4.2.3 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}'
```

4.3 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 Section 2.6.1, page 31.

The DMF daemon automatically creates journal files that track database transactions. They have the following path name format (*JOURNAL_DIR* refers to the directory defined by the JOURNAL_DIR configuration parameter):

JOURNAL_DIR/*daemon_name*/dmd_db.*yyyymmdd[.hhmmss]*

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

- When the first transaction after midnight occurs

- When the journal file reaches size defined by the JOURNAL_SIZE configuration parameter

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

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

For example, the following shows the contents of a *JOURNAL_DIR*/*daemon_name* directory on 15 June 1998:

```
dmd_db.19980604.235959   dmd_db.19980612.235959
dmd_db.19980605.235959   dmd_db.19980613.145514
dmd_db.19980608.235959   dmd_db.19980613.214233
dmd_db.19980609.235959   dmd_db.19980613.235959
dmd_db.19980610.235959   dmd_db.19980614.235959
dmd_db.19980611.094745   dmd_db.19980615
dmd_db.19980611.101937
dmd_db.19980611.110429
dmd_db.19980611.235959
```

For every date on which database transactions occurred, there will exist a file with that date and the suffix .235959, with the exception of an existing open journal file. Some dates have additional files because the transaction log reached JOURNAL_SIZE at a specified time and the file was closed.

You can configure daemon_tasks parameters to remove old journal files (using the run_remove_journals.sh task and the JOURNAL_RETENTION parameter. For more information, see Section 2.6.1, page 31.

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 Section 7.6, page 133.

The DMF Lock Manager [5]

The dmlockmgr(8) process must be executing at all times for any DMF process to safely access and update a DMF database. The dmlockmgr and its clients (DMF processes such as dmatmsp, dmdaemon(8), dmvoladm(8), dmcatadm(8) and others) communicate through various methods. These methods include files, semaphores, and message queues. There are times when abnormal process terminations will result in non-orderly exit processing which 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: In this chapter, *SPOOL_DIR* refers to the value of the SPOOL_DIR parameter in the DMF configuration file.

The dmlockmgr files used by the database utilities are found in several different places. There are 3 types of files:

- dmlockmgr communication and log files

- Individual transaction log files

5.1 `dmlockmgr` Communication and Log Files

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

The token files in *SPOOL_DIR*/RDM_LM have the form shown in Table 3, page 90:

Table 3. dmlockmgr Token Files

File	Description
dmlockmgr	Used by the dmlockmgr and its clients to access dmlockmgr's semaphore and input message queue
dmatmsp*msp_name*	Used by the MSP *msp_name* and dmlockmgr to access the MSP's input message queue
dmdaemon*daemon*	Used by the DMF daemon and dmlockmgr to access the daemon's input message queue
dmatread*PID*	Used by the process whose process ID is *PID* to access the process's input message queue
dmatsnf*PID*	
dmcatadm*PID*	
dmdbrecover*PID*	
dmdbase*PID*	
dmvoladm*PID*	

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

The files of the *PID* versions listed in Table 3 are removed from the lockmgr directory automatically when the command terminates or when the DMF daemon initializes. Do not create files of this name format in this directory because the daemon is likely to remove them.

The IPC resources used by DMF are always released during normal process exit cleanup. If one of the dmlockmgr client processes dies without removing its message queue, dmlockmgr will remove that queue when it detects the death of the client. It will not remove the token file.

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

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

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

SPOOL_DIR/RDM_LM/dmlocklog.*yyyymmdd*

The log file is closed and a new one opened at the first log request of a new day. These files are not typically large files, but a new file will be created each day and you should periodically remove older versions. You should maintain the `dmlockmgr` log files for as long as you maintain the database transaction journal files.

5.2 `dmlockmgr` Individual Transaction Log Files

The individual transaction log files have the following form:

dmatmsp*msp_name*.log
dmdaemon*PID*.log
dmvoladm*PID*.log
dmcatadm*PID*.log
dmdbase*PID*.log
dmdbrecover*PID*.log
dmselect*PID*.log

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

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

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

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

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

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

⚠️ **Caution:** Do **not** delete the *SPOOL_DIR*/RDM_LM/`vista.taf` file.

Media Specific Processes (MSPs) [6]

A file is migrated from online disk to offline media by a media-specific process (MSP). There are three types of MSPs:

- Tape MSP, which copies files from a disk to tape, or copies files from tape to disk

- File transfer protocol (FTP) MSP, which allows the DMF daemon to manage data by moving it to a remote machine

- Disk MSP, which migrates data to a directory that is accessed on the current system

The following sections provide a general description of MSP operation.

6.1 The Tape MSP

The tape MSP consists of three programs: `dmatmsp`, `dmatwc`, and `dmatrc`.

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

The `dmatmsp` maintains two types of records in its database:

- CAT records, which contain information about the files the MSP maintains

- VOL records, which contain information about the media the MSP 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 Section 6.1.3, page 96, and Section 6.1.4, page 97.

The tape MSP provides a mechanism for copying active data from volumes that contain largely obsolete data to volumes that contain mostly active data. This process is referred to as *volume merging* or *compression*. Data on MSP volumes becomes obsolete when users delete or modify their files. Volume merging can be configured to occur automatically (see Section 2.9.1, page 50). It can also be triggered by marking MSP volumes as sparse with the `dmvoladm`(8) command.

