Storage Foundation 8.0 Administrator's Guide - Linux

Last updated: 2021-12-21

Legal Notice

THE DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID. VERITAS TECHNOLOGIES LLC SHALL NOT BE LIABLE FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS DOCUMENTATION. THE INFORMATION CONTAINED IN THIS DOCUMENTATION IS SUBJECT TO CHANGE WITHOUT NOTICE.

Veritas Technologies LLC
2625 Augustine Drive
Santa Clara, CA 95054
http://www.veritas.com

Technical Support

Technical Support maintains support centers globally. All support services will be delivered in accordance with your support agreement and the then-current enterprise technical support policies. For information about our support offerings and how to contact Technical Support, visit our website:
https://www.veritas.com/support

You can manage your Veritas account information at the following URL:
https://my.veritas.com

If you have questions regarding an existing support agreement, please email the support agreement administration team for your region as follows:

Worldwide (except Japan)	CustomerCare@veritas.com
Japan	CustomerCare_Japan@veritas.com

Documentation

Make sure that you have the current version of the documentation. Each document displays the date of the last update on page 2. The latest documentation is available on the Veritas website:
https://sort.veritas.com/documents

Documentation feedback

Your feedback is important to us. Suggest improvements or report errors or omissions to the documentation. Include the document title, document version, chapter title, and section title of the text on which you are reporting. Send feedback to:
infoscaledocs@veritas.com

You can also see documentation information or ask a question on the Veritas community site:
http://www.veritas.com/community/

Veritas Services and Operations Readiness Tools (SORT)

Veritas Services and Operations Readiness Tools (SORT) is a website that provides information and tools to automate and simplify certain time-consuming administrative tasks. Depending on the product, SORT helps you prepare for installations and upgrades, identify risks in your datacenters, and improve operational efficiency. To see what services and tools SORT provides for your product, see the data sheet:
https://sort.veritas.com/data/support/SORT_Data_Sheet.pdf

Contents

Section 1

Introducing Storage Foundation

Chapter 1

Overview of Storage Foundation

This chapter includes the following topics:

About Storage Foundation

Storage Foundation is a storage management solution to enable robust, manageable, and scalable storage deployment. SF maximizes your storage efficiency, availability, agility, and performance across heterogeneous server and storage platforms.

Storage Foundation consists of product components and features that can be used individually and together to improve performance, resilience and ease of management for your storage and applications.

Table 1-1 describes the components of Storage Foundation.

Table 1-1 Storage Foundation components

Component	Description
Dynamic Multi-Pathing (DMP)	Manages the I/O performance and path availability of the physical storage devices that are configured on the system. DMP creates DMP metadevices across all of the paths to each LUN. DMP uses the DMP metadevices to manage path failover and I/O load balancing across the paths to the physical devices. DMP metadevices provide the foundation for Veritas Volume Manager (VxVM) and Veritas File System (VxFS). DMP also supports native operating system volumes and file systems on DMP devices.
Veritas Volume Manager (VxVM)	Provides a logical storage abstraction layer or storage management between your operating system devices and your applications. VxVM enables you to create logical devices called volumes on the physical disks and LUNs.The applications such as file systems or databases access the volumes as if the volumes were physical devices but without the physical limitations. VxVM features enable you to configure, share, manage, and optimize storage I/O performance online without interrupting data availability. Additional VxVM features enhance fault tolerance and fast recovery from disk failure or storage array failure.
Veritas File System (VxFS)	Provides a high-performance journaling file system. VxFS is designed for use in operating environments that deal with large amounts of data and that require high performance and continuous availability. VxFS features provide quick-recovery for applications, scalable performance, continuous availability, increased I/O throughput, and increased structural integrity.
Replicator (VR)	Enables you to maintain a consistent copy of application data at one or more remote locations for disaster recovery. Replicator provides the flexibility of block-based continuous replication with Volume Replicator (VVR) and file-based periodic replication with File Replicator (VFR). Replicator option is a separately-licensable feature of Storage Foundation.

A related product, Veritas Operations Manager, provides a centralized management console that you can use with Veritas InfoScale products.

See “About Veritas InfoScale Operations Manager” on page 30.

Note: The commands used for the Red Hat Enterprise Linux (RHEL) operating system in this document also apply to supported RHEL-compatible distributions.

About Dynamic Multi-Pathing (DMP)

Dynamic Multi-Pathing (DMP) provides multi-pathing functionality for the operating system native devices that are configured on the system. DMP creates DMP metadevices (also known as DMP nodes) to represent all the device paths to the same physical LUN.

DMP is available as a component of Storage Foundation. DMP supports Veritas Volume Manager (VxVM) volumes on DMP metadevices, and Veritas File System (VxFS) file systems on those volumes.

DMP metadevices support the OS native logical volume manager (LVM). You can create LVM volumes and volume groups on DMP metadevices.

Veritas Volume Manager (VxVM) volumes and disk groups can co-exist with LVM volumes and volume groups. But, each device can only support one of the types. If a disk has a VxVM label, then the disk is not available to LVM. Similarly, if a disk is in use by LVM, then the disk is not available to VxVM.

About Veritas Volume Manager

Veritas™ Volume Manager (VxVM) by Veritas is a storage management subsystem that allows you to manage physical disks and logical unit numbers (LUNs) as logical devices called volumes. A VxVM volume appears to applications and the operating system as a physical device on which file systems, databases, and other managed data objects can be configured.

VxVM provides easy-to-use online disk storage management for computing environments and Storage Area Network (SAN) environments. By supporting the Redundant Array of Independent Disks (RAID) model, VxVM can be configured to protect against disk and hardware failure, and to increase I/O throughput. Additionally, VxVM provides features that enhance fault tolerance and fast recovery from disk failure or storage array failure.

VxVM overcomes restrictions imposed by hardware disk devices and by LUNs by providing a logical volume management layer. This allows volumes to span multiple disks and LUNs.

VxVM provides the tools to improve performance and ensure data availability and integrity. You can also use VxVM to dynamically configure storage while the system is active.

About Veritas File System

A file system is simply a method for storing and organizing computer files and the data they contain to make it easy to find and access them. More formally, a file system is a set of abstract data types (such as metadata) that are implemented for the storage, hierarchical organization, manipulation, navigation, access, and retrieval of data.

Veritas File System (VxFS) was the first commercial journaling file system. With journaling, metadata changes are first written to a log (or journal) then to disk. Since changes do not need to be written in multiple places, throughput is much faster as the metadata is written asynchronously.

VxFS is also an extent-based, intent logging file system. VxFS is designed for use in operating environments that require high performance and availability and deal with large amounts of data.

The maximum size of the file system you can create depends on the block size.

Block Size	Currently-Supported Maximum File System Size
1024 bytes	68,719,472,624 sectors (≈32 TB)
2048 bytes	137,438,945,248 sectors (≈64 TB)
4096 bytes	274,877,890,496 sectors (≈128 TB)
8192 bytes	549,755,780,992 sectors (≈256 TB)

VxFS major components include:

File system logging	About the Veritas File System intent log
Extents	About extents
File system disk layouts	About file system disk layouts

About the Veritas File System intent log

Most file systems rely on full structural verification by the `fsck` utility as the only means to recover from a system failure. For large disk configurations, this involves a time-consuming process of checking the entire structure, verifying that the file system is intact, and correcting any inconsistencies. VxFS provides fast recovery with the VxFS intent log and VxFS intent log resizing features.

VxFS reduces system failure recovery times by tracking file system activity in the VxFS intent log. This feature records pending changes to the file system structure in a circular intent log. The intent log recovery feature is not readily apparent to users or a system administrator except during a system failure. By default, VxFS file systems log file transactions before they are committed to disk, reducing time spent recovering file systems after the system is halted unexpectedly.

During system failure recovery, the VxFS `fsck` utility performs an intent log replay, which scans the intent log and nullifies or completes file system operations that were active when the system failed. The file system can then be mounted without requiring a full structural check of the entire file system. Replaying the intent log might not completely recover the damaged file system structure if there was a disk hardware failure; hardware problems might require a complete system check using the `fsck` utility provided with VxFS.

The `mount` command automatically runs the VxFS `fsck` command to perform an intent log replay if the mount command detects a dirty log in the file system. This functionality is only supported on a file system mounted on a Veritas Volume Manager (VxVM) volume, and is supported on cluster file systems.

See the `fsck_vxfs`(1M) manual page and `mount_vxfs`(1M) manual page.

The VxFS intent log is allocated when the file system is first created. The size of the intent log is based on the size of the file system—the larger the file system, the larger the intent log. You can resize the intent log at a later time by using the `fsadm` commnad.

See the `fsadm_vxfs`(1M) manual page.

The maximum default intent log size for disk layout Version 7 or later is 256 megabytes.

Note: Inappropriate sizing of the intent log can have a negative impact on system performance.

See "Intent log size" on page 164.

About extents

An extent is a contiguous area of storage in a computer file system, reserved for a file. When starting to write to a file, a whole extent is allocated. When writing to the file again, the data continues where the previous write left off. This reduces or eliminates file fragmentation. An extent is presented as an address-length pair, which identifies the starting block address and the length of the extent (in file system or logical blocks). Since Veritas File System (VxFS) is an extent-based file system, addressing is done through extents (which can consist of multiple blocks) rather than in single-block segments. Extents can therefore enhance file system throughput.

Extents allow disk I/O to take place in units of multiple blocks if storage is allocated in contiguous blocks. For sequential I/O, multiple block operations are considerably faster than block-at-a-time operations; almost all disk drives accept I/O operations on multiple blocks.

Extent allocation only slightly alters the interpretation of addressed blocks from the inode structure compared to block-based inodes. A VxFS inode references 10 direct extents, each of which are pairs of starting block addresses and lengths in blocks.

Disk space is allocated in 512-byte sectors to form logical blocks. VxFS supports logical block sizes of 1024, 2048, 4096, and 8192 bytes. The default block size is 1 KB for file system sizes of up to 2 TB, and 8 KB for file system sizes 2 TB or larger.

About file system disk layouts

The disk layout is the way file system information is stored on disk. On Veritas File System (VxFS), several disk layout versions are supported to provide new features and specific UNIX environments.

You can use one of the following commands to upgrade the disk layout version.

`vxupgrade`

Upgrades an existing VxFS file system to a supported disk layout version while the file system remains online.

See the `vxupgrade`(1M) manual page.

`vxfsconvert`

Upgrades a no-longer supported disk layout version to a supported version while the file system is not mounted.

The `vxfsconvert` command can also be used to convert a native file system (ext2, ext3, and ext4) to VxFS, while the file system is not mounted.

See the `vxfsconvert`(1M) manual page.

Table 1-2 lists the supported disk layout versions.

Table 1-2 Supported disk layout versions

Version	Supported features
Version 13	■ Added support for WORM ■ Clone creation performance improvement when extended file attributes are used
Version 14	Supports SmartIO FEL-based caching

Table 1-2 Supported disk layout versions *(continued)*

Version	Supported features
Version 15	■ Performance enhancements in SELinux attribute storage and retrieval ■ Secure Clock support for WORM files
Version 16	Audit logging support for WORM files
Version 17	■ Added support for soft WORM-enabled objects, non-modifiable storage checkpoints, and retention period beyond the year 2038 for WORM-enabled objects ■ Audit logging is decoupled from file system layouts and the `fsck` module supports backward compatibility from higher file system layout versions or audit log versions

Currently, only versions 13, 14, 15, 16, and 17 can be created and mounted. Versions 6, 7, 8, 9, 10, 11, and 12 can be mounted, but only for upgrading to a supported version.

About Storage Foundation Cluster File System (SFCFS)

Extends the VxFS file system for use with multiple systems (or nodes) in a cluster. CFS enables you to simultaneously mount the same file system on multiple nodes. CFS features simplify management, improve performance, and enable fast failover of applications and databases.

About Veritas InfoScale Operations Manager

Veritas InfoScale Operations Manager provides a centralized management console for Veritas InfoScale products. You can use Veritas InfoScale Operations Manager to monitor, visualize, and manage storage resources and generate reports.

Veritas recommends using Veritas InfoScale Operations Manager to manage Storage Foundation and Cluster Server environments.

You can download Veritas InfoScale Operations Manager from:

https://www.veritas.com/content/support/en_US/downloads

Refer to the Veritas InfoScale Operations Manager documentation for installation, upgrade, and configuration instructions.

The Veritas Enterprise Administrator (VEA) console is no longer packaged with Veritas InfoScale products. If you want to continue using VEA, a software version is available for download from:

https://www.veritas.com/form/trialware/vcs-utilities

Storage Foundation Management Server is deprecated.

About Veritas Replicator

Veritas Replicator provides organizations with a comprehensive solution for heterogeneous data replication. As an option to Storage Foundation, Veritas Replicator enables cost-effective replication of data over IP networks, giving organizations an extremely flexible, storage hardware independent alternative to traditional array-based replication architectures. Veritas Replicator provides the flexibility of block-based continuous replication with Volume Replicator Option (VVR) and file-based periodic replication with File Replicator Option (VFR).

What is VFR?

Veritas File Replicator (VFR) enables cost-effective periodic replication of data over IP networks, giving organizations an extremely flexible storage independent data availability solution for disaster recovery and off-host processing. With flexibility of scheduling the replication intervals to match the business requirements, Veritas File Replicator tracks all updates to the file system and replicates these updates at the end of the configured time interval. VFR leverages data deduplication provided by Veritas File System (VxFS) to reduce the impact that replication can have on scarce network resources. VFR is included, by default, with Virtual Store 6.0 on Linux and is available as an option with Storage Foundation and associated products on Linux.

VFR has WORM replication support. WORM replication is required to protect files from remove or rename operation till retention period on target side.

For more details, see the*Storage Foundation and High Availability Solutions Replication Administrator’s Guide*

Features of VFR

Veritas File Replicator (VFR) includes the following features:

- Supports periodic replication of a subset of a file system ranging from a single file to an entire file system.
- Supports reversible data transfer. The target of replication may become the source at runtime, with the former source system becoming a target.

- Supports automatic recovery from the last good successfully replicated point in time image.
- Periodically replicates changes. The interval is configurable by the user.
- Supports deduplication to increase storage efficiency on the target system.
- Supports protection of the target file system from accidental writes.
- Supports IPv4, IPv6, and dual-stack configurations.

See the *Storage Foundation and High Availability Solutions Replication Administrator’s Guide* for more information.

Use cases for Storage Foundation

Storage Foundation components and features can be used individually and together to improve performance, resilience, and ease of management for your storage and applications. Storage Foundation features can be used for:

- Improving database performance: you can use Storage Foundation database accelerators to improve I/O performance. SFHA Solutions database accelerators achieve the speed of raw disk while retaining the management features and convenience of a file system.
- Optimizing thin array usage: you can use Storage Foundation thin provisioning and thin reclamation solutions to set up and maintain thin storage.
- Backing up and recovering data: you can use Storage Foundation Flashsnap, Storage Checkpoints, and NetBackup point-in-time copy methods to back up and recover your data.
- Processing data off-host: you can avoid performance loss to your production hosts by using Storage Foundation volume snapshots.
- Optimizing test and development environments: you can optimize copies of your production database for test, decision modeling, and development purposes using Storage Foundation point-in-time copy methods.
- Optimizing virtual desktop environments: you can use Storage Foundation FileSnap to optimize your virtual desktop environment.
- Maximizing storage utilization: you can use Storage Foundation SmartTier to move data to storage tiers based on age, priority, and access rate criteria.
- Maximizing storage utilization: you can use Storage Foundation Flexible Storage Sharing for data redundancy, high availability, and disaster recovery, without physically shared storage.

- Migrating your data: you can use Storage Foundation Portable Data Containers to easily and reliably migrate data from one environment to another.

For a supplemental guide that documents Storage Foundation use case solutions using example scenarios: See the *Veritas InfoScale Solutions Guide*.

How Dynamic Multi-Pathing works

This chapter includes the following topics:

How DMP works

Dynamic Multi-Pathing (DMP) provides greater availability, reliability, and performance by using the path failover feature and the load balancing feature. These features are available for multiported disk arrays from various vendors.

Disk arrays can be connected to host systems through multiple paths. To detect the various paths to a disk, DMP uses a mechanism that is specific to each supported array. DMP can also differentiate between different enclosures of a supported array that are connected to the same host system.

See "Discovering and configuring newly added disk devices" on page 193.

The multi-pathing policy that DMP uses depends on the characteristics of the disk array.

DMP supports the following standard array types:

Table 2-1

Array type	Description
Active/Active (A/A)	Allows several paths to be used concurrently for I/O. Such arrays allow DMP to provide greater I/O throughput by balancing the I/O load uniformly across the multiple paths to the LUNs. In the event that one path fails, DMP automatically routes I/O over the other available paths.
Asymmetric Active/Active (A/A-A)	A/A-A or Asymmetric Active/Active arrays can be accessed through secondary storage paths with little performance degradation. The behavior is similar to ALUA, except that it does not support the SCSI commands that an ALUA array supports.
Asymmetric Logical Unit Access (ALUA)	DMP supports all variants of ALUA.
Active/Passive (A/P)	Allows access to its LUNs (logical units; real disks or virtual disks created using hardware) via the primary (active) path on a single controller (also known as an access port or a storage processor) during normal operation. In implicit failover mode (or autotrespass mode), an A/P array automatically fails over by scheduling I/O to the secondary (passive) path on a separate controller if the primary path fails. This passive port is not used for I/O until the active port fails. In A/P arrays, path failover can occur for a single LUN if I/O fails on the primary path. This array mode supports concurrent I/O and load balancing by having multiple primary paths into a controller. This functionality is provided by a controller with multiple ports, or by the insertion of a SAN switch between an array and a controller. Failover to the secondary (passive) path occurs only if all the active primary paths fail.

Table 2-1 *(continued)*

Array type	Description
Active/Passive in explicit failover mode or non-autotrespass mode (A/PF)	The appropriate command must be issued to the array to make the LUNs fail over to the secondary path. This array mode supports concurrent I/O and load balancing by having multiple primary paths into a controller. This functionality is provided by a controller with multiple ports, or by the insertion of a SAN switch between an array and a controller. Failover to the secondary (passive) path occurs only if all the active primary paths fail.
Active/Passive with LUN group failover (A/PG)	For Active/Passive arrays with LUN group failover (A/PG arrays), a group of LUNs that are connected through a controller is treated as a single failover entity. Unlike A/P arrays, failover occurs at the controller level, and not for individual LUNs. The primary controller and the secondary controller are each connected to a separate group of LUNs. If a single LUN in the primary controller’s LUN group fails, all LUNs in that group fail over to the secondary controller. This array mode supports concurrent I/O and load balancing by having multiple primary paths into a controller. This functionality is provided by a controller with multiple ports, or by the insertion of a SAN switch between an array and a controller. Failover to the secondary (passive) path occurs only if all the active primary paths fail.

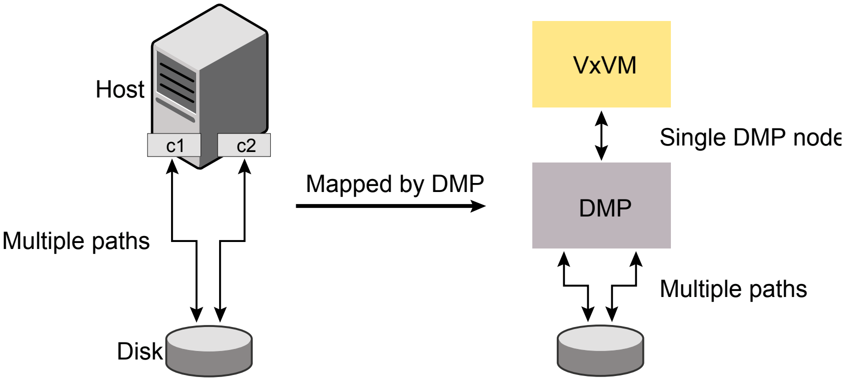
An array policy module (APM) may define array types to DMP in addition to the standard types for the arrays that it supports.

Storage Foundation uses DMP metanodes (DMP nodes) to access disk devices connected to the system. For each disk in a supported array, DMP maps one node to the set of paths that are connected to the disk. Additionally, DMP associates the appropriate multi-pathing policy for the disk array with the node.

For disks in an unsupported array, DMP maps a separate node to each path that is connected to a disk. The raw and block devices for the nodes are created in the directories `/dev/vx/rdmp` and `/dev/vx/dmp` respectively.

Figure 2-1 shows how DMP sets up a node for a disk in a supported disk array.

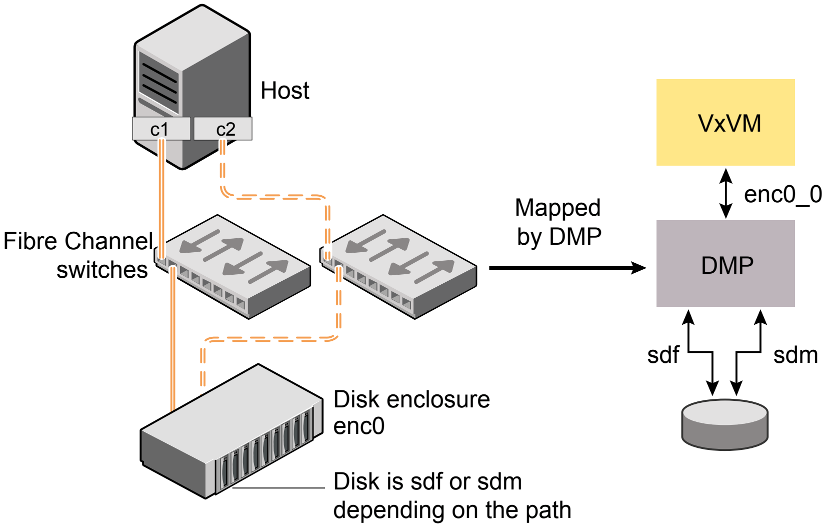
Figure 2-1 How DMP represents multiple physical paths to a disk as one node



DMP implements a disk device naming scheme that allows you to recognize to which array a disk belongs.

Figure 2-2 shows an example where two paths, `sdf` and `sdm`, exist to a single disk in the enclosure, but VxVM uses the single DMP node, `enc0_0`, to access it.

Figure 2-2 Example of multi-pathing for a disk enclosure in a SAN environment



See “About enclosure-based naming” on page 38.

See “Changing the disk device naming scheme” on page 286.

See “Discovering and configuring newly added disk devices” on page 193.

Device discovery

Device discovery is the term used to describe the process of discovering the disks that are attached to a host. This feature is important for DMP because it needs to support a growing number of disk arrays from a number of vendors. In conjunction with the ability to discover the devices attached to a host, the Device Discovery service enables you to add support for new disk arrays. The Device Discovery uses a facility called the Device Discovery Layer (DDL).

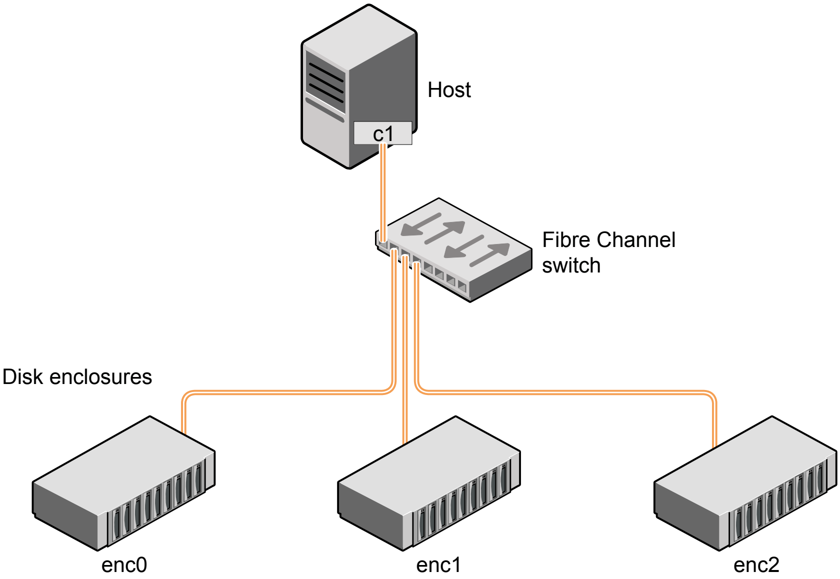
The DDL enables you to add support for new disk arrays without the need for a reboot.

About enclosure-based naming

Enclosure-based naming provides an alternative to operating system-based device naming. In a Storage Area Network (SAN) that uses Fibre Channel switches, information about disk location provided by the operating system may not correctly indicate the physical location of the disks. Enclosure-based naming allows SF to access enclosures as separate physical entities. By configuring redundant copies of your data on separate enclosures, you can safeguard against failure of one or more enclosures.

Figure 2-3 shows a typical SAN environment where host controllers are connected to multiple enclosures through a Fibre Channel switch.

Figure 2-3 Example configuration for disk enclosures connected through a Fibre Channel switch



In such a configuration, enclosure-based naming can be used to refer to each disk within an enclosure. For example, the device names for the disks in enclosure `enc0` are named `enc0_0`, `enc0_1`, and so on. The main benefit of this scheme is that it lets you quickly determine where a disk is physically located in a large SAN configuration.

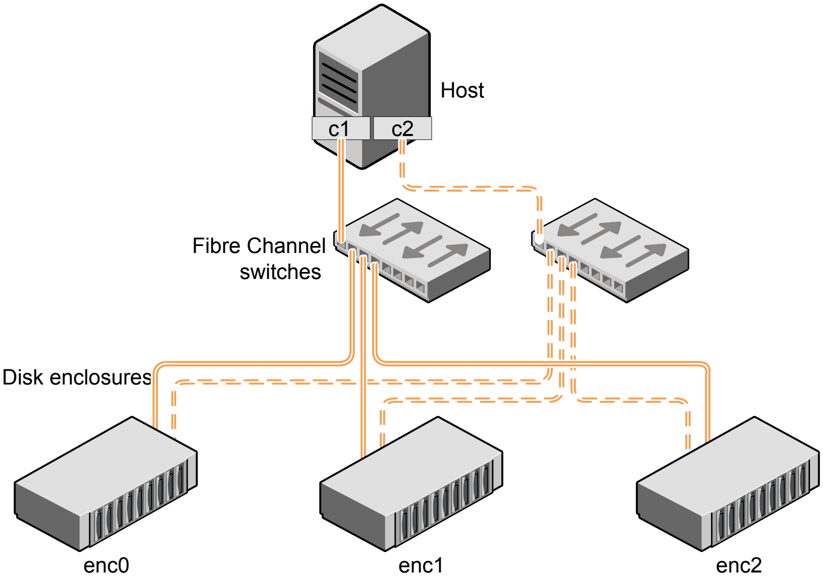
In most disk arrays, you can use hardware-based storage management to represent several physical disks as one LUN to the operating system. In such cases, VxVM also sees a single logical disk device rather than its component disks. For this reason, when reference is made to a disk within an enclosure, this disk may be either a physical disk or a LUN.

Another important benefit of enclosure-based naming is that it enables VxVM to avoid placing redundant copies of data in the same enclosure. This is a good thing to avoid as each enclosure can be considered to be a separate fault domain. For example, if a mirrored volume were configured only on the disks in enclosure `enc1`, the failure of the cable between the switch and the enclosure would make the entire volume unavailable.

If required, you can replace the default name that SF assigns to an enclosure with one that is more meaningful to your configuration.

Figure 2-4 shows a High Availability (HA) configuration where redundant-loop access to storage is implemented by connecting independent controllers on the host to separate switches with independent paths to the enclosures.

Figure 2-4 Example HA configuration using multiple switches to provide redundant loop access



Such a configuration protects against the failure of one of the host controllers (`c1` and `c2`), or of the cable between the host and one of the switches. In this example, each disk is known by the same name to VxVM for all of the paths over which it can be accessed. For example, the disk device `enc0_0` represents a single disk for which two different paths are known to the operating system, such as `sdf` and `sdm`.

See "Changing the disk device naming scheme" on page 286.

To take account of fault domains when configuring data redundancy, you can control how mirrored volumes are laid out across enclosures.

How DMP monitors I/O on paths

In VxVM prior to release 5.0, DMP had one kernel daemon (`errord`) that performed error processing, and another (`restored`) that performed path restoration activities.

From release 5.0, DMP maintains a pool of kernel threads that are used to perform such tasks as error processing, path restoration, statistics collection, and SCSI request callbacks. The name `restored` has been retained for backward compatibility.