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

- dmatread(8), which copies all or part of a migrated file to disk

- dmatsnf(8), which audits and verifies MSP volumes

6.1.1 Tape MSP Directories

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

- Databases

- Database journal files

- Log files

Sites define the location of these directories by editing the base object configuration file parameters HOME_DIR, JOURNAL_DIR, and SPOOL_DIR, whose values are referred to as *HOME_DIR*, *JOURNAL_DIR*, and *SPOOL_DIR* in this document. A given instance of the tape MSP creates a subdirectory named after itself in each of these three directories.

For example, if an instance of the tape MSP is called cart1, its database files reside in directory *HOME_DIR*/cart1. If another instance of the tape MSP is called cart2, its database files reside in *HOME_DIR*/cart2.

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

6.1.2 Media Concepts

The tape MSP takes full advantage of the capabilities of modern tape devices, including data compression and fast media positioning. To accommodate these capabilities and to provide recovery from surface or other media defects, dmatmsp 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 5, page 96). Thus, every migrated file has at least one, and

sometimes many, chunks. Such a concept is necessary because the capacity of a volume is unknown until written, both because of natural variation in the medium itself and because the effect of data compression varies with the data contents.

- A *zone* is a logical block containing several physical blocks ending with a tape mark. A zone has a target size that is configurable by media type. The default zone target size is 50 MB.

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

  – The zone size is exceeded

  – The MSP exhausts chunks to write

  – The end of tape is encountered

  Thus, the actual zone size can vary from well below the target size to the entire tape volume. A zone never spans physical volumes.

  The zone plays several roles:

  – The zone size is the amount of data that triggers dmatmsp to start a process to write files to tape.

  – The MSP 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 5 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.
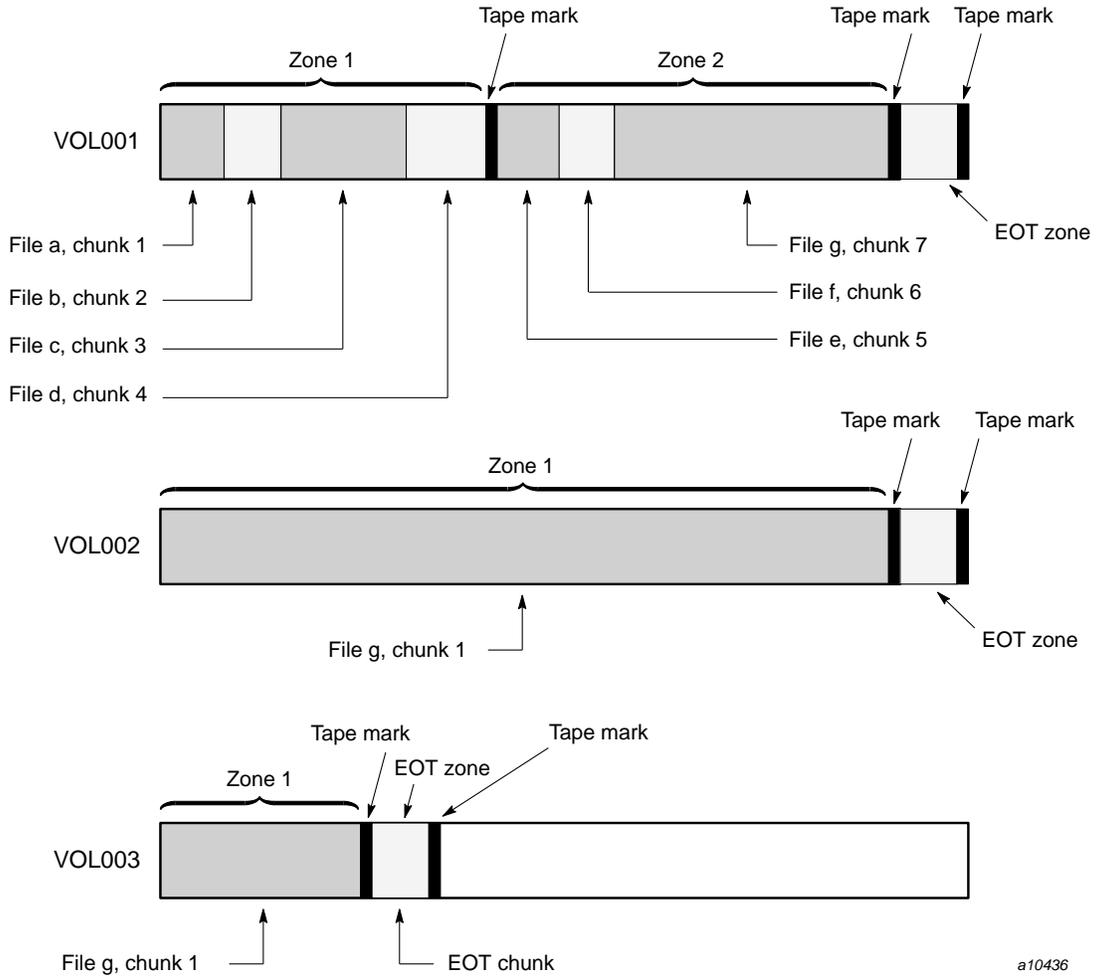
Figure 5. Media Concepts

6.1.3 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 with files migrate.

The CAT portion of the MSP database consists of three 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 `atmsp_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 Section 6.1.8, page 103).

> **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.

6.1.4 VOL Database Records

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

The VOL portion of the MSP database consists of two 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 `atmsp_db.dbd`.

Both files contain binary data and require special maintenance utilities. The `dmvoladm` command, described in more detail in Section 6.1.9, page 110, provides facilities to create, query, and modify VOL records in the database. Additional database maintenance utilities are described in Section 7.6, page 133.

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

6.1.5 Tape MSP Journals

Each instance of `dmatmsp` protects its database by recording every transaction in a journal file. The journal file path names have the following format:

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

The MSP creates journal files automatically.

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

- When the first transaction after midnight occurs

- When the journal file reaches the size defined by the `JOURNAL_SIZE` configuration parameter

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

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

For example, the following shows the contents of a *JOURNAL_DIR*/*msp_name* directory on 15 June 1998:

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

For every date on which database transactions occurred, there will exist a file with that date and the suffix `.235959`, with the exception of an existing open journal file. Some dates may have additional files because the transaction log reached `JOURNAL_SIZE` at a specified time and the file was closed.

You can configure daemon_tasks parameters to remove old journal files (using the run_remove_journals.sh task and the JOURNAL_RETENTION parameter. For more information, see Section 2.6.1, page 31.

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

6.1.6 Tape MSP Logs

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.

These log files are distinct from the logs maintained by the DMF daemon; however, some of the messages that occur in the daemon log are responses that the tape MSP 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_config(5) man page.

The msplog.*yyyymmdd* file is the primary log for the tape MSP and contains most of the messages. This file is written by dmatmsp, dmatrc, and dmatwc. A new msplog.*yyyymmdd* is created for each day.

This section describes informational statistics provided by the tape log files. These messages appear in the *SPOOL_DIR*/*msp_name*/msplog.*yymmdd* files. Timing information provided (such as MB transferred per second) should not be used as an accurate benchmark of actual data transfer rates. This information is provided for monitoring DMF and should only be used in comparison to similar data provided by DMF. Text in all uppercase references a parameter defined in the DMF configuration file. You can reference the comments in the sample configuration file or in the dmf_config(5) man page for a more detailed definition of these parameters.

Note: Because the MSP will continue to create log files and journal files without limit, you must remove obsolete files periodically by configuring the run_remove_logs.sh and run_remove_journals.sh tasks in the configuration file, as described in Section 2.6.1, page 31.

Example 1: Tape MSP Statistics Messages

The following is an example of advanced tape MSP statistics messages taken from an `msplog.`*yyyymmdd* file. These messages are automatically issued by the MSP every 5 minutes.