One kernel thread responds to I/O failures on a path by initiating a probe of the host bus adapter (HBA) that corresponds to the path. Another thread then takes the appropriate action according to the response from the HBA. The action taken can be to retry the I/O request on the path, or to fail the path and reschedule the I/O on an alternate path.

The restore kernel task is woken periodically (by default, every 5 minutes) to check the health of the paths, and to resume I/O on paths that have been restored. As some paths may suffer from intermittent failure, I/O is only resumed on a path if the path has remained healthy for a given period of time (by default, 5 minutes). DMP can be configured with different policies for checking the paths.

See “Configuring DMP path restoration policies” on page 254.

The statistics-gathering task records the start and end time of each I/O request, and the number of I/O failures and retries on each path. DMP can be configured to use this information to prevent the SCSI driver being flooded by I/O requests. This feature is known as I/O throttling.

If an I/O request relates to a mirrored volume, VxVM specifies the FAILFAST flag. In such cases, DMP does not retry failed I/O requests on the path, and instead marks the disks on that path as having failed.

See “Path failover mechanism” on page 41.

See “I/O throttling” on page 42.

Path failover mechanism

DMP enhances system availability when used with disk arrays having multiple paths. In the event of the loss of a path to a disk array, DMP automatically selects the next available path for I/O requests without intervention from the administrator.

DMP is also informed when a connection is repaired or restored, and when you add or remove devices after the system has been fully booted (provided that the operating system recognizes the devices correctly).

If required, the response of DMP to I/O failure on a path can be tuned for the paths to individual arrays. DMP can be configured to time out an I/O request either after a given period of time has elapsed without the request succeeding, or after a given number of retries on a path have failed.

See “Configuring the response to I/O failures” on page 249.

Subpaths Failover Group (SFG)

A subpaths failover group (SFG) represents a group of paths which could fail and restore together. When an I/O error is encountered on a path in an SFG, DMP does proactive path probing on the other paths of that SFG as well. This behavior adds greatly to the performance of path failover thus improving I/O performance. Currently the criteria followed by DMP to form the subpaths failover groups is to bundle the paths with the same endpoints from the host to the array into one logical storage failover group.

See “Configuring Subpaths Failover Groups (SFG)” on page 252.

Low Impact Path Probing (LIPP)

The restore daemon in DMP keeps probing the LUN paths periodically. This behavior helps DMP to keep the path states up-to-date even when no I/O occurs on a path. Low Impact Path Probing adds logic to the restore daemon to optimize the number of the probes performed while the path status is being updated by the restore daemon. This optimization is achieved with the help of the logical subpaths failover groups. With LIPP logic in place, DMP probes only a limited number of paths within a subpaths failover group (SFG), instead of probing all the paths in an SFG. Based on these probe results, DMP determines the states of all the paths in that SFG.

See “Configuring Low Impact Path Probing (LIPP)” on page 252.

I/O throttling

If I/O throttling is enabled, and the number of outstanding I/O requests builds up on a path that has become less responsive, DMP can be configured to prevent new I/O requests being sent on the path either when the number of outstanding I/O requests has reached a given value, or a given time has elapsed since the last successful I/O request on the path. While throttling is applied to a path, the new I/O requests on that path are scheduled on other available paths. The throttling is removed from the path if the HBA reports no error on the path, or if an outstanding I/O request on the path succeeds.

See “Configuring the I/O throttling mechanism” on page 251.

Load balancing

By default, DMP uses the Minimum Queue I/O policy for load balancing across paths for all array types. Load balancing maximizes I/O throughput by using the total bandwidth of all available paths. I/O is sent down the path that has the minimum outstanding I/Os.

For Active/Passive (A/P) disk arrays, I/O is sent down the primary paths. If all of the primary paths fail, I/O is switched over to the available secondary paths. As the continuous transfer of ownership of LUNs from one controller to another results in severe I/O slowdown, load balancing across primary and secondary paths is not performed for A/P disk arrays unless they support concurrent I/O.

For other arrays, load balancing is performed across all the currently active paths.

You can change the I/O policy for the paths to an enclosure or disk array. This operation is an online operation that does not impact the server or require any downtime.

DMP in a clustered environment

In a clustered environment where Active/Passive (A/P) type disk arrays are shared by multiple hosts, all nodes in the cluster must access the disk through the same physical storage controller port. Accessing a disk through multiple paths simultaneously can severely degrade I/O performance (sometimes referred to as the ping-pong effect). Path failover on a single cluster node is also coordinated across the cluster so that all the nodes continue to share the same physical path.

Prior to release 4.1 of VxVM, the clustering and DMP features could not handle automatic failback in A/P arrays when a path was restored, and did not support failback for explicit failover mode arrays. Failback could only be implemented manually by running the `vxdctl enable` command on each cluster node after the path failure had been corrected. From release 4.1, failback is now an automatic cluster-wide operation that is coordinated by the master node. Automatic failback in explicit failover mode arrays is also handled by issuing the appropriate low-level command.

Note: Support for automatic failback of an A/P array requires that an appropriate Array Support Library (ASL) is installed on the system. An Array Policy Module (APM) may also be required.

See “About discovering disks and dynamically adding disk arrays” on page 195.

For Active/Active type disk arrays, any disk can be simultaneously accessed through all available physical paths to it. In a clustered environment, the nodes do not need to access a disk through the same physical path.

See “How to administer the Device Discovery Layer” on page 197.

See “Configuring Array Policy Modules” on page 256.

About enabling or disabling controllers with shared disk groups

Prior to release 5.0, Veritas Volume Manager (VxVM) did not allow enabling or disabling of paths or controllers connected to a disk that is part of a shared Veritas Volume Manager disk group. From VxVM 5.0 onward, such operations are supported on shared DMP nodes in a cluster.

Veritas Volume Manager co-existence with Oracle Automatic Storage Management disks

Automatic Storage Management (ASM) disks are the disks used by Oracle Automatic Storage Management software. Veritas Volume Manager (VxVM) co-exists with Oracle ASM disks, by recognizing the disks as the type Oracle ASM. VxVM protects ASM disks from any operations that may overwrite the disk. VxVM classifies and displays the ASM disks as ASM format disks. You cannot initialize an ASM disk, or perform any VxVM operations that may overwrite the disk.

If the disk is claimed as an ASM disk, disk initialization commands fail with an appropriate failure message. The `vxdisk init` command and the `vxdisksetup` command fail, even if the force option is specified. The `vxprivutil` command also fails for disks under ASM control, to prevent any on-disk modification of the ASM device.

If the target disk is under ASM control, any rootability operations that overwrite the target disk fail. A message indicates that the disk is already in use as an ASM disk. The rootability operations include operations to create a VM root image (`vxcp_lvmroot` command) , create a VM root mirror (`vxrootmir` command), or restore the LVM root image (`vxres_lvmroot` command). The `vxdestroy_lvmroot` command also fails for ASM disks, since the target disk is not under LVM control as expected.

Disks that ASM accessed previously but that no longer belong to an ASM disk group are called FORMER ASM disks. If you remove an ASM disk from ASM control, VxVM labels the disk as a FORMER ASM disk. VxVM enforces the same restrictions for FORMER ASM disks as for ASM disks, to enable ASM to reuse the disk in the future. To use a FORMER ASM disk with VxVM, you must clean the disk of ASM information after you remove the disk from ASM control. If a disk initialization command is issued on a FORMER ASM disk, the command fails. A message indicates that the disk must be cleaned up before the disk can be initialized for use with VxVM.

To remove a FORMER ASM disk from ASM control for use with VxVM

1. Clean the disk with the `dd` command to remove all ASM identification information on it. For example:

   ```
   dd if=/dev/zero of=/dev/rdsk/<wholedisk|partition> count=1 bs=1024
   ```

   where *wholedisk* is a disk name in the format: `cxtydz`

   where *partition* is a partition name in the format:`cxtydzsn`

2. Perform a disk scan:

   ```
   # vxdisk scandisks
   ```

To view the ASM disks

◆ You can use either of the following commands to display ASM disks:

The `vxdisk list` command displays the disk type as `ASM`.

```
# vxdisk list
DEVICE       TYPE            DISK         GROUP        STATUS
Disk_0s2     auto:LVM        -            -            LVM
Disk_1       auto:ASM        -            -            ASM
EVA4K6K0_0   auto            -            -            online
EVA4K6K0_1   auto            -            -            online
```

You

To check if a particular disk is under ASM control

◆ Use the `vxmediadisc` utility to check if a particular disk is under ASM control.

```
# /etc/vx/diag.d/vxmediadisc 3pardata0_2798
 3pardata0_2799   ACTIVE
```

Alternatively, use the utility to check if the disk is under control of any foreign software like LVM or ASM:

```
# /etc/vx/bin/vxisforeign 3pardata0_2799
 3pardata0_2799   ASM    ACTIVE
```

```
# /etc/vx/bin/vxisforeign 3pardata0_2798
 3pardata0_2798   ASM    FORMER
```

Chapter 3

How Veritas Volume Manager works

This chapter includes the following topics:

How Veritas Volume Manager works with the operating system

Veritas Volume Manager (VxVM) operates as a subsystem between your operating system and your data management systems, such as file systems and database management systems. VxVM is tightly coupled with the operating system. Before a disk or LUN can be brought under VxVM control, the disk must be accessible through the operating system device interface. VxVM is layered on top of the operating system interface services, and is dependent upon how the operating system accesses physical disks.

VxVM is dependent upon the operating system for the following functionality:

- operating system (disk) devices
- device handles
- VxVM Dynamic Multi-Pathing (DMP) metadevice

VxVM relies on the following constantly-running daemons and kernel threads for its operation:

`vxconfigd`

The VxVM configuration daemon maintains disk and group configurations and communicates configuration changes to the kernel, and modifies configuration information stored on disks.

See the `vxconfigd`(1m) manual page.

`vxiod`

VxVM I/O kernel threads provide extended I/O operations without blocking calling processes. By default, 16 I/O threads are started at boot time, and at least one I/O thread must continue to run at all times.

See the `vxiod`(1m) manual page.

`vxrelocd`

The hot-relocation daemon monitors VxVM for events that affect redundancy, and performs hot-relocation to restore redundancy. If thin provision disks are configured in the system, then the storage space of a deleted volume is reclaimed by this daemon as configured by the policy.

See the `vxrelocd`(1m) manual page.

How data is stored

Several methods are used to store data on physical disks. These methods organize data on the disk so the data can be stored and retrieved efficiently. The basic method of disk organization is called formatting. Formatting prepares the hard disk so that files can be written to and retrieved from the disk by using a prearranged storage pattern.

Two methods are used to store information on formatted hard disks: physical-storage layout and logical-storage layout. VxVM uses the logical-storage layout method.

See “How Veritas Volume Manager handles storage management” on page 48.

How Veritas Volume Manager handles storage management

Veritas Volume Manager (VxVM) uses the following types of objects to handle storage management:

Physical objects	Physical disks, LUNs (virtual disks implemented in hardware), or other hardware with block and raw operating system device interfaces that are used to store data. See “Physical objects” on page 48.
Virtual objects	When one or more physical disks are brought under the control of VxVM, it creates virtual objects called volumes on those physical disks. Each volume records and retrieves data from one or more physical disks. Volumes are accessed by file systems, databases, or other applications in the same way that physical disks are accessed. Volumes are also composed of other virtual objects (plexes and subdisks) that are used in changing the volume configuration. Volumes and their virtual components are called virtual objects or VxVM objects. See “Virtual objects” on page 50.

Physical objects

A physical disk is the basic storage device (media) where the data is ultimately stored. You can access the data on a physical disk by using a device name to locate the disk. The physical disk device name varies with the computer system you use. Not all parameters are used on all systems.

Typical device names are of the form `sda` or `hdb`, where `sda` references the first (`a`) SCSI disk, and `hdb` references the second (`b`) EIDE disk.

Figure 3-1 shows how a physical disk and device name (*devname*) are illustrated in the Veritas Volume Manager (VxVM) documentation.

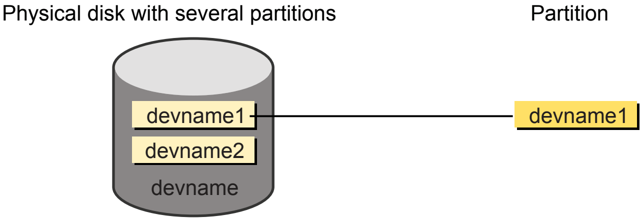
Figure 3-1 Physical disk example

VxVM writes identification information on physical disks under VxVM control (VM disks). VxVM disks can be identified even after physical disk disconnection or system outages. VxVM can then re-form disk groups and logical objects to provide failure detection and to speed system recovery.

About disk partitions

Figure 3-2 shows how a physical disk can be divided into one or more partitions.

Figure 3-2 Partition example



The partition number is added at the end of the *devname*.

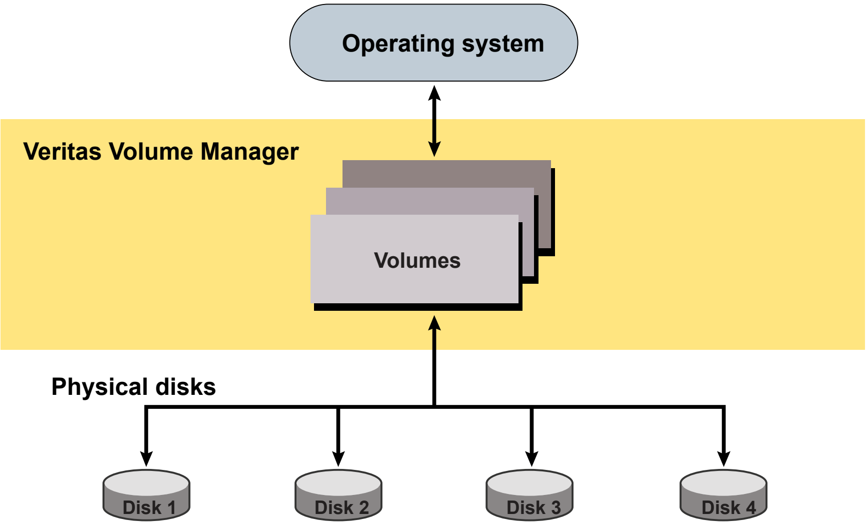
Disk arrays

Performing I/O to disks is a relatively slow process because disks are physical devices that require time to move the heads to the correct position on the disk before reading or writing. If all of the read or write operations are done to individual disks, one at a time, the read-write time can become unmanageable. Performing these operations on multiple disks can help to reduce this problem.

A disk array is a collection of physical disks that VxVM can represent to the operating system as one or more virtual disks or volumes. The volumes created by VxVM look and act to the operating system like physical disks. Applications that interact with volumes should work in the same way as with physical disks.

Figure 3-3 shows how VxVM represents the disks in a disk array as several volumes to the operating system.

Figure 3-3 How VxVM presents the disks in a disk array as volumes to the operating system



Data can be spread across several disks within an array, or across disks spanning multiple arrays, to distribute or balance I/O operations across the disks. Using parallel I/O across multiple disks in this way improves I/O performance by increasing data transfer speed and overall throughput for the array.

Virtual objects

Veritas Volume Manager (VxVM) uses multiple virtualization layers to provide distinct functionality and reduce physical limitations. The connection between physical objects and VxVM objects is made when you place a physical disk under VxVM control.

Table 3-1 describes the virtual objects in VxVM.

Table 3-1 VxVM virtual objects

Virtual object	Description
Disk groups	A disk group is a collection of disks that share a common configuration and which are managed by VxVM. A disk group configuration is a set of records with detailed information about related VxVM objects, their attributes, and their connections. A disk group name can be up to 29 characters long. Disk group names must not contain periods (.).

Table 3-1 VxVM virtual objects *(continued)*

Virtual object	Description
VxVM disks	A VxVM disk is assigned to a physical disk, when you place the physical disk under VxVM control. A VxVM disk is usually in a disk group. VxVM allocates storage from a contiguous area of VxVM disk space. Each VxVM disk corresponds to at least one physical disk or disk partition. A VxVM disk typically includes a public region (allocated storage) and a small private region where VxVM internal configuration information is stored.
Subdisks	A subdisk is a set of contiguous disk blocks. A block is a unit of space on the disk. VxVM allocates disk space using subdisks. A VxVM disk can be divided into one or more subdisks. Each subdisk represents a specific portion of a VxVM disk, which is mapped to a specific region of a physical disk.
Plexes	A plex consists of one or more subdisks located on one or more physical disks.
Volumes	A volume is a virtual disk device that appears to applications, databases, and file systems like a physical disk device, but does not have the physical limitations of a physical disk device. A volume consists of one or more plexes, each holding a copy of the selected data in the volume. Due to its virtual nature, a volume is not restricted to a particular disk or a specific area of a disk. The configuration of a volume can be changed by using VxVM user interfaces. Configuration changes can be accomplished without causing disruption to applications or file systems that are using the volume. For example, a volume can be mirrored on separate disks or moved to use different disk storage.

After installing VxVM on a host system, you must bring the contents of physical disks under VxVM control by collecting the VxVM disks into disk groups and allocating the disk group space to create logical volumes.

Bringing the contents of physical disks under VxVM control is accomplished only if VxVM takes control of the physical disks and the disk is not under control of another storage manager such as LVM.

For more information on how LVM and VxVM disks co-exist or how to convert LVM disks to VxVM disks, see the *Veritas InfoScale Solutions Guide*.

VxVM creates virtual objects and makes logical connections between the objects. The virtual objects are then used by VxVM to do storage management tasks.

The `vxprint` command displays detailed information about the VxVM objects that exist on a system.

See the `vxprint`(1M) manual page.

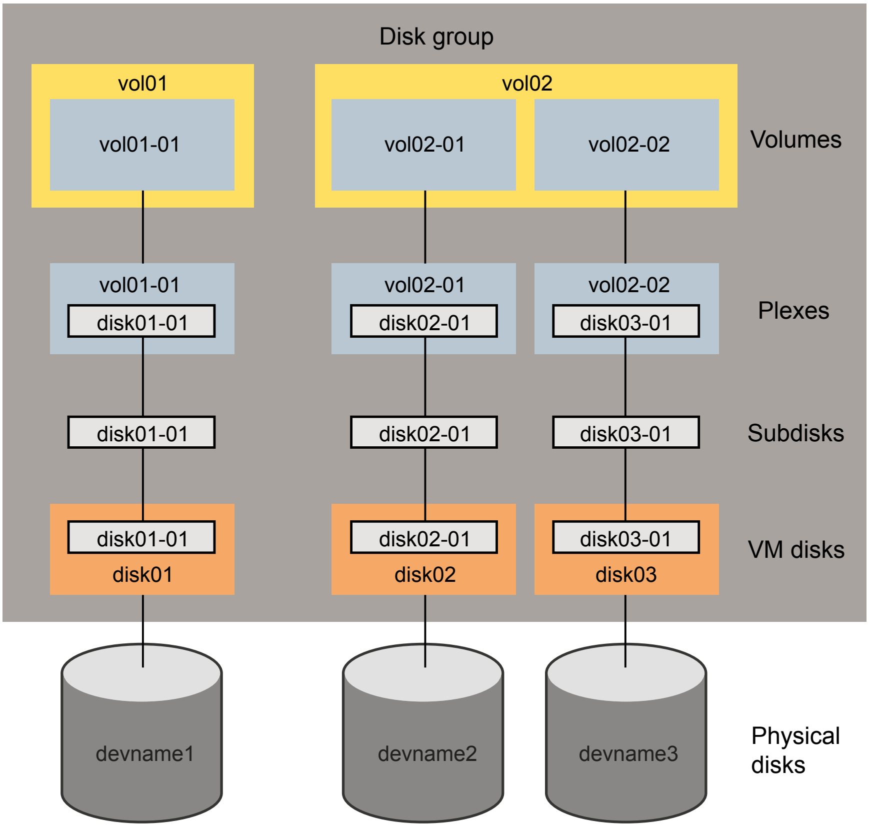
Combining virtual objects in Veritas Volume Manager

Veritas Volume Manager (VxVM) virtual objects are combined to build volumes. The virtual objects contained in volumes are VxVM disks, disk groups, subdisks, and plexes. VxVM virtual objects are organized in the following ways:

- VxVM disks are grouped into disk groups
- Subdisks (each representing a specific region of a disk) are combined to form plexes
- Volumes are composed of one or more plexes

Figure 3-4 shows the connections between VxVM virtual objects and how they relate to physical disks.

Figure 3-4 Connection between objects in VxVM



The disk group contains three VxVM disks which are used to create two volumes. Volume `vol01` is simple and has a single plex. Volume `vol02` is a mirrored volume with two plexes.

The various types of virtual objects (disk groups, VM disks, subdisks, plexes, and volumes) are described in the following sections. Other types of objects exist in Veritas Volume Manager, such as data change objects (DCOs), and volume sets, to provide extended functionality.

About the configuration daemon in Veritas Volume Manager

The Veritas Volume Manager (VxVM) configuration daemon (`vxconfigd`) provides the interface between VxVM commands and the kernel device drivers. `vxconfigd` handles configuration change requests from VxVM utilities, communicates the change requests to the VxVM kernel, and modifies configuration information stored on disk. vxconfigd also initializes VxVM when the system is booted.

The vxdctl command is the command-line interface to the vxconfigd daemon.

You can use vxdctl to:

- Control the operation of the vxconfigd daemon.
- Change the system-wide definition of the default disk group.

In VxVM 4.0 and later releases, disk access records are no longer stored in the /etc/vx/volboot file. Non-persistent disk access records are created by scanning the disks at system startup. Persistent disk access records for simple and nopriv disks are permanently stored in the /etc/vx/darecs file in the root file system. The vxconfigd daemon reads the contents of this file to locate the disks and the configuration databases for their disk groups.

The /etc/vx/darecs file is also used to store definitions of foreign devices that are not autoconfigurable. Such entries may be added by using the vxddladm addforeign command.

See the vxddladm(1M) manual page.

If your system is configured to use Dynamic Multi-Pathing (DMP), you can also use vxdctl to:

- Reconfigure the DMP database to include disk devices newly attached to, or removed from the system.
- Create DMP device nodes in the /dev/vx/dmp and /dev/vx/rdmp directories.
- Update the DMP database with changes in path type for active/passive disk arrays. Use the utilities provided by the disk-array vendor to change the path type between primary and secondary.

See the vxdctl(1M) manual page.

Multiple paths to disk arrays

Some disk arrays provide multiple ports to access their disk devices. These ports, coupled with the host bus adaptor (HBA) controller and any data bus or I/O processor local to the array, make up multiple hardware paths to access the disk devices. Such disk arrays are called multipathed disk arrays. This type of disk array can be connected to host systems in many different configurations, (such as multiple ports connected to different controllers on a single host, chaining of the ports through a single controller on a host, or ports connected to different hosts simultaneously).

See “How DMP works” on page 34.

Volume layouts in Veritas Volume Manager

A Veritas Volume Manager (VxVM) virtual device is defined by a volume. A volume has a layout defined by the association of a volume to one or more plexes, each of which map to one or more subdisks. The volume presents a virtual device interface that is exposed to other applications for data access. These logical building blocks re-map the volume address space through which I/O is re-directed at run-time.

Different volume layouts provide different levels of availability and performance. A volume layout can be configured and changed to provide the desired level of service.

Non-layered volumes

In a non-layered volume, a subdisk maps directly to a VxVM disk. This allows the subdisk to define a contiguous extent of storage space backed by the public region of a VxVM disk. When active, the VxVM disk is directly associated with an underlying physical disk. The combination of a volume layout and the physical disks therefore determines the storage service available from a given virtual device.

Layered volumes

A layered volume is constructed by mapping its subdisks to underlying volumes. The subdisks in the underlying volumes must map to VxVM disks, and hence to attached physical storage.

Layered volumes allow for more combinations of logical compositions, some of which may be desirable for configuring a virtual device. For example, layered volumes allow for high availability when using striping. Because permitting free use of layered volumes throughout the command level would have resulted in unwieldy administration, some ready-made layered volume configurations are designed into VxVM.

See “About layered volumes” on page 69.

These ready-made configurations operate with built-in rules to automatically match desired levels of service within specified constraints. The automatic configuration is done on a “best-effort” basis for the current command invocation working against the current configuration.

To achieve the desired storage service from a set of virtual devices, it may be necessary to include an appropriate set of VxVM disks into a disk group and to execute multiple configuration commands.

To the extent that it can, VxVM handles initial configuration and on-line re-configuration with its set of layouts and administration interface to make this job easier and more deterministic.

Layout methods

Data in virtual objects is organized to create volumes by using the following layout methods:

- Concatenation, spanning, and carving
  See “Concatenation, spanning, and carving” on page 56.
- Striping (RAID-0)
  See “Striping (RAID-0)” on page 58.
- Mirroring (RAID-1)
  See “Mirroring (RAID-1)” on page 61.
- Striping plus mirroring (mirrored-stripe or RAID-0+1)
  See “Striping plus mirroring (mirrored-stripe or RAID-0+1)” on page 62.
- Mirroring plus striping (striped-mirror, RAID-1+0 or RAID-10)
  See “Mirroring plus striping (striped-mirror, RAID-1+0, or RAID-10)” on page 63.
- RAID-5 (striping with parity)
  See “RAID-5 (striping with parity)” on page 64.

Concatenation, spanning, and carving

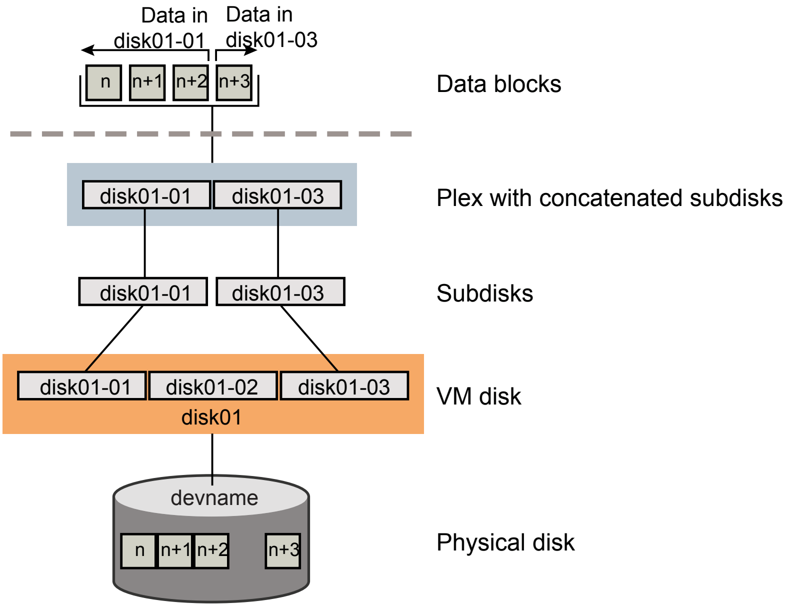
Concatenation maps data in a linear manner onto one or more subdisks in a plex. To access all of the data in a concatenated plex sequentially, data is first accessed in the first subdisk from the beginning to the end. Data is then accessed in the remaining subdisks sequentially from the beginning to the end of each subdisk, until the end of the last subdisk.

The subdisks in a concatenated plex do not have to be physically contiguous and can belong to more than one VxVM disk. Concatenation using subdisks that reside on more than one VxVM disk is called spanning.

Figure 3-5 shows the concatenation of two subdisks from the same VxVM disk.

If a single LUN or disk is split into multiple subdisks, and each subdisk belongs to a unique volume, it is called carving.

Figure 3-5 Example of concatenation



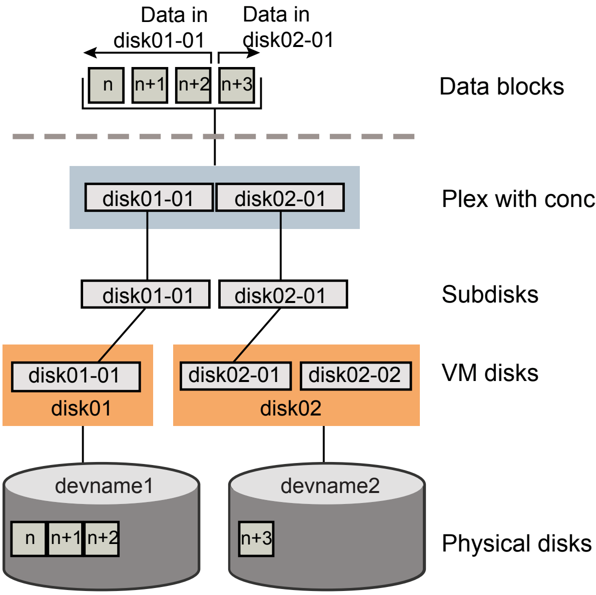
The blocks `n`, `n+1`, `n+2` and `n+3` (numbered relative to the start of the plex) are contiguous on the plex, but actually come from two distinct subdisks on the same physical disk.