```
02:02:00-I 27082-dmatmsp stats: children=5/5/8, btp=2818446/100000000/0, wc=1/8, cwc=47952
02:02:00-I 27082-dmatmsp stats: data put=48480.521 mb, data recalled= 93.446 mb
02:02:00-I 27082-dmatmsp stats:      dm_put -     0  65179      0     77
02:02:00-I 27082-dmatmsp stats:      dm_get -     2  23706    743      5
02:02:00-I 27082-dmatmsp stats:   dm_delete -     0  27459      0      0
02:02:00-I 27082-dmatmsp stats:   dm_cancel -     0     82     12      0
02:02:00-I 27082-dmatmsp stats: dm_flushall -     0   1820      0      0
02:02:00-I 27082-dmatmsp stats: dm_snapshot -     0    120      0      0
02:02:00-I 27082-dmatmsp stats:     dm_merge -    33   5967      0      0
02:02:00-I 27082-dmatmsp stats: mc=8, ms=1000000000, mu=74854430, sm=0
```

The information provided by these entries is defined as follows:

- The first line contains the following information:

  - `children=5/5/8` represents the total child processes (5), the active child processes (5), and the configured value of `CHILD_MAXIMUM` (8)

  - `btp=2818446/100000000/0` represents the bytes queued for putting (2818446), the threshold at which to start the next put child (100,000,000), and the bytes assigned to socket I/O (0)

  - `wc=1/8` represents the active write child processes (1) and the configured value of `MAX_PUT_CHILDREN` (8)

  - `cwc=47952` represents the process ID of the current write child (that is, the write child which is accepting data to write)

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

  request_name   *active*   *successful*   *errors*   *canceled*

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

- The last line provides the following information:

- mc is the configured value for MERGE_CUTOFF, the cutoff to stop scheduling tapes for merging (8)

- ms is the configured value for CACHE_SPACE, the merge cache space available (1 Gbyte)

- mu is the merge cache space used (74854430)

- sm is the number of socket merge children (0)

The tape MSP write child (dmatwc) and read child (dmatrc) also produce statistics messages in the MSP log file. These messages contain timing statistics whose format changes from release to release, and they are not documented in this manual.

6.1.7 Volume Merging

When users delete or modify their migrated files, the copy on tape becomes obsolete. Over time, some volumes will become entirely empty and can be reused. However, most volumes experience a gradual increase in the ratio of obsolete data to active data; such volumes are said to be *sparsely populated* or simply *sparse*. To reclaim the unused space on these volumes, DMF provides a *volume merge* facility, which copies the active data from several sparse volumes to a new volume, thus freeing the sparse volumes for reuse. Volume merging can be configured to occur automatically by using the run_merge_tapes.sh task (see Section 2.9.1, page 50).

Volume merging can also be done manually. dmatmsp performs merge operations whenever sparse volumes and the necessary resources exist at the same time. Use the dmvoladm select directive to mark MSP volumes as sparse. (The select directive is described in Section 6.1.9, page 110.) Because the merge processing occurs simultaneously with other DMF activities, it is easiest to configure DMF to automatically perform merges at night or during other periods of relatively low activity.

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

In busy environments that have heavy contention for tape drives, the close coupling between the socket's tape reader and tape writer can be costly, especially when short files are being transferred. For large files, the overhead

and possible delays in waiting for both tapes to be mounted is small compared to the benefit of rapid transfer and zero impact on free disk space. For this reason, you can move small files through a disk cache and big files through a socket. This process is mediated by the following configuration parameters:

Parameter	Description
CACHE_SPACE	Specifies the amount of disk space that will be used to temporarily store chunks during a merge operation.
CACHE_DIR	Specifies the directory into which the MSP will store chunks while merging them from sparse tapes. If CACHE_DIR is not specified, TMP_DIR is used.
MAX_CACHE_FILE	Specifies the largest chunk that will be stored temporarily on disk during a merge operation.
MERGE_CUTOFF	Specifies the number of child processes after which the MSP will stop scheduling tapes for merging. This number is the sum of the active and queued children generated from gets, puts, and merges.

Using a small amount of disk space to hold small chunks can have a significant impact on the total time required to perform merges. The default configuration options are set to move 100% of merge data across sockets.

Note: It is important to avoid volume merging on more than one MSP simultaneously if the two MSPs share a tape device. If you initiate a merge process on more than one MSP on the same device at the same time (either by entering the same time in the `dmf_config` configuration file or by triggering the process manually), both processes will compete for tape transports. When a limited number of tape transports are available, a deadlock can occur. If you chose not to configure DMF to perform merges automatically by configuring the `run_tape_merge.sh` task, ensure that your `cron` jobs that automatically initiate volume merging refrain from initiating a second merge process until after all previously initiated merges are complete. You can accomplish this by using the `dmvoladm` command within the `cron` job to check for tapes that have the `hsparse` flag:

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

6.1.8 `dmcatadm` Command

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

When you are inside the `dmcatadm` interface (that is, when you see the adm *command_number* > prompt), the command has a 30–minute timeout associated with it. If you do not enter a response within 30 minutes of the prompt having been displayed, the `dmcatadm` session terminates with a descriptive message. This behavior on all the database administrative commands limits the amount of time that an administrator can lock the daemon and MSP databases from updates.

Note: Most of these facilities, especially the ability to create and modify CAT database records, are intended primarily for testing purposes.

6.1.8.1 `dmcatadm` Directives

The `dmcatadm` command executes directives from `stdin` or from the command line when you use the `-c` option. All directives start with a directive name followed by one or more parameters. Parameters may be positional or keyword-value pairs, depending on the command. White space separates the directive name, keywords, and values.

The dmcatadm directives are as follows:

Directive	Description
count	Displays the number of records that match the expression provided.
create	Creates a CAT record.
delete	Deletes an existing CAT record.
dump	Prints the specified CAT records to standard out in ASCII; each database field is separated by the pipe character (|).
help	Displays help.
list	Shows the fields of selected CAT records. You may specify which fields are shown.
load	Applies records to the MSP database obtained from running the dump directive.
quit	Stops program execution after flushing any changed database records to disk. The abbreviation q and the string exit produce the same effect.
set	Specifies the fields to be displayed in subsequent list directives.
update	Modifies an existing CAT record.
verify	Verifies the MSP database against the dmdaemon database.

The first parameter of most directives specifies the database records to manipulate, and the remaining parameters are keyword-value pairs.

The syntax for the dmcatadm directives is summarized as follows:

```
count selection [limit]
create key field...
delete selection [limit]
dump selection [limit]
help
list selection [limit] [format]
load filename
quit (or q or exit)
set [format]
update selection [limit] to fields...
verify selection [entries] [mspname] [limit]
```

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

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

- A *key* or range of *keys* in the form *key* [–] [*key*]. *key*– specifies all records starting with *key*, and –*key* specifies all records up to *key*.

- The keyword all

- A period (.), which recalls the previous selection

- An expression involving any of the above, field value comparisons, and, or, or parentheses.

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

The syntax for *selection* is as follows:

selection	::=	*or-expr*
or-expr	::=	*and-expr* [or *or-expr*]
and-expr	::=	*nested-expr* [and *and-expr*]
nested-expr	::=	*comparison* | (*expression*)
comparison	::=	*key-range* | *field-keyword* op *field-value*
op	::=	< | > | = | >= | <=
key-range	::=	*key* [– *key*] | [*key* – [*key*]] | *key-macro*
key-macro	::=	all
field-keyword	::=	*name or abbreviation of the record field*
field-value	::=	*appropriate value for the field*
key	::=	*character representation of the record key*

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

```
1510-1514
10000000000-
-15138
all
chunkoffset>0
chunknumber>0 and writeage<5d
. recordorder data
vsn=S07638
```

6.1.8.2 `dmcatadm` Field Keywords

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

Keyword	Description
chunkdata (cd)	Specifies the actual number of bytes written to tape by the MSP for the chunk. In the case of sparse files, this field will be smaller than `chunklength`. This is valid only in unsafe (`-u`) mode.
chunklength (cl)	The size of the chunk in bytes; an integer. This is valid only in unsafe (`-u`) mode.
chunknumber (cn)	The ordinal of the chunk on its volume. For example, 1 if the chunk is the first chunk on the volume, 2 if it is the second, and so on. Valid only as part of *selection*.
chunkoffset (co)	The byte offset within the file where the chunk begins; an integer. For example, the first chunk of a file has `chunkoffset` 0. If that first chunk is 1,000,000 bytes long, the second chunk would have `chunkoffset` 1000000. This is valid only in unsafe (`-u`) mode.
filesize (fs)	The original file size in bytes, an integer. This is valid only in unsafe (`-u`) mode.
flags (fl)	Not yet used by DMF.

readage (ra)	The date and time when the chunk was last read; the same as readdate, except specified as *age*.
readcount (rc)	The number of times the chunk has been recalled to disk; an integer.
readdate (rd)	The date and time when the chunk was last read, an integer that reflects raw UNIX time.
vsn (v)	The volume serial number(s); a list of one or more 6-character alphanumeric volume serial numbers separated by colons (:).
writeage (wa)	The date and time when the chunk was written; the same as writedate, except specified as *age*. This is valid only in unsafe (-u) mode.
writedate(wd)	The date and time when the chunk was written, an integer that reflects raw UNIX time. This is valid only in unsafe (-u) mode.
zoneblockid (zb)	Allows just the block ID portion of the zonepos to be displayed, returned, or changed. This is valid only in unsafe (-u) mode.
zonenumber (zn)	Allows just the zone number portion of the zonepos to be displayed, returned, or changed. This is valid only in unsafe (-u) mode.
zonepos (zp)	The physical address of the zone on the volume, expressed in the form *integer/hexinteger*, designating a zone number and block ID. A value of zero is used for *hexinteger* if no block ID is known. *integer* is the same as zonenumber, and *hexinteger* is the same as zoneblockid. This is valid only in unsafe (-u) mode.

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

The age field keywords (readage and writeage) let you express time as *age*, a
string in a form such as 8w12d7h16m20s (meaning 8 weeks, 12 days, 7 hours,
16 minutes, and 20 seconds old). The age keywords display their value as an

integer followed by w, d, h, m, or s (weeks, days, hours, minutes, and seconds, respectively). The comparison > used with the age keywords means older than the value given (that is, >5d is older than 5 days).

The *limit* keywords limit the records acted upon:

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

The following keywords specify files of daemon database entries:

Keyword	Description
entries (e)	Specifies a file of daemon database entries; a string.
mspname (mn)	Specifies the name of the MSP associated with the record; a string.

The *format* keyword selects a format to use for the display. If, for example, you want to display fields in a different order than the default or want to include fields that are not included in the default display, you specify them with the format keyword. Values for format can be default, keyword, or a list of field keywords enclosed in quotation marks.

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

For information about the role of the dmcatadm(8) command in database recovery, see Section 7.6, page 133.

Example 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
                       FILE   WRITE      CHUNK       CHUNK CHUNK
                KEY    SIZE   AGE       OFFSET      LENGTH NUM VSN
-----------------------------------------------------------------------------
       3273d5420001e242    77863935    61d            0    77863935   13 S12940
       3273d5420001e244    48365568    61d            0    24821760  168 S12936
       3273d5420001e244    48365568    61d     24821760    23543808    1 S12945
       3273d5420001e245    51019776    61d            0    51019776    2 S12945
       3273d5420001e246    45629440    61d            0    45629440   59 S12938
       3273d5420001e247    35586048    61d            0    35586048   60 S12938
       3273d5420001e248     9568256    61d            0     9568256    3 S12944
       3273d5420001e249    14221312    61d            0    14221312    4 S12944
       3273d5420001e24a      458752    61d            0      458752    5 S12944
       3273d5420001e24b    14155776    61d            0    14155776    6 S12944
```

6.1.8.3 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`

6.1.9 `dmvoladm` Command

The `dmvoladm`(8) command provides maintenance services for VOL records in the MSP database. In addition to the creation and modification of volume records, `dmvoladm` has an important role in the recovery of VOL records from a database checkpoint and is the mechanism that triggers volume merge activity.

When you are inside the `dmvoladm` interface (that is, when you see the `adm` *command_number* > prompt), the command has a 30–minute timeout associated with it. If you do not enter a response within 30 minutes of the prompt having been displayed, the `dmvoladm` session terminates with a descriptive message. This behavior on all the database administrative commands limits the amount of time that an administrator can lock the daemon and MSP databases from updates.

6.1.9.1 `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. `dmvoladm` directives follow:

Directive	Description
`count`	Displays the number of records that match the expression provided.
`create`	Creates a VOL record.
`delete`	Deletes an existing VOL record.
`dump`	Prints the specified VOL records to standard output in ASCII. Each database field is separated by the pipe character (`|`).
`help`	Displays help.
`list`	Shows the fields of selected VOL records. You may specify which fields are shown.
`load`	Applies VOL records to the database obtained from running the `dump` directive.

quit	Stops program execution after flushing any changed database records to disk. The abbreviation q and the string exit produce the same effect.
repair	Causes dmvoladm to adjust the usage information for specified volumes based on CAT data in the database. This directive is valid only in unsafe (-u) mode.
select	Marks selected volumes as being sparse. Equivalent to update *expression* to hsparse on.
set	Specifies the fields to be shown in subsequent list directives.
update	Modifies an existing VOL record.
verify	Verifies the MSP databases against the dmdaemon databases.

The syntax for the dmvoladm directives is summarized as follows:

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

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

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

- A *vsnlist* or range of VSNs in the form *vsn*[-*vsn*]. *vsn*- specifies all records starting with *vsn*, and -*vsn* specifies all records up to *vsn*.

- A period (.), which recalls the previous selection

- The name of one of the flags in the keyword list that follows in this section.

- One of the words `all`, `used`, `empty`, or `partial` or any of the `hflags`, whose meanings are as follows:

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

The syntax for *selection* is as follows:

selection	::=	*or-expr*
or-expr	::=	*and-expr* [or *or-expr*]
and-expr	::=	*nested-expr* [and *and-expr*]
nested-expr	::=	*comparison* | (*expression*)
comparison	::=	*vsnlist* | *field-keyword* *op* *field-value*
op	::=	< | > | = | >= | <=
key-range	::=	*vsn* [– *vsn*] | [*vsn* – [*vsn*]] | *key-macro*
key-macro	::=	all | empty | used | partial | *flag(s)*
field-keyword	::=	*name or abbreviation of the record field*
field-value	::=	*appropriate value for the field*
vsnlist	::=	*character representation of the volume serial number*

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

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

6.1.9.2 `dmvoladm` Field and Format Keywords

The *field* keywords specify new values for fields:

Keyword	Description
blocksize (bs)	Specifies the data block size in bytes when the tape was first written; an integer. The default is 65,536. This keyword is used only when mounting tapes with existing valid data. When an empty tape is first written, the MSP uses the default value for the tape type, unless it is overridden by a value in the BLOCK_SIZE parameter for the tape device in the DMF configuration file. This is valid only in unsafe (-u) mode.
chunksleft (cl)	Specifies the number of active chunks on the volume; an integer. This is valid only in unsafe (-u) mode.
dataleft (dl)	Specifies the number of bytes of active data on the volume. You specify this number as an integer, but for readability purposes it is displayed in megabytes (MB). This is valid only in unsafe (-u) mode.
datawritten (dw)	Specifies the maximum number of bytes ever written to the volume. You specify this number as an integer, but for readability purposes it is displayed in MB. This is valid only in unsafe (-u) mode.
eotblockid (eb)	Specifies the blockid of the chunk containing the end-of-tape marker; a hexinteger. This is valid only in unsafe (-u) mode.
eotchunk (ec)	Specifies the number of the chunk containing the end-of-tape marker; an integer. This is valid only in unsafe (-u) mode.
eotpos (ep)	Specifies the absolute position of the end-of-tape marker zone in the form *integer*/*hexinteger*, designating a zone number and block ID. A value of zero is used for *hexinteger* if no block ID is known. *integer* the same as eotzone, and *hexinteger* is the same as eotblockid. This is valid only in unsafe (-u) mode.

eotzone (ez)	Specifies the number of the zone containing the end-of-tape marker; an integer. This is valid only in unsafe (-u) mode.
label (lb)	Specifies the label type: al for ANSI standard labels; sl for IBM standard labels; or nl for nonlabeled volumes. The default is al.
tapesize (ts)	Specifies the estimated capacity in bytes; an integer. The default is 215 MB.
threshold (th)	Specifies the ratio of dataleft to datawritten as a percentage. This field cannot be displayed or updated.
upage (ua)	(Display only.) Specifies the date and time of the last update to the volume's database record. The same as for update, except that it is expressed as *age*. This is valid only in unsafe (-u) mode.
update (ud)	(Display only.) Specifies the date and time of the last update to the volume's database record, expressed as an integer that reflects raw UNIX time. This is valid only in unsafe (-u) mode.
version (v)	Specifies the DMF tape format version, an integer. This is valid only in unsafe (-u) mode.
wfage (wa)	Specifies the date and time that the volume was written to or freed for reuse. The same as for wfdate, except that it is expressed as *age*. This is valid only in unsafe (-u) mode.
wfdate (wd)	Specifies the date and time that the volume was written to or freed for reuse, expressed as an integer that reflects raw UNIX time. This is valid only in unsafe (-u) mode.

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

The age field keywords (upage and wfage) let you express time as *age*, a string in a form such as 8w12d7h16m20s (meaning 8 weeks, 12 days, 7 hours, 16 minutes, and 20 seconds old). The age keywords display their value as an

integer followed by w, d, h, m, or s (weeks, days, hours, minutes, and seconds, respectively). The comparison > used with the age keywords means older than the value given (that is, >5d is older than 5 days).

The *limit* keywords restrict the volumes acted upon:

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

The *format* keyword selects a format to use for the display. If, for example, you want to display fields in a different order than the default or want to include fields that are not included in the default display, you specify them with the format keyword. Values for format can be default, keyword, or a list of field keywords enclosed in quotation marks.

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

Keyword	Description
hbadmnt (hb)	Indicates that the volume could not be mounted. If the problem causing the mount to fail is transient, the MSP will clear the flag the next time it attempts to mount the tape and succeeds. Typically this flag indicates a permanent condition that should be investigated and corrected. It is displayed as -------b.
herr (he)	Indicates that an I/O error has occurred on the volume; displayed as e--------.

hflags (no abbreviation)	(Display only.) Shows the complete set of hold flags as a 9–character string. Each flag has a specific position and alphabetic value. If the flag is off, a dash (-) is displayed in its position; if the flag is on, the alphabetic character is displayed in that position.
hfree (no abbreviation)	Indicates that the volume has no active data and is available for reuse after HFREE_TIME has expired, displayed as -f-------. See the dmf_config(5) man page for information about the HFREE_TIME configuration parameter. This is valid only in unsafe (-u) mode.
hfull (hu)	Indicates that the volume cannot hold any more data; displayed as ------u--.
hlock (hl)	Currently unused, displayed as ----l----.
hoa (ho)	Indicates that the volume is not to be used for either input or output, displayed as --o------.
hro (hr)	Indicates that the volume is read-only, displayed as ---r-----; this inhibits the MSP from using the volume for output.
hrsv (h*)	Currently unused (reserved); displayed as ----*----. This is valid only in unsafe (-u) mode.
hsparse (hs)	Indicates that the volume is considered sparse and thus a candidate for a volume merge operation, displayed as -------s-.

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

For information about the role of the dmvoladm command in database recovery, see Section 7.6, page 133. For details about dmvoladm syntax, see the man page.

Example 3: `dmvoladm list` directives

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

"on". The positions spell the string `efor*lmusb`, representing `herr`, `hfree`, `hoa`, `hro`, `hrsv`, `hlock`, `hfull`, `hsparse`, and `hbadmnt`, respectively.

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

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

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

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

```
adm 21>list S03232-S03254 format "eotchunk eotzone eotpos"
          EOT   EOT
VSN      CHUNK  ZONE    EOTPOS
------------------------------
S03232     10     2 2/4294967295
S03233      2     2 2/4294967295
S03234    598     2 2/4294967295
S03235     18     2 2/4294967295
S03236     38     2 2/4294967295
S03237     92     2 2/4294967295
S03238      1     1 1/4294967295
S03239      1     1 1/4294967295
S03240      1     1 1/4294967295
S03241    325     2 2/4294967295
S03242     81     2 2/4294967295
S03243     26     2 2/4294967295
S03244      1     1 1/4294967295
S03245     26     2 2/4294967295
S03246      5     2 2/4294967295
S03247    186     2 2/4294967295
S03248     17     2 2/4294967295
S03249    526     2 2/4294967295
S03250      1     1 1/4294967295
S03251    533     2 2/4294967295
S03252    157    17 17/2147483648
S03253    636     2 2/4294967295
S03254     38     2 2/4294967295
```

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

```
adm 23>list 003232-003254 format "herr hfree hfull hlock hoa hro"
       herr hfree hfull hlock hoa hro
VSN
-------------------------------------
003232  off    off    on    off off off
003233  off    off   off    off off off
003234  off    off   off    off off off
003235  off    off   off    off off off
003236  off    off    on    off off off
003237  off    off    on    off off off
003238  off    off    on    off off off
003239  off    off    on    off off off
003240  off    off   off    off off off
003241  off    off    on    off off off
003242  off    off    on    off off off
003243  off    off   off    off off off
003244  off    off   off    off off off
003245  off    off    on    off off off
003246  off    off   off    off off off
003247  off    off    on    off off off
003248  off    off    on    off off  on
003249  off     on   off    off off  on
003250  off     on   off    off off  on
003251  off     on   off    off off  on
003252  off     on   off    off off  on
003253  off    off    on    off off  on
003254  off    off    on    off off  on
```

6.1.9.3 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. lbtype

4. `capacity`

5. `blocksize`

6. `hflags` (in octal)

7. `version`

8. `datawritten`

9. `eotchunk`

10. `eotposition` (`eotzone/eotblockid`) (in hexadecimal)

11. `dataleft`

12. `chunksleft`

13. `wfdate`

14. `update`

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

6.1.10 `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 4: Restoring Hard-deleted Files Using `dmatread`

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

1. Determine the bfid of the file you want to restore. You can use backup copies of `dmdlog` or your `dbrec.dat` files, or a restored dump copy of the deleted file's inode (and the `dmattr` command).

2. Using backup copies of the MSP databases, use a `dmatread`(8) command similar to the following:

   ```
   dmatread -p /a/dmbackup -B 342984C50000000000084155
   ```

342984C50000000000084155 is the bfid of the file to be restored, and
/a/dmbackup is the directory containing the backup copies of the MSP
databases. Your file will be restored to the current directory as
B342984C50000000000084155

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

If you have access to chunk and VSN information for the file to be restored, you
can use the dmatread -c and -v options and avoid using backup copies of the
MSP 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.

6.1.11 `dmatsnf` Command

Use the dmatsnf(8) command to verify the readability of or to audit the
contents of MSP volumes. You may also generate text database records that can
be applied to the MSP databases (using the load directive in dmcatadm and
dmvoladm, respectively), in order to add the contents of a volume to the MSP
database (although this is impractical for large numbers of volumes).

dmatsnf can be used to verify one or more tape volumes against the MSP
databases. It also can be used to generate journal entries, which can be added
to the MSP databases by using the load directive in dmvoladm and dmcatadm.

6.1.12 `dmaudit verifymsp` Command

Use the verifymsp option of the dmaudit(8) command to check the
consistency of the DMF daemon and MSP databases after an MSP, DMF, or
system failure. This command captures the database files and compares the
contents of the daemon database with each MSP 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.

6.2 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.

6.2.1 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.

6.2.2 Activity Log

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

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

Note: Because the MSP will continue to create log files and journal files without limit, you must remove obsolete files periodically by configuring the run_remove_logs and run_remove_journals tasks in the configuration file, as described in Section 2.6.1, page 31.

6.2.3 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 file system 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 file system to be restored to a
previous state.