The remaining free space in the subdisk `disk01-02` on VxVM disk `disk01` can be put to other uses.

You can use concatenation with multiple subdisks when there is insufficient contiguous space for the plex on any one disk. This form of concatenation can be used for load balancing between disks, and for head movement optimization on a particular disk.

Figure 3-6 shows data spread over two subdisks in a spanned plex.

Figure 3-6 Example of spanning



The blocks `n`, `n+1`, `n+2` and `n+3` (numbered relative to the start of the plex) are contiguous on the plex, but actually come from two distinct subdisks from two distinct physical disks.

The remaining free space in the subdisk `disk02-02` on VxVM disk `disk02` can be put to other uses.

Warning: Spanning a plex across multiple disks increases the chance that a disk failure results in failure of the assigned volume. Use mirroring or RAID-5 to reduce the risk that a single disk failure results in a volume failure.

Striping (RAID-0)

Striping (RAID-0) is useful if you need large amounts of data written to or read from physical disks, and performance is important. Striping is also helpful in balancing the I/O load from multi-user applications across multiple disks. By using parallel data transfer to and from multiple disks, striping significantly improves data-access performance.

Striping maps data so that the data is interleaved among two or more physical disks. A striped plex contains two or more subdisks, spread out over two or more physical disks. Data is allocated alternately and evenly to the subdisks of a striped plex.

The subdisks are grouped into “columns,” with each physical disk limited to one column. Each column contains one or more subdisks and can be derived from one or more physical disks. The number and sizes of subdisks per column can vary. Additional subdisks can be added to columns, as necessary.

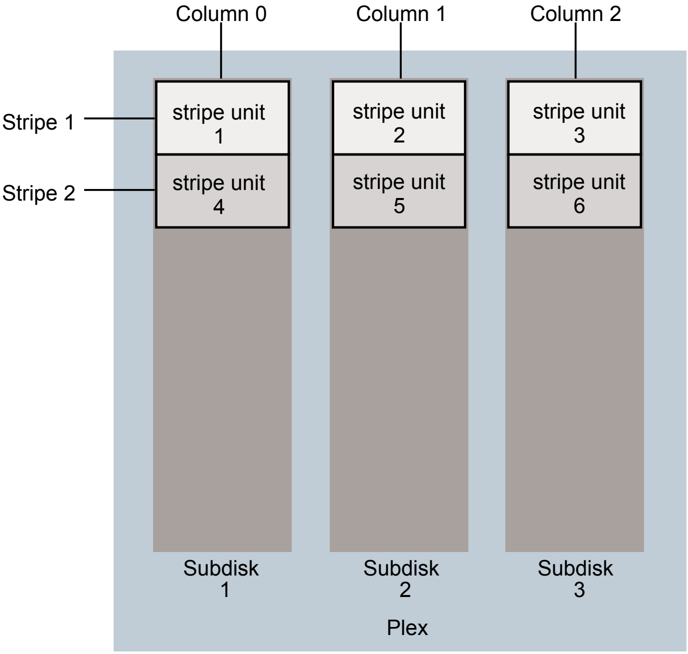
Warning: Striping a volume, or splitting a volume across multiple disks, increases the chance that a disk failure will result in failure of that volume.

If five volumes are striped across the same five disks, then failure of any one of the five disks will require that all five volumes be restored from a backup. If each volume is on a separate disk, only one volume has to be restored. (As an alternative to or in conjunction with striping, use mirroring or RAID-5 to substantially reduce the chance that a single disk failure results in failure of a large number of volumes.)

Data is allocated in equal-sized stripe units that are interleaved between the columns. Each stripe unit is a set of contiguous blocks on a disk. The default stripe unit size is 64 kilobytes.

Figure 3-7 shows an example with three columns in a striped plex, six stripe units, and data striped over the three columns.

Figure 3-7 Striping across three columns



A stripe consists of the set of stripe units at the same positions across all columns. In the figure, stripe units 1, 2, and 3 constitute a single stripe.

Viewed in sequence, the first stripe consists of:

- stripe unit 1 in column 0
- stripe unit 2 in column 1
- stripe unit 3 in column 2

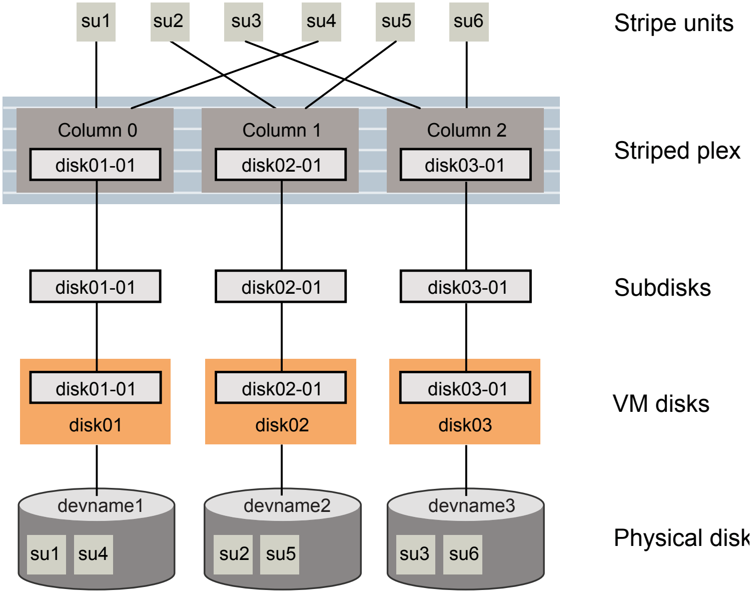
The second stripe consists of:

- stripe unit 4 in column 0
- stripe unit 5 in column 1
- stripe unit 6 in column 2

Striping continues for the length of the columns (if all columns are the same length), or until the end of the shortest column is reached. Any space remaining at the end of subdisks in longer columns becomes unused space.

Figure 3-8 shows a striped plex with three equal sized, single-subdisk columns.

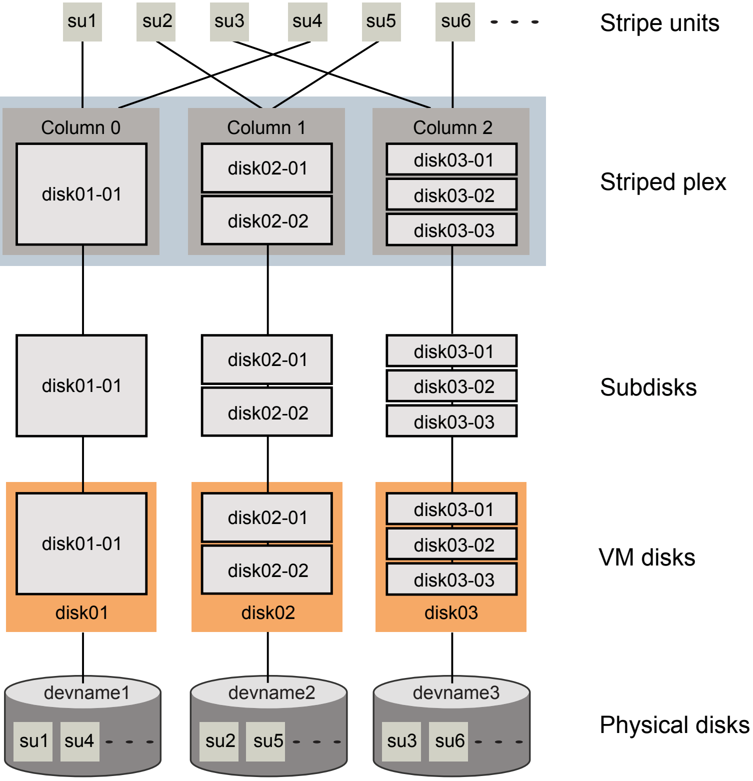
Figure 3-8 Example of a striped plex with one subdisk per column



There is one column per physical disk. This example shows three subdisks that occupy all of the space on the VM disks. It is also possible for each subdisk in a striped plex to occupy only a portion of the VM disk, which leaves free space for other disk management tasks.

Figure 3-9 shows a striped plex with three columns containing subdisks of different sizes.

Figure 3-9 Example of a striped plex with concatenated subdisks per column



Each column contains a different number of subdisks. There is one column per physical disk. Striped plexes can be created by using a single subdisk from each of the VM disks being striped across. It is also possible to allocate space from different regions of the same disk or from another disk (for example, if the size of the plex is increased). Columns can also contain subdisks from different VM disks.

See "Creating a striped volume" on page 147.

Mirroring (RAID-1)

Mirroring uses multiple mirrors (plexes) to duplicate the information contained in a volume. In the event of a physical disk failure, the plex on the failed disk becomes unavailable, but the system continues to operate using the unaffected mirrors. Similarly, mirroring two LUNs from two separate controllers lets the system operate if there is a controller failure.

Although a volume can have a single plex, at least two plexes are required to provide redundancy of data. Each of these plexes must contain disk space from different disks to achieve redundancy.

When striping or spanning across a large number of disks, failure of any one of those disks can make the entire plex unusable. Because the likelihood of one out of several disks failing is reasonably high, you should consider mirroring to improve the reliability (and availability) of a striped or spanned volume.

See "Creating a mirrored volume" on page 145.

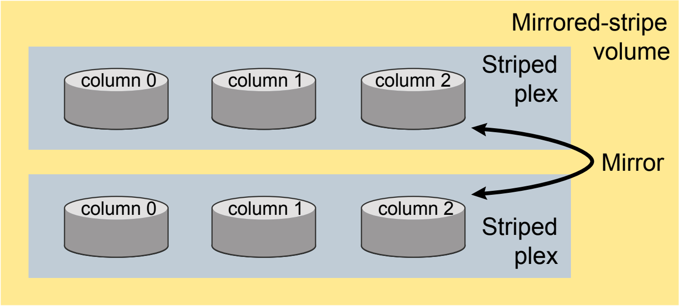
Striping plus mirroring (mirrored-stripe or RAID-0+1)

VxVM supports the combination of mirroring above striping. The combined layout is called a mirrored-stripe layout. A mirrored-stripe layout offers the dual benefits of striping to spread data across multiple disks, while mirroring provides redundancy of data.

For mirroring above striping to be effective, the striped plex and its mirrors must be allocated from separate disks.

Figure 3-10 shows an example where two plexes, each striped across three disks, are attached as mirrors to the same volume to create a mirrored-stripe volume.

Figure 3-10 Mirrored-stripe volume laid out on six disks



See "Creating a mirrored-stripe volume" on page 148.

The layout type of the data plexes in a mirror can be concatenated or striped. Even if only one is striped, the volume is still termed a mirrored-stripe volume. If they are all concatenated, the volume is termed a mirrored-concatenated volume.

Mirroring plus striping (striped-mirror, RAID-1+0, or RAID-10)

Veritas Volume Manager (VxVM) supports the combination of striping above mirroring. This combined layout is called a striped-mirror layout. Putting mirroring below striping mirrors each column of the stripe. If there are multiple subdisks per column, each subdisk can be mirrored individually instead of each column.

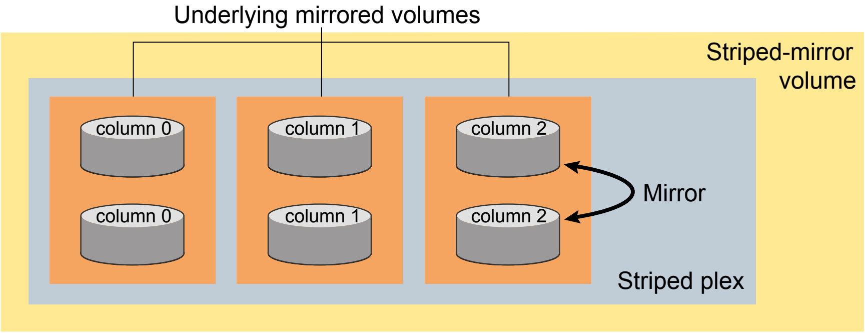
A striped-mirror volume is an example of a layered volume.

See "About layered volumes" on page 69.

As for a mirrored-stripe volume, a striped-mirror volume offers the dual benefits of striping to spread data across multiple disks, while mirroring provides redundancy of data. In addition, it enhances redundancy, and reduces recovery time after disk failure.

Figure 3-11 shows an example where a striped-mirror volume is created by using each of three existing 2-disk mirrored volumes to form a separate column within a striped plex.

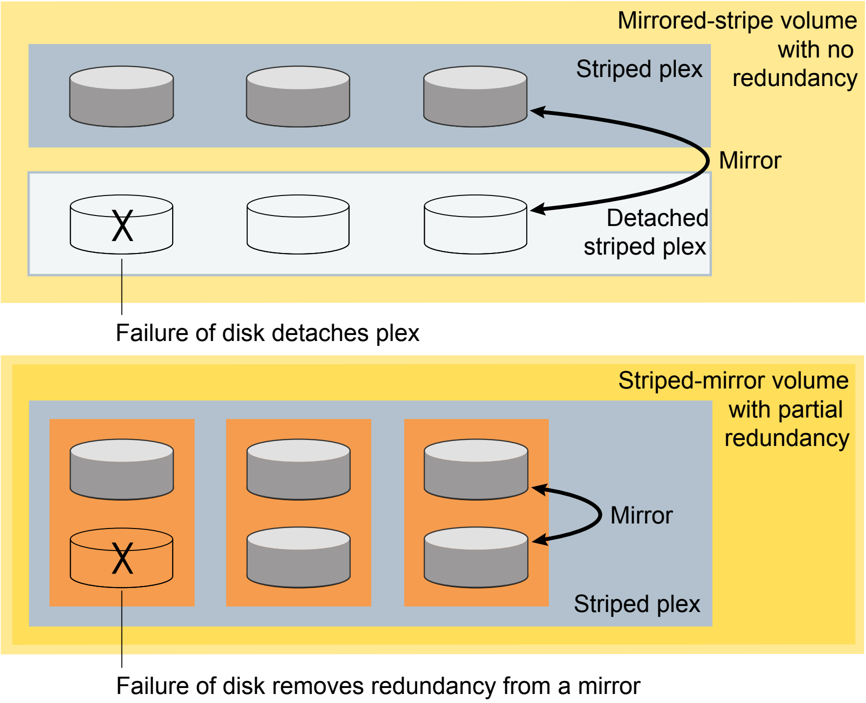
Figure 3-11 Striped-mirror volume laid out on six disks



See "Creating a striped-mirror volume" on page 148.