6.3 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. 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 meta-data 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.

6.3.1 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.

6.3.2 Activity Log

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

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

Note: Because the MSP will continue to create log files and journal files without limit, you must remove obsolete files periodically by configuring the run_remove_logs and run_remove_journals tasks in the configuration file, as described in Section 2.6.1, page 31.

6.4 Moving Migrated Data between MSPs

DMF provides a mechanism to move copies of offline or dual-state files from one MSP to another. The dmmove(8) command takes a list of such files and moves them to a specified set of MSPs. The list of MSPs specified to the dmmove command indicates which MSPs are to contain migrated copies of a file after the move process is completed. All other migrated copies are hard-deleted.

Note: All migrated copies of files are hard-deleted, including those on MSPs that are not indicated on the dmmove command.

If a file's migrated state is offline, dmmove recalls the file to disk and then remigrates it to the specified MSPs. When the migration process is complete, the online copy is removed. The file is recalled to a scratch file system 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 path names as input.

See the man pages for `dmselect` and `dmmove` for all the possible options and further information.

DMF Maintenance and Recovery [7]

This chapter contains information for the administrative maintenance of DMF.

7.1 Retaining Old DMF Daemon Log Files

The daemon generates the *SPOOL_DIR*/*daemon_name*/`dmdlog.`*yyyymmdd* log file, which contains a record of DMF activity and can be useful for problem solving for several months after creation. All MSPs generate a *SPOOL_DIR*/*msp_name*/`msplog.`*yyyymmdd* log file, which also contains sometimes useful information about its activity. These log files should be retained for a period of some months. Log files more than a year old are probably not very useful.

Do not use DMF to manage the *SPOOL_DIR* file system.

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 Section 2.6.1, page 31, for more information.

7.2 Retaining Old DMF Daemon Journal Files

Both the daemon and tape MSP generate journal files that are needed to recover databases in the event of file system 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 Section 2.6.1, page 31, for more information.

7.3 Soft- and Hard-deletes

When a file is first migrated, a *bit-file identifier*, or *bfid*, (the key into the daemon database) is placed in the inode. 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 file system is damaged, the files will be irretrievable after you restore the file system.

For example, assume that migrated files were located in the /x file system, 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 file system to as recent a state as possible. If you restore the file system 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 file system, 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 file system 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. 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 Section 2.6.1, page 31, 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.

7.4 Using **xfsdump** and **xfsrestore** with Migrated Files

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

The xfsdump(1M) and xfsrestore(1M) commands back up file systems. These utilities are designed to perform the backup function quickly and with minimal system overhead. They operate with DMF in two ways:

- When xfsdump encounters an offline file, it does not cause the associated data to be recalled. This distinguishes the utility from tar(1) and cpio(1), both of which cause the file to be recalled when they reference an offline file.

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

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

Most installations periodically do a full (level 0) dump of file systems. Incremental dumps (levels 1 through 9) are done between full dumps; these may happen once per day or several times per day. You can continue this practice after DMF is enabled. When a file is migrated (or recalled), the inode change time is updated. The inode change time ensures that the file gets dumped at the time of the next incremental dump.

You can configure tasks in the dump_tasks object to automatically do full and incremental dumps of the DMF-managed file systems. See Section 2.6.1, page 31, for more information.

The dump_tasks object employs scripts that call the xfsdump(1M) command in conjunction with the dmtape DMF support program. This mechanism gives

you flexible and efficient use of a predetermined set of backup volumes that are automatically allocated to the `xfsdump` program as needed during the backup. In order to allow you an equally flexible and efficient method for restoring files backed up by the `dump_tasks` object, the `dmxfsrestore`(8) command should be used any time a restore is required for a `dump_tasks`-managed file system. Please see the `dmxfsrestore`(8) man page for more information on running the command.

7.4.1 Dumping and Restoring Files without the `dump_tasks` Object

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

- The `dump_tasks` object uses `xfsdump` with the `-a` option to dump only data not backed up by DMF. You may also wish to consider using the `-a` option on `xfsdump` when dumping DMF file systems manually.

- Do **not** use the `-A` option on either `xfsdump` or `xfsrestore`. The `-A` option avoids dumping or restoring extended attribute information. DMF information is stored within files as extended attributes, so if you do use `-A`, migrated files restored from those dump tapes will not be recallable by DMF.

- When restoring migrated files using `xfsrestore`, you must specify the `-D` option in order to guarantee that restored files will be recallable by DMF.

- If you use the Tape Management Facility (TMF) to mount tapes for use by `xfsdump`, be aware that `xfsdump` will not detect the fact that the device is a tape, and will behave as if the dump is instead being written to a regular disk file. This means that `xfsdump` will not be able to append new dumps to the end of an existing tape. It also means that if `xfsdump` encounters end-of-tape, it will abort the backup rather than prompting for additional volumes. You must ensure that you specify enough volumes using the `tmmnt -v` option before beginning the dump in order to guarantee that `xfsdump` will not encounter end-of-tape.

7.4.2 File System 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 Section 7.3, page 130.) `dmaudit`(8) will report this as an inconsistency between the file system and the database, indicating that the database entry should not be soft-deleted.

Another form of inconsistency occurs if you happen to duplicate offline or dual-state files by restoring all or part of an existing directory into another directory. In this case, dmaudit will report as an inconsistency that two files share the same bfid. If one of the files is subsequently deleted causing the database entry to be soft-deleted, the dmaudit-reported inconsistency will change to the type described in the previous paragraph.

While these dmaudit-reported inconsistencies may seem serious, there is no risk of any user data loss. The dmhdelete(8) program responsible for removing unused database entries always first scans all DMF-managed file systems to make sure that there are no remaining files which reference the database entries it is about to remove. It is able to detect either of these inconsistencies and will not remove the database entries in that case.

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

7.5 Using `dmfill`

The dmfill(8) command allows you to fill a restored file system 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 file system is restored, only the inodes are restored, not the data. You can use dmfill in conjunction with xfsrestore to restore a corrupted file system 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 file system.

7.6 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.

7.6.1 Database Backups

You configure tasks in the `run_copy_databases.sh` task in the `dump_tasks` object to automatically generate DMF database backups. See Section 2.6.1, page 31, for more information.

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

- *HOME_DIR*/*daemon_name*/dbrec.dat

- *HOME_DIR*/*daemon_name*/dbrec.keys

- *HOME_DIR*/*daemon_name*/pathseg.dat

- *HOME_DIR*/*daemon_name*/pathseg.keys

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

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

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

- The VOL database (files `tpvrdm.dat` and `tpvrdm.vsn.keys`)

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

7.6.2 Database Recovery Procedures

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

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

Procedure 17: Recovering the Databases

If you lose a database through disk spindle failure or through some form of external corruption, use the following procedure to recover it:

1. Stop DMF.

2. If you have configured the `run_copy_databases` task, copy the files from the directory with the most recent copy of the databases that were in *HOME_DIR*.

3. If you have **not** configured the `run_copy_databases` task, reload an old version of the daemon or tape MSP database. Typically, these will be from the most recent dump tapes of your file system.

4. Ensure that the default *JOURNAL_DIR/daemon_name* (or *JOURNAL_DIR/msp_name*) directory contains all of the time-ordered journal files since the last update of the older database.

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

   For the tape MSP, the journal files are named `atmsp_db.`*yyyymmdd[.hhmmss]*.