Figure 3-12 shows that the failure of a disk in a mirrored-stripe layout detaches an entire data plex, thereby losing redundancy on the entire volume.

Figure 3-12 How the failure of a single disk affects mirrored-stripe and striped-mirror volumes


Mirrored-stripe volume with no redundancy
Striped plex
Mirror
X
Detached striped plex
Failure of disk detaches plex
Striped-mirror volume with partial redundancy
X
Mirror
Striped plex
Failure of disk removes redundancy from a mirror

When the disk is replaced, the entire plex must be brought up to date. Recovering the entire plex can take a substantial amount of time. If a disk fails in a striped-mirror layout, only the failing subdisk must be detached, and only that portion of the volume loses redundancy. When the disk is replaced, only a portion of the volume needs to be recovered. Additionally, a mirrored-stripe volume is more vulnerable to being put out of use altogether should a second disk fail before the first failed disk has been replaced, either manually or by hot-relocation.

Compared to mirrored-stripe volumes, striped-mirror volumes are more tolerant of disk failure, and recovery time is shorter.

If the layered volume concatenates instead of striping the underlying mirrored volumes, the volume is termed a concatenated-mirror volume.

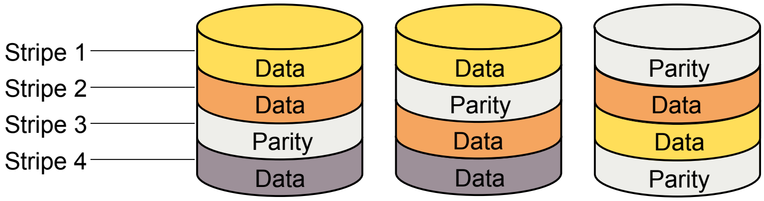
RAID-5 (striping with parity)

Although both mirroring (RAID-1) and RAID-5 provide redundancy of data, they use different methods. Mirroring provides data redundancy by maintaining multiple complete copies of the data in a volume. Data being written to a mirrored volume is reflected in all copies. If a portion of a mirrored volume fails, the system continues to use the other copies of the data.

RAID-5 provides data redundancy by using parity. Parity is a calculated value used to reconstruct data after a failure. While data is being written to a RAID-5 volume, parity is calculated by doing an exclusive OR (XOR) procedure on the data. The resulting parity is then written to the volume. The data and calculated parity are contained in a plex that is “striped” across multiple disks. If a portion of a RAID-5 volume fails, the data that was on that portion of the failed volume can be recreated from the remaining data and parity information. It is also possible to mix concatenation and striping in the layout.

Figure 3-13 shows parity locations in a RAID-5 array configuration.

Figure 3-13 Parity locations in a RAID-5 model



Every stripe has a column containing a parity stripe unit and columns containing data. The parity is spread over all of the disks in the array, reducing the write time for large independent writes because the writes do not have to wait until a single parity disk can accept the data.

RAID-5 volumes can additionally perform logging to minimize recovery time. RAID-5 volumes use RAID-5 logs to keep a copy of the data and parity currently being written. RAID-5 logging is optional and can be created along with RAID-5 volumes or added later.

See “Veritas Volume Manager RAID-5 arrays” on page 66.

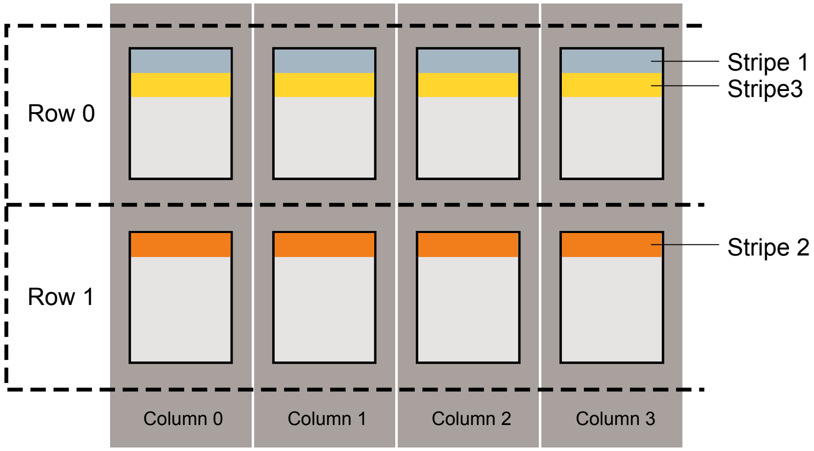
Note: Veritas Volume Manager (VxVM) supports RAID-5 for private disk groups, but not for shareable disk groups in a Cluster Volume Manager (CVM) environment. In addition, VxVM does not support the mirroring of RAID-5 volumes that are configured using VxVM software. RAID-5 LUNs hardware may be mirrored.

Traditional RAID-5 arrays

A traditional RAID-5 array is several disks organized in rows and columns. A column is a number of disks located in the same ordinal position in the array. A row is the minimal number of disks necessary to support the full width of a parity stripe.

Figure 3-14 shows the row and column arrangement of a traditional RAID-5 array.

Figure 3-14 Traditional RAID-5 array



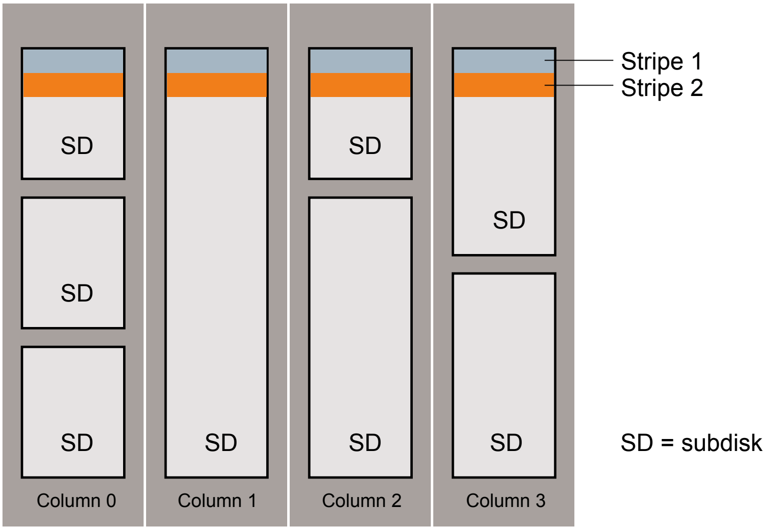
This traditional array structure supports growth by adding more rows per column. Striping is accomplished by applying the first stripe across the disks in Row 0, then the second stripe across the disks in Row 1, then the third stripe across the Row 0 disks, and so on. This type of array requires all disks columns and rows to be of equal size.

Veritas Volume Manager RAID-5 arrays

The RAID-5 array structure in Veritas Volume Manager (VxVM) differs from the traditional structure. Due to the virtual nature of its disks and other objects, VxVM does not use rows.

Figure 3-15 shows how VxVM uses columns consisting of variable length subdisks, where each subdisk represents a specific area of a disk.

Figure 3-15 Veritas Volume Manager RAID-5 array



VxVM allows each column of a RAID-5 plex to consist of a different number of subdisks. The subdisks in a given column can be derived from different physical disks. Additional subdisks can be added to the columns as necessary. Striping is implemented by applying the first stripe across each subdisk at the top of each column, then applying another stripe below that, and so on for the length of the columns. Equal-sized stripe units are used for each column. For RAID-5, the default stripe unit size is 16 kilobytes.

See “Striping (RAID-0)” on page 58.

Note: Mirroring of RAID-5 volumes is not supported.

See “Creating a RAID-5 volume” on page 149.

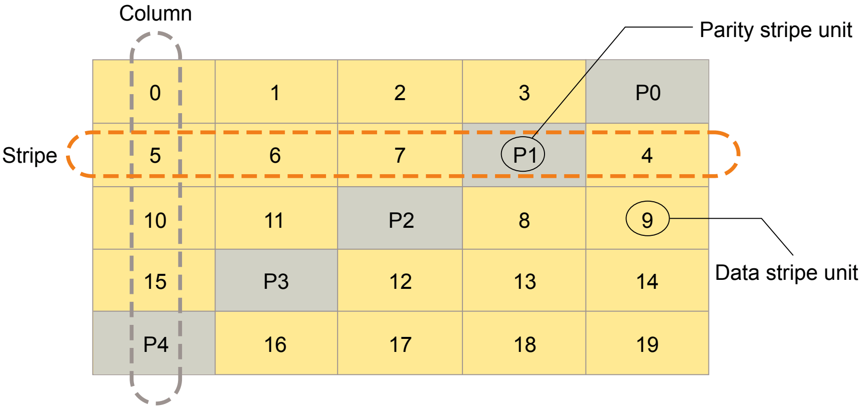
Left-symmetric layout

There are several layouts for data and parity that can be used in the setup of a RAID-5 array. The implementation of RAID-5 in VxVM uses a left-symmetric layout. This provides optimal performance for both random I/O operations and large sequential I/O operations. However, the layout selection is not as critical for performance as are the number of columns and the stripe unit size.

Left-symmetric layout stripes both data and parity across columns, placing the parity in a different column for every stripe of data. The first parity stripe unit is located in the rightmost column of the first stripe. Each successive parity stripe unit is located in the next stripe, shifted left one column from the previous parity stripe unit location. If there are more stripes than columns, the parity stripe unit placement begins in the rightmost column again.

Figure 3-16 shows a left-symmetric parity layout with five disks (one per column).

Figure 3-16 Left-symmetric layout



0	1	2	3	P0
5	6	7	P1	4
10	11	P2	8	9
15	P3	12	13	14
P4	16	17	18	19

For each stripe, data is organized starting to the right of the parity stripe unit. In the figure, data organization for the first stripe begins at P0 and continues to stripe units 0-3. Data organization for the second stripe begins at P1, then continues to stripe unit 4, and on to stripe units 5-7. Data organization proceeds in this manner for the remaining stripes.

Each parity stripe unit contains the result of an exclusive OR (XOR) operation performed on the data in the data stripe units within the same stripe. If one column's data is inaccessible due to hardware or software failure, the data for each stripe can be restored by XORing the contents of the remaining columns data stripe units against their respective parity stripe units.

For example, if a disk corresponding to the whole or part of the far left column fails, the volume is placed in a degraded mode. While in degraded mode, the data from the failed column can be recreated by XORing stripe units 1-3 against parity stripe unit P0 to recreate stripe unit 0, then XORing stripe units 4, 6, and 7 against parity stripe unit P1 to recreate stripe unit 5, and so on.

Failure of more than one column in a RAID-5 plex detaches the volume. The volume is no longer allowed to satisfy read or write requests. Once the failed columns have been recovered, it may be necessary to recover user data from backups.

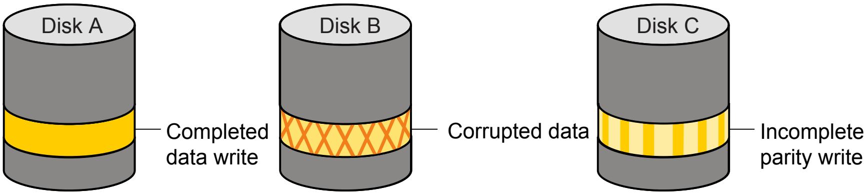
RAID-5 logging

Logging is used to prevent corruption of data during recovery by immediately recording changes to data and parity to a log area on a persistent device such as a volume on disk or in non-volatile RAM. The new data and parity are then written to the disks.

Without logging, it is possible for data not involved in any active writes to be lost or silently corrupted if both a disk in a RAID-5 volume and the system fail. If this double-failure occurs, there is no way of knowing if the data being written to the data portions of the disks or the parity being written to the parity portions have actually been written. Therefore, the recovery of the corrupted disk may be corrupted itself.

Figure 3-17 shows a RAID-5 volume configured across three disks (A, B, and C).

Figure 3-17 Incomplete write to a RAID-5 volume



In this volume, recovery of disk B's corrupted data depends on disk A's data and disk C's parity both being complete. However, only the data write to disk A is complete. The parity write to disk C is incomplete, which would cause the data on disk B to be reconstructed incorrectly.

This failure can be avoided by logging all data and parity writes before committing them to the array. In this way, the log can be replayed, causing the data and parity updates to be completed before the reconstruction of the failed drive takes place.

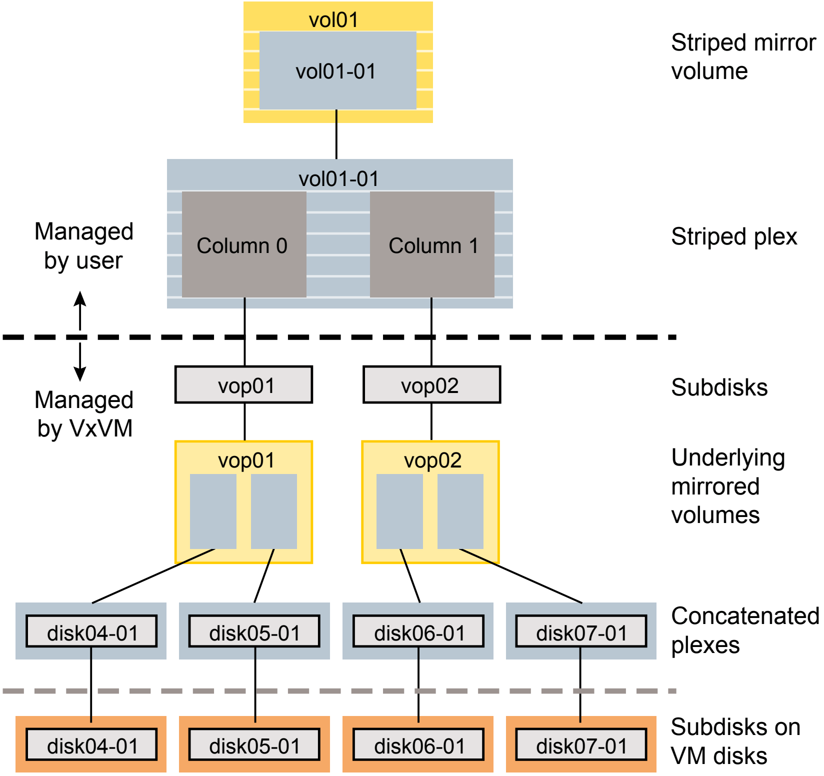
Logs are associated with a RAID-5 volume by being attached as log plexes. More than one log plex can exist for each RAID-5 volume, in which case the log areas are mirrored.