5. Note the time of the last database update from step 2.

6. Use `dmdbrecover` to update the old database with the journal entries from journal files identified in step 3.

Example 5: Database Recovery Example

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

1. Stop DMF.

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

JOURNAL_DIR/daemon_name

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

JOURNAL_DIR/msp_name

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

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

HOME_DIR/daemon_name

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

HOME_DIR/MSP_Name

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

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

```
dmdbrecover -n daemon_name dmd_db
dmdbrecover -n msp_name atmsp_db
```

Messages [A]

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

A.1 Message Format

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

A.1.1 Message Format for Catalog (CAT) Database and Daemon Database Comparisons

Error messages generated when comparing the CAT database to the daemon
database will start with the following phrase:

Bfid *bfid* –

The *bfid* is the bit file ID associated with the message.

The preceding phrase will be completed by one or more of the following
phrases:

```
missing from cat db
missing from daemon db
for vsn volume_serial_number chunk chunk_number msg1 msg2
```

In the above, *msgn* can be one of the following:

```
filesize < 0
chunkoffset < 0
chunklength < 0
zonenumber < 0
chunknumber <0
filesize < chunklength + chunkoffset
zonenumber
missing or improper vsn
filesize != file size in daemon entry (size)

no chunk for bytes msg1, msg2
```

In the above, *msgn* gives the byte range as *nnn* – *nnn*

```
nnn bytes duplicated
```

A.1.2 Message Format for Volume (VOL) Database and Catalog (CAT) Database and Daemon Database Comparisons

Error messages generated when comparing the VOL database to the CAT database will start with the following phrase:

```
Vsn vsn
```

The *vsn* is the volume serial number associated with the message.

The preceding phrase will be completed by one or more of the following phrases:

```
missing

eotpos < largest position in cat (3746)
eotchunk < largest chunk in cat (443)
eotzone < largest zone in cat (77)
chunksleft != number of cat chunks (256)
dataleft !=sum of cat chunk lengths (4.562104mb)

tapesize is bad
version is bad
blocksize is bad
zonesize is bad
eotchunk < chunksleft
dataleft > datawritten

volume is empty but msg1, msg2
```

In the above, *msgn* can be one of the following:

```
   hfull is on
   hsparse is on
   hrsv is on
   datawritten != 0
   eotpos != 1/0
   eotchunk != 1


volume is not empty but msg1, msg2
```

In the above, *msgn* is one of the following:

```
hfree is on
version < 4 but msg1, msg2
```

In the above, *msgn* can be one of the following:

```
volume contains new chunks
hfull is off
eotpos !=2/0
```

A.2 `dmcatadm` Message Interpretation

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

nnn `bytes duplicated`	Two or more chunks in the database contain data from the same region of the file.
`for vsn DMF001 chunk 77 chunkoffset < 0`	The `chunkoffset` value for chunk 77 on volume serial number (VSN) DMF001 is obviously bad because it is less than 0.
`for vsn DMF001 chunk 77 chunklength < 0`	The `chunklength` value for chunk 77 on VSN DMF001 is obviously bad because it is less than 0.
`for vsn DMF001 chunk 77 chunknumber < 0`	The `chunknumber` value for chunk 77 on VSN DMF001 is obviously bad because it is less than 0.
`for vsn DMF001 chunk 77 filesize < 0`	The `filesize` value for chunk 77 on DMF001 is obviously bad because it is less than 0.
`for vsn DMF001 chunk 77 filesize < chunklength + chunkoffset`	The value of `chunklength` plus `chunkoffset` should be less than or equal to the `filesize`. Therefore, one or more of these values is wrong.

`for vsn DMF001 chunk 77 missing or improper vsn`	The list of volume serial numbers for the chunk is improperly constructed. The list should contain one or more 6-character names separated by colons.
`for vsn DMF001 chunk 77 zonenumber < 0`	The `zonenumber` value for chunk 77 on DMF001 is obviously bad because it is less than 0.
`for vsn DMF001 chunk 77 zonenumber > chunknumber`	Either the `zonenumber` value or the `chunknumber` value for chunk 77 on DMF001 is wrong, because the `zonenumber` is larger than the `chunknumber` value. (Each zone contains at least two chunks, because the end-of-zone header on the tape counts as a chunk.)
`for vsn DMF001 chunk 77 filesize != file size in daemon entry (nnn)`	The `filesize` value in the chunk entry is different from the file size in the daemon record. If no daemon record was provided, this message indicates that more than one chunk exists for the bfid and that the `filesize` value is not the same for all the chunks.
`missing from cat db`	The daemon entry was not found in the CAT database.
`missing from daemon db`	No daemon entry was found for the entry in the CAT database.
`no chunk for bytes nnn – nnn`	There is no chunk that contains the specified bytes of the file.

A.3 `dmvoladm` Message Interpretation

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

`blocksize is bad`	The `tapesize` field for the tape is not a multiple of 4096.
`eotpos < largest position in cat (3746)`	The position for the EOT descriptor on the tape is less than the largest position of all the chunk entries for the tape.
`chunksleft != number of cat chunks (256)`	The number of chunks referencing the tape in the CAT database does not equal the number of chunks left recorded in the VOL entry for the tape.
`dataleft != sum of cat chunk lengths (4.562104mb)`	The sum of the chunks length for chunks referencing the tape in the CAT database does not equal the `dataleft` value recorded in the VOL entry for the tape.
`dataleft > datawritten`	The entry shows that more data remains on the tape than was written.
`eotchunk < chunksleft`	The entry shows that more chunks remain on the tape than were written.
`eotchunk < largest chunk in cat (443)`	The chunk number of the EOT descriptor on the tape is less than the largest chunk number of all the chunk entries for the tape.
`eotzone < largest zone in cat (77)`	The zone number of the EOT descriptor on the tape is less than the largest zone number of all the chunk entries for the tape.
`missing`	The volume was found in a chunk entry from the CAT database but is not in the VOL database.
`tapesize is bad`	The `tapesize` field for the tape is an impossible number.

`version is bad`	The `version` field for the tape is not 1, 2, or 3 (for a tape still containing data written by an old MSP) or 4 (for a tape written by this MSP).

`volume is empty but hfull is on`
`volume is empty but hsparse is on`
`volume is empty but hrsv is on`
	When a volume is empty, the hfull, hsparse, and hrsv hold flags should be off.

`volume is empty but datawritten != 0`
`volume is empty but eotpos != 1/0`
`volume is empty but eotchunk != 1`
	When the `hfree` hold flag is cleared, the `datawritten` field is set to 0, the `eotpos` field is set to 1/0, and the `eotchunk` is set to 1. The entry is inconsistent and should be checked.

`volume is not empty but hfree is on`	When a volume contains data, the `hfree` hold flag must be off.
`volume is not empty and version < 3 but volume contains new chunks`	One or more of the chunks associated with this volume were written by the advanced tape MSP, which only writes tapes that have a version value equal to 3. However, the `version` value of the tape is less than 3.
`volume is not empty and version < 3 but hfull is off`	Tapes containing data with a version value of less than 3 must have `hfull` set, because the MSP cannot append to the tape.
`volume is not empty and version < 3 but eotpos != 2/0`	Tapes imported from the old MSP only have one zone of data, so `eotpos` must be 2/0.
`zonesize is too small`	The `zonesize` field for the tape is an impossible number.

Glossary

active database entry

A valid daemon database entry. See also *soft-deleted database entry* and *hard-deleted database entry*.

alternate media

The media onto which migrated data blocks are stored, usually tapes.

automated space management

The combination of utilities that allows DMF to maintain a specified level of free space on a file system through automatic file migration.

base object

The configuration object that defines path name and file size parameters necessary for DMF operation.

bitfile ID

See *bfid*.

bfid

The bit file identifier, or bfid, is a unique identifier, assigned to each file during the migration process, that links a migrated file to its data on alternate media.

bfid set

The collection of database entries and the user file associated with a particular bfid.

bfid-set state

The sum of the states of the components that comprise a bfid set: the file state of any user file and the state of any database entries (incomplete, complete, soft-deleted, or active).

block

Physical unit of I/O to and from media, usually tape. The size of a block is determined by the type of device being written. A tape block is accompanied by a header identifying the chunk number, zone number, and its position within the chunk.

candidate list

A list that contains an entry for each file in a file system eligible for migration, ordered from largest file weight (first to be migrated) to smallest. This list is generated and used internally by dmfsmon(8). The dmscanfs(8) command prints similar file status information to standard output.

CAT records

The catalog (CAT) records in the tape MSP database that track which migrated files reside on which tape volumes.

chunk

That portion of a user file that fits on the current media (tape) volume. Most small files are written as single chunks. When a migrated file cannot fit onto a single volume, the file is split into chunks.

complete MSP daemon-database entry

An entry in the daemon database whose path field contains a key returned by its MSP, indicating that the MSP maintains a valid copy of the user file.

compression

The mechanism provided by the tape MSP for copying active data from volumes that contain largely obsolete data to volumes that contain mostly active data. This process is also known as *volume merging* or *tape merging*.

configuration object

A series of parameter definitions in the DMF configuration file that controls the way DMF operates. By changing the parameters associated with objects, you can modify the behavior of DMF.

configuration parameter

A string in the DMF configuration file that defines a part of a configuration object. By changing the values associated with these parameters, you can modify the behavior of DMF. The parameter serves as the name of the line. Some parameters are reserved words, some are supplied by the site.

daemon database

A database maintained by the DMF daemon. This database contains such information as the bfid, the MSP name, and MSP key for each copy of a migrated file.

daemon object

The configuration object that defines parameters necessary for dmdaemon(8) operation

data-pointer area

The portion of the inode that points to the file's data blocks.

device object

The configuration objects that define parameters for DMF's use of tape devices.

direct-access storage device (DASD)

An IBM disk drive.

DMF state

See *file state*.

dual-state file

A file whose data resides both online and offline.

dual-state file systems

Those file systems that have the necessary inode space to support dual-state files.

fhandle

See *file handle*.

file

An inode and its associated data blocks; an empty file has an inode but no data blocks.

file handle

The DMAPI identification for a file. You can use the dmscanfs(8), dmattr(1), and dmfind(1) commands to find file handles.

file state

The migration state of a file as indicated by the dmattr(1) command. A file can be regular (not migrated), migrating, dual-state, offline, unmigrating, never-migrated, or have an invalid DMF state.

freed file

A user file that has been migrated and whose data blocks have been released.

fully backed up file

A file that has one or more complete offline copies and no pending or incomplete offline copies.

hard-deleted database entry

An MSP database entry that has been removed from the daemon database and whose MSP copy has been discarded. See also *active database entry* and *soft-deleted database entry*.

inode

The portion of a file that contains the bfid, the state field, and the data pointers.

incomplete MSP daemon-database entry

An entry in the daemon database for an MSP 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.

media-specific process (MSP)

The daemon-like process by which data blocks are copied onto alternate media, and which assigns keys to identify the location of the migrated data.

migrated file

A file that has a bfid and whose offline copies (or copy) are completed. Migrated files can be *dual-state* or *offline*.

migrating file

A file that has a bfid but whose offline copies (or copy) are in progress.

MSP

See *media-specific process (MSP)*.

MSP database entry

The configuration objects that define parameters necessary for that MSP's operation. There is one MSP object for each MSP.

MSP objects

The configuration objects that define parameters necessary for that MSP's operation

nonmigrated file

A file that does not have a bfid or any offline copies. See *regular file*.

offline file

A file whose inode contains a bfid but whose disk blocks have been removed. The file's data exists elsewhere in copies on alternate media.

offline pointer

In tape MSP processing, a character string that the MSP returns to the daemon to indicate how a file is to be retrieved. For the tape MSP, the offline pointer is the character key into the MSP catalog (CAT) records of the database.

orphan chunks

Unused chunks in the tape MSP catalog (CAT) database entries resulting from the removal of migrated files.

orphan database entries

Unused database entries resulting from the removal of migrated files during a period in which the DMF daemon is not running.

parameter

See *configuration parameter*.

policy objects

The configuration objects that specify parameters to determine MSP selection, automated space management policies, and/or file weight calculations in automatic space management.

recall

To request that a migrated file's data be moved back (unmigrated) onto the file system disk, either by explicitly entering the dmget(1) command or by executing another command that will open the file, such as the vi(1) command.

regular file

DMF considers a regular file to be one with no bfid and no offline copies.

snapshot

The information about all bfid sets that is collected and analyzed by dmaudit(8). The snapshot analysis is available from the report function.

soft-deleted database entry

A daemon database entry for which the MSP copy of the data is no longer valid. Data remains on the alternate media until the database entry is hard-deleted. See also *active database entry* and *hard-deleted database entry*.

sparse tape

A tape containing only a small amount of active information.

special file

UNIX special files are never migrated by DMF.

state field

The field in the inode that shows the current migration state of a file.

tape block

See *block.*

tape chunk

See *chunk*.

task

A process initiated by the DMF event mechanism. Configuration tasks that allow certain recurring administrative duties to be automated are defined with configuration file parameters.

unmigratable file

A file that the daemon will never select as a migration candidate.

unmigrate

See *recall*.

unmigration file

A special file created in the user file's file system by the DMF daemon when a file is recalled. The unmigration file holds the data pointers until the MSP process successfully copies the file's data back onto the file system disk from the alternate media; when the copy is complete, the daemon moves the data pointers from the unmigration file inode back into the user file's inode.

unmigration directory

A directory in which unmigration files are stored.

voided bfid-set state

A bfid-set state that consists of one or more soft-deleted daemon database entries, either incomplete or complete. There is no user file.

voiding the bfid

The process of removing the bfid from the user file inode and soft-deleting all associated database entries.

VOL records

The volume (VOL) records in the tape MSP database that contain information about each tape volume that exists in the pool of tapes used by the tape MSP.

volume merging

The mechanism provided by the tape MSP for copying active data from volumes that contain largely obsolete data to volumes that contain mostly active data.

zone

A logical grouping of chunks. Zones are separated by file marks and are the smallest block-addressable unit on the tape volume. The target size of a zone is configurable by media type.

Index

W

Weighting
  of files for migration
    configuration parameter definition, 41
    procedure for configuring, 43
wfage keyword
  dmvoladm command, 114
wfdate keyword
  dmvoladm command, 114
wfdate record
  dmvoladm text field order, 121
when clause
  configuration file
    definition, 42
WRITE_CHECKSUM configuration parameter
  device object
    definition, 54
writeage keyword
  dmcatadm command, 107
writedata record
  dmcatadm text field order, 109
writedate keyword
  dmcatadm command, 107

X

xfsdump command, 131
xfsrestore command, 131

Z

ZONE_SIZE configuration parameter
  dmatmsp
    definition, 54
zoneblockid keyword
  dmcatadm command, 107
zonenumber keyword
  dmcatadm command, 107
zonepos keyword
  dmcatadm command, 107
zoneposition record
  dmcatadm text field order, 110
Zones
  DMF tape concepts, 95