About layered volumes

A layered volume is a virtual Veritas Volume Manager (VxVM) object that is built on top of other volumes. The layered volume structure tolerates failure better and has greater redundancy than the standard volume structure. For example, in a striped-mirror layered volume, each mirror (plex) covers a smaller area of storage space, so recovery is quicker than with a standard mirrored volume.

Figure 3-18 shows a typical striped-mirror layered volume where each column is represented by a subdisk that is built from an underlying mirrored volume.

Figure 3-18 Example of a striped-mirror layered volume



The volume and striped plex in the "Managed by user" area allow you to perform normal tasks in VxVM. User tasks can be performed only on the top-level volume of a layered volume.

Underlying volumes in the "Managed by VxVM" area are used exclusively by VxVM and are not designed for user manipulation. You cannot detach a layered volume or perform any other operation on the underlying volumes by manipulating the internal structure. You can perform all necessary operations in the "Managed by user" area that includes the top-level volume and striped plex (for example, resizing the volume, changing the column width, or adding a column).

System administrators can manipulate the layered volume structure for troubleshooting or other operations (for example, to place data on specific disks). Layered volumes are used by VxVM to perform the following tasks and operations:

Creating striped-mirrors	See "Creating a striped-mirror volume" on page 148. See the vxassist(1M) manual page.
Creating concatenated-mirrors	See "Creating a concatenated-mirror volume" on page 147. See the vxassist(1M) manual page.
Online Relayout	See "Online relayout" on page 71. See the vxassist(1M) manual page. See the vxrelayout(1M) manual page.
Moving RAID-5 subdisks	See the vxsd(1M) manual page.
Creating Snapshots	See "Volume snapshots" on page 78. See the vxassist(1M) manual page. See the vxsnap(1M) manual page.

Online relayout

Online relayout allows you to convert between storage layouts in VxVM, with uninterrupted data access. Typically, you would do this to change the redundancy or performance characteristics of a volume. VxVM adds redundancy to storage either by duplicating the data (mirroring) or by adding parity (RAID-5). Performance characteristics of storage in VxVM can be changed by changing the striping parameters, which are the number of columns and the stripe width.

See "Performing online relayout" on page 660.

How online relayout works

Online relayout allows you to change the storage layouts that you have already created in place without disturbing data access. You can change the performance characteristics of a particular layout to suit your changed requirements. You can transform one layout to another by invoking a single command.

For example, if a striped layout with a 128KB stripe unit size is not providing optimal performance, you can use relayout to change the stripe unit size.

File systems mounted on the volumes do not need to be unmounted to achieve this transformation provided that the file system (such as Veritas File System) supports online shrink and grow operations.

Online relayout reuses the existing storage space and has space allocation policies to address the needs of the new layout. The layout transformation process converts a given volume to the destination layout by using minimal temporary space that is available in the disk group.

The transformation is done by moving one portion of data at a time in the source layout to the destination layout. Data is copied from the source volume to the temporary area, and data is removed from the source volume storage area in portions. The source volume storage area is then transformed to the new layout, and the data saved in the temporary area is written back to the new layout. This operation is repeated until all the storage and data in the source volume has been transformed to the new layout.

The default size of the temporary area used during the relayout depends on the size of the volume and the type of relayout. For volumes larger than 50MB, the amount of temporary space that is required is usually 10% of the size of the volume, from a minimum of 50MB up to a maximum of 1GB. For volumes smaller than 50MB, the temporary space required is the same as the size of the volume.

The following error message displays the number of blocks required if there is insufficient free space available in the disk group for the temporary area:

```
tmpsize too small to perform this relayout (nblks minimum required)
```

You can override the default size used for the temporary area by using the `tmpsize` attribute to `vxassist`.

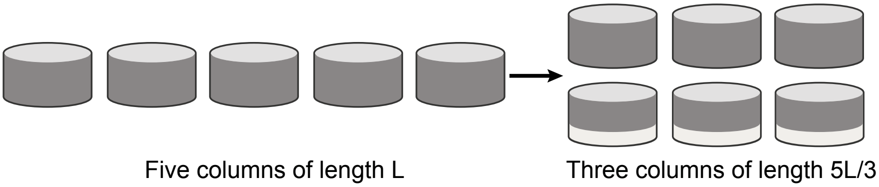
See the `vxassist`(1M) manual page.

As well as the temporary area, space is required for a temporary intermediate volume when increasing the column length of a striped volume. The amount of space required is the difference between the column lengths of the target and source volumes. For example, 20GB of temporary additional space is required to relayout a 150GB striped volume with 5 columns of length 30GB as 3 columns of length 50GB. In some cases, the amount of temporary space that is required is relatively large. For example, a relayout of a 150GB striped volume with 5 columns as a concatenated volume (with effectively one column) requires 120GB of space for the intermediate volume.

Additional permanent disk space may be required for the destination volumes, depending on the type of relayout that you are performing. This may happen, for example, if you change the number of columns in a striped volume.

Figure 3-19 shows how decreasing the number of columns can require disks to be added to a volume.

Figure 3-19 Example of decreasing the number of columns in a volume



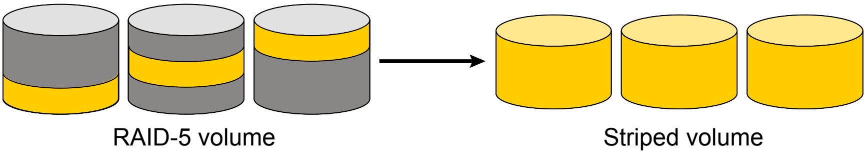
Note that the size of the volume remains the same but an extra disk is needed to extend one of the columns.

The following are examples of operations that you can perform using online relayout:

- Remove parity from a RAID-5 volume to change it to a concatenated, striped, or layered volume.

  Figure 3-20 shows an example of applying relayout a RAID-5 volume.

Figure 3-20 Example of relayout of a RAID-5 volume to a striped volume

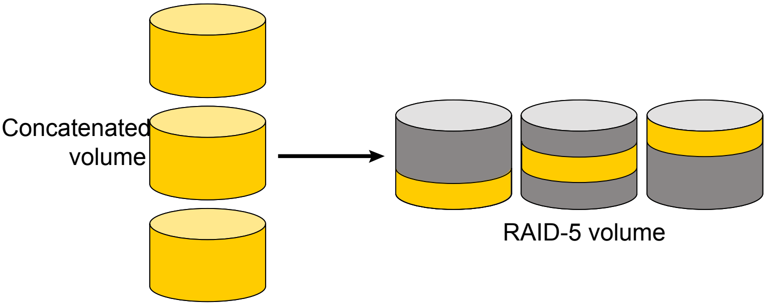


Note that removing parity decreases the overall storage space that the volume requires.

- Add parity to a volume to change it to a RAID-5 volume.

  Figure 3-21 shows an example.

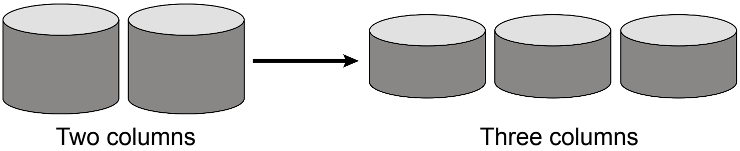
Figure 3-21 Example of relayout of a concatenated volume to a RAID-5 volume



Note that adding parity increases the overall storage space that the volume requires.

- Change the number of columns in a volume.
  Figure 3-22 shows an example of changing the number of columns.

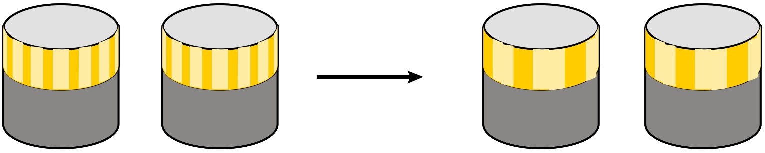
Figure 3-22 Example of increasing the number of columns in a volume



Note that the length of the columns is reduced to conserve the size of the volume.

- Change the column stripe width in a volume.
  Figure 3-23 shows an example of changing the column stripe width.

Figure 3-23 Example of increasing the stripe width for the columns in a volume



See "Performing online relayout" on page 660.

See "Permitted relayout transformations" on page 661.

Limitations of online relayout

Note the following limitations of online relayout:

- Log plexes cannot be transformed.
- Volume snapshots cannot be taken when there is an online relayout operation running on the volume.
- Online relayout cannot create a non-layered mirrored volume in a single step. It always creates a layered mirrored volume even if you specify a non-layered mirrored layout, such as `mirror-stripe` or `mirror-concat`. Use the `vxassist convert` command to turn the layered mirrored volume that results from a relayout into a non-layered volume.

- The usual restrictions apply for the minimum number of physical disks that are required to create the destination layout. For example, mirrored volumes require at least as many disks as mirrors, striped and RAID-5 volumes require at least as many disks as columns, and striped-mirror volumes require at least as many disks as columns multiplied by mirrors.
- To be eligible for layout transformation, the plexes in a mirrored volume must have identical stripe widths and numbers of columns. Relayout is not possible unless you make the layouts of the individual plexes identical.
- Online relayout cannot transform sparse plexes, nor can it make any plex sparse. (A sparse plex is a plex that is not the same size as the volume, or that has regions that are not mapped to any subdisk.)
- The number of mirrors in a mirrored volume cannot be changed using relayout. Instead, use alternative commands, such as the `vxassist mirror` command.
- Only one relayout may be applied to a volume at a time.

Transformation characteristics

Transformation of data from one layout to another involves rearrangement of data in the existing layout to the new layout. During the transformation, online relayout retains data redundancy by mirroring any temporary space used. Read and write access to data is not interrupted during the transformation.

Data is not corrupted if the system fails during a transformation. The transformation continues after the system is restored and both read and write access are maintained.

You can reverse the layout transformation process at any time, but the data may not be returned to the exact previous storage location. Before you reverse a transformation that is in process, you must stop it.

You can determine the transformation direction by using the `vxrelayout status` *`volume`* command.

These transformations are protected against I/O failures if there is sufficient redundancy and space to move the data.

Transformations and volume length

Some layout transformations can cause the volume length to increase or decrease. If either of these conditions occurs, online relayout uses the `vxresize` command to shrink or grow a file system.

See the `vxresize`(1M) manual page.

Volume resynchronization

When storing data redundantly and using mirrored or RAID-5 volumes, VxVM ensures that all copies of the data match exactly. However, under certain conditions (usually due to complete system failures), some redundant data on a volume can become inconsistent or unsynchronized. The mirrored data is not exactly the same as the original data. Except for normal configuration changes (such as detaching and reattaching a plex), this can only occur when a system crashes while data is being written to a volume.

Data is written to the mirrors of a volume in parallel, as is the data and parity in a RAID-5 volume. If a system crash occurs before all the individual writes complete, it is possible for some writes to complete while others do not. This can result in the data becoming unsynchronized. For mirrored volumes, it can cause two reads from the same region of the volume to return different results, if different mirrors are used to satisfy the read request. In the case of RAID-5 volumes, it can lead to parity corruption and incorrect data reconstruction.

VxVM ensures that all mirrors contain exactly the same data and that the data and parity in RAID-5 volumes agree. This process is called volume resynchronization. For volumes that are part of the disk group that is automatically imported at boot time (usually aliased as the reserved system-wide disk group, `bootdg`), resynchronization takes place when the system reboots.

Not all volumes require resynchronization after a system failure. Volumes that were never written or that were quiescent (that is, had no active I/O) when the system failure occurred could not have had outstanding writes and do not require resynchronization.

Dirty flags

VxVM records when a volume is first written to and marks it as dirty. When a volume is closed by all processes or stopped cleanly by the administrator, and all writes have been completed, VxVM removes the dirty flag for the volume. Only volumes that are marked dirty require resynchronization.

Resynchronization process

The process of resynchronization depends on the type of volume. For mirrored volumes, resynchronization is done by placing the volume in recovery mode (also called read-writeback recovery mode). Resynchronization of data in the volume is done in the background. This allows the volume to be available for use while recovery is taking place. RAID-5 volumes that contain RAID-5 logs can “replay” those logs. If no logs are available, the volume is placed in reconstruct-recovery mode and all parity is regenerated.

Resynchronization can impact system performance. The recovery process reduces some of this impact by spreading the recoveries to avoid stressing a specific disk or controller.

For large volumes or for a large number of volumes, the resynchronization process can take time. These effects can be minimized by using dirty region logging (DRL) and FastResync (fast mirror resynchronization) for mirrored volumes, or by using RAID-5 logs for RAID-5 volumes.

See "Dirty region logging" on page 77.

For mirrored volumes used by Oracle, you can use the SmartSync feature, which further improves performance.

Hot-relocation

Hot-relocation is a feature that allows a system to react automatically to I/O failures on redundant objects (mirrored or RAID-5 volumes) in VxVM and restore redundancy and access to those objects. VxVM detects I/O failures on objects and relocates the affected subdisks. The subdisks are relocated to disks designated as spare disks or to free space within the disk group. VxVM then reconstructs the objects that existed before the failure and makes them accessible again.

When a partial disk failure occurs (that is, a failure affecting only some subdisks on a disk), redundant data on the failed portion of the disk is relocated. Existing volumes on the unaffected portions of the disk remain accessible.

See "How hot-relocation works" on page 598.

Dirty region logging

Dirty region logging (DRL), if enabled, speeds recovery of mirrored volumes after a system crash. DRL tracks the regions that have changed due to I/O writes to a mirrored volume. DRL uses this information to recover only those portions of the volume.

If DRL is not used and a system failure occurs, all mirrors of the volumes must be restored to a consistent state. Restoration is done by copying the full contents of the volume between its mirrors. This process can be lengthy and I/O intensive.

Note: DRL adds a small I/O overhead for most write access patterns. This overhead is reduced by using SmartSync.

If an instant snap DCO volume is associated with a volume, a portion of the DCO volume can be used to store the DRL log. There is no need to create a separate DRL log for a volume which has an instant snap DCO volume.

Log subdisks and plexes

DRL log subdisks store the dirty region log of a mirrored volume that has DRL enabled. A volume with DRL has at least one log subdisk; multiple log subdisks can be used to mirror the dirty region log. Each log subdisk is associated with one plex of the volume. Only one log subdisk can exist per plex. If the plex contains only a log subdisk and no data subdisks, that plex is referred to as a log plex.

The log subdisk can also be associated with a regular plex that contains data subdisks. In that case, the log subdisk risks becoming unavailable if the plex must be detached due to the failure of one of its data subdisks.

If the `vxassist` command is used to create a dirty region log, it creates a log plex containing a single log subdisk by default. A dirty region log can also be set up manually by creating a log subdisk and associating it with a plex. The plex then contains both a log and data subdisks.

Sequential DRL

Some volumes, such as those that are used for database replay logs, are written sequentially and do not benefit from delayed cleaning of the DRL bits. For these volumes, sequential DRL can be used to limit the number of dirty regions. This allows for faster recovery. However, if applied to volumes that are written to randomly, sequential DRL can be a performance bottleneck as it limits the number of parallel writes that can be carried out.

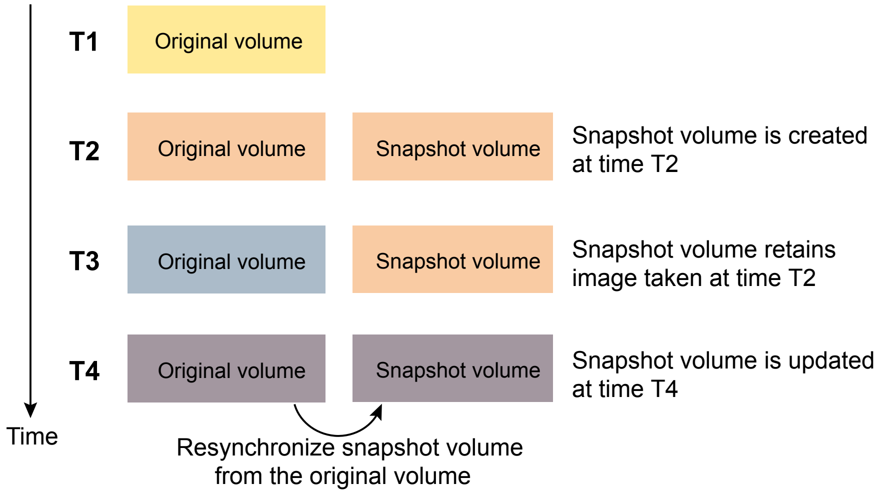
The maximum number of dirty regions allowed for sequential DRL is controlled by a tunable as detailed in the description of `voldrl_max_seq_dirty`.

Volume snapshots

Veritas Volume Manager provides the capability for taking an image of a volume at a given point in time. Such an image is referred to as a volume snapshot. Such snapshots should not be confused with file system snapshots, which are point-in-time images of a Veritas File System.

Figure 3-24 shows how a snapshot volume represents a copy of an original volume at a given point in time.

Figure 3-24 Volume snapshot as a point-in-time image of a volume


T1
Original volume
T2
Original volume
Snapshot volume
Snapshot volume is created at time T2
T3
Original volume
Snapshot volume
Snapshot volume retains image taken at time T2
T4
Original volume
Snapshot volume
Snapshot volume is updated at time T4
Time
Resynchronize snapshot volume from the original volume

Even though the contents of the original volume can change, the snapshot volume preserves the contents of the original volume as they existed at an earlier time.

The snapshot volume provides a stable and independent base for making backups of the contents of the original volume, or for other applications such as decision support. In the figure, the contents of the snapshot volume are eventually resynchronized with the original volume at a later point in time.

Another possibility is to use the snapshot volume to restore the contents of the original volume. This may be useful if the contents of the original volume have become corrupted in some way.

Warning: If you write to the snapshot volume, it may no longer be suitable for use in restoring the contents of the original volume.

One type of volume snapshot in VxVM is the third-mirror break-off type. This name comes from its implementation where a snapshot plex (or third mirror) is added to a mirrored volume. The contents of the snapshot plex are then synchronized from the original plexes of the volume. When this synchronization is complete, the snapshot plex can be detached as a snapshot volume for use in backup or decision support applications. At a later time, the snapshot plex can be reattached to the original volume, requiring a full resynchronization of the snapshot plex’s contents.

The FastResync feature was introduced to track writes to the original volume. This tracking means that only a partial, and therefore much faster, resynchronization is required on reattaching the snapshot plex. In later releases, the snapshot model was enhanced to allow snapshot volumes to contain more than a single plex, reattachment of a subset of a snapshot volume's plexes, and persistence of FastResync across system reboots or cluster restarts.

Release 4.0 of VxVM introduced full-sized instant snapshots and space-optimized instant snapshots, which offer advantages over traditional third-mirror snapshots such as immediate availability and easier configuration and administration. You can also use the third-mirror break-off usage model with full-sized snapshots, where this is necessary for write-intensive applications.

For information about how and when to use volume snapshots, see the *Veritas InfoScale Solutions Guide*.

See the `vxassist`(1M) manual page.

See the `vxsnap`(1M) manual page.

Comparison of snapshot features

Table 3-2 compares the features of the various types of snapshots that are supported in VxVM.

Table 3-2 Comparison of snapshot features for supported snapshot types

Snapshot feature	Full-sized instant (vxsnap)	Space-optimized instant (vxsnap)	Break-off (vxassist or vxsnap)
Immediately available for use on creation	Yes	Yes	No
Requires less storage space than original volume	No	Yes	No
Can be reattached to original volume	Yes	No	Yes
Can be used to restore contents of original volume	Yes	Yes	Yes
Can quickly be refreshed without being reattached	Yes	Yes	No
Snapshot hierarchy can be split	Yes	No	No
Can be moved into separate disk group from original volume	Yes	No	Yes

Table 3-2 Comparison of snapshot features for supported snapshot types *(continued)*

Snapshot feature	Full-sized instant (vxsnap)	Space-optimized instant (vxsnap)	Break-off (vxassist or vxsnap)
Can be turned into an independent volume	Yes	No	Yes
FastResync ability persists across system reboots or cluster restarts	Yes	Yes	Yes
Synchronization can be controlled	Yes	No	No
Can be moved off-host	Yes	No	Yes

Full-sized instant snapshots are easier to configure and offer more flexibility of use than do traditional third-mirror break-off snapshots. For preference, new volumes should be configured to use snapshots that have been created using the `vxsnap` command rather than using the `vxassist` command. Legacy volumes can also be reconfigured to use `vxsnap` snapshots, but this requires rewriting of administration scripts that assume the `vxassist` snapshot model.

Support for atomic writes

Veritas InfoScale supports atomic write operations on Fusion-io devices. Atomic write capable devices ensure that all blocks in write I/O operation (which may span multiple sectors) either pass or fail. If a write fails in-between, the storage reverts back to old data.

Atomic write resolves a problem of indeterminate status of failed writes that often requires two-part write – one write to an update log buffer and the other write on actual data volumes. Enabling atomic write eliminates the writes on log buffer, which in turn results in a better performance.

Storage Foundation lets you configure the atomic write support when you create a Veritas Volume Manager (VxVM) volume on a device that has atomic write capability. The atomic write I/O size of an atomic write capable volume is 16KB.

While creating an atomic write capable volume, VxVM ensures that all underlying subdisks are aligned to the 16KB boundary. Atomic write capable volumes can span multiple atomic write enabled devices, but I/O crossing atomic write boundary is not supported.

Atomic write is supported on raw VxVM volumes as well as on VxFS configured on VxVM volumes.

For information about using the Storage Foundation atomic write I/O feature with MySQL, see the *Storage Foundation and High Availability Solutions Solutions Guide*.

FastResync

Note: You need to have Veritas InfoScale Enterprise product license to use this feature.

The FastResync feature (previously called Fast Mirror Resynchronization or FMR) performs quick and efficient resynchronization of stale mirrors (a mirror that is not synchronized). This feature increases the efficiency of the Veritas Volume Manager (VxVM) snapshot mechanism, and improves the performance of operations such as backup and decision support applications. Typically, these operations require that the volume is quiescent, and that they are not impeded by updates to the volume by other activities on the system. To achieve these goals, the snapshot mechanism in VxVM creates an exact copy of a primary volume at an instant in time. After a snapshot is taken, it can be accessed independently of the volume from which it was taken.

In a Cluster Volume Manager (CVM) environment with shared access to storage, it is possible to eliminate the resource contention and performance overhead of using a snapshot simply by accessing it from a different node.

How FastResync works

FastResync provides the following enhancements to VxVM:

Faster mirror resynchronization

FastResync optimizes mirror resynchronization by keeping track of updates to stored data that have been missed by a mirror. (A mirror may be unavailable because it has been detached from its volume, either automatically by VxVM as the result of an error, or directly by an administrator using a utility such as `vxplex` or `vxassist`. A returning mirror is a mirror that was previously detached and is in the process of being re-attached to its original volume as the result of the `vxrecover` or `vxplex att` operation.) When a mirror returns to service, only the updates that it has missed need to be re-applied to resynchronize it. This requires much less effort than the traditional method of copying all the stored data to the returning mirror.

Once FastResync has been enabled on a volume, it does not alter how you administer mirrors. The only visible effect is that repair operations conclude more quickly.

See the `vxplex`(1M), `vxassist`(1M), and `vxrecover`(1M) manual pages.

Re-use of snapshots

FastResync allows you to refresh and re-use snapshots rather than discard them. You can quickly re-associate (snap back) snapshot plexes with their original volumes. This reduces the system overhead required to perform cyclical operations such as backups that rely on the volume snapshots.

FastResync can be implemented in one of two ways:

Non-persistent FastResync

Non-persistent FastResync allocates its change maps in memory. The maps do not reside on disk nor in persistent store.

See “How non-persistent FastResync works with snapshots” on page 83.

Persistent FastResync

Persistent FastResync keeps the FastResync maps on disk so that they can survive system reboots, system crashes and cluster crashes.

See “How persistent FastResync works with snapshots” on page 84.

How non-persistent FastResync works with snapshots

If FastResync is enabled on a volume before a snapshot is taken, the snapshot feature of VxVM uses FastResync change tracking to record updates to the original volume after a snapshot plex is created. When the `snapback` option is used to reattach the snapshot plex, the changes that FastResync recorded are used to resynchronize the volume during the snapback. This behavior considerably reduces the time needed to resynchronize the volume.

Non-persistent FastResync uses a map in memory to implement change tracking. The map does not exist on disk or in persistent store. The advantage of non-persistent FastResync is that updates to the FastResync map have little impact on I/O performance, because no disk updates are performed. However, FastResync must remain enabled until the snapshot is reattached, and the system cannot be rebooted. If FastResync is disabled or the system is rebooted, the information in the map is lost and a full resynchronization is required on snapback.

This limitation can be overcome for volumes in cluster-shareable disk groups, provided that at least one of the nodes in the cluster remained running to preserve the FastResync map in its memory. However, a node crash in a High Availability (HA) environment requires the full resynchronization of a mirror when it is reattached to its parent volume.

Each bit in the FastResync map represents a contiguous number of blocks in a volume's address space. The default size of the map is 4 blocks. The kernel tunable `vol_fmr_logsz` can be used to limit the maximum size in blocks of the map

For information about tuning VxVM, see the *Storage Foundation and High Availability Solutions Tuning Guide*.

How persistent FastResync works with snapshots

Persistent FastResync keeps the FastResync maps on disk so that they can survive system reboots, system crashes, and cluster crashes. Persistent FastResync uses a map in a data change object (DCO) volume on disk to implement change tracking. Each bit in the map represents a contiguous number of blocks in a volume's address space.

Persistent FastResync can also track the association between volumes and their snapshot volumes after they are moved into different disk groups. When the disk groups are rejoined, this allows the snapshot plexes to be quickly resynchronized. This ability is not supported by non-persistent FastResync.

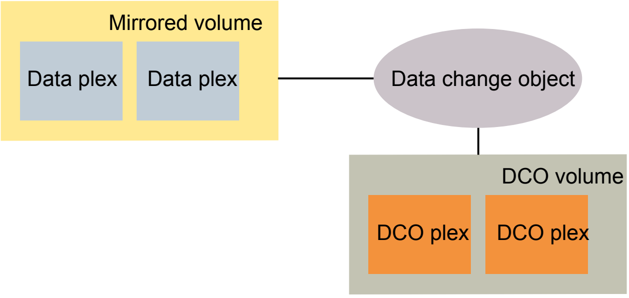
See "Reorganizing the contents of disk groups" on page 643.

When persistent FastResync is enabled on a volume or on a snapshot volume, a data change object (DCO) and a DCO volume are associated with the volume.

See "DCO volume versioning " on page 87.

Figure 3-25 shows an example of a mirrored volume with two plexes on which persistent FastResync is enabled.

Figure 3-25 Mirrored volume with persistent FastResync enabled

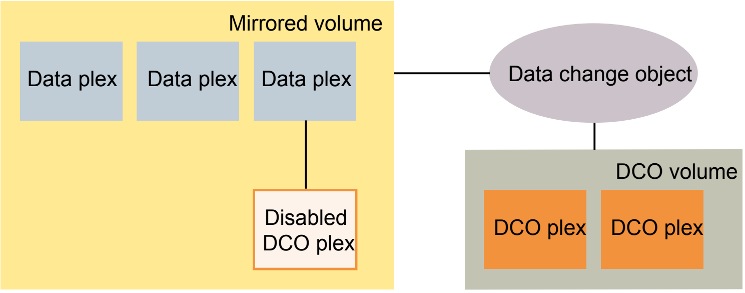


Associated with the volume are a DCO object and a DCO volume with two plexes.

Create an instant snapshot by using the `vxsnap make` command, or create a traditional third-mirror snapshot by using the `vxassist snapstart` command.

Figure 3-26 shows how a snapshot plex is set up in the volume, and how a disabled DCO plex is associated with it.

Figure 3-26 Mirrored volume after completion of a snapstart operation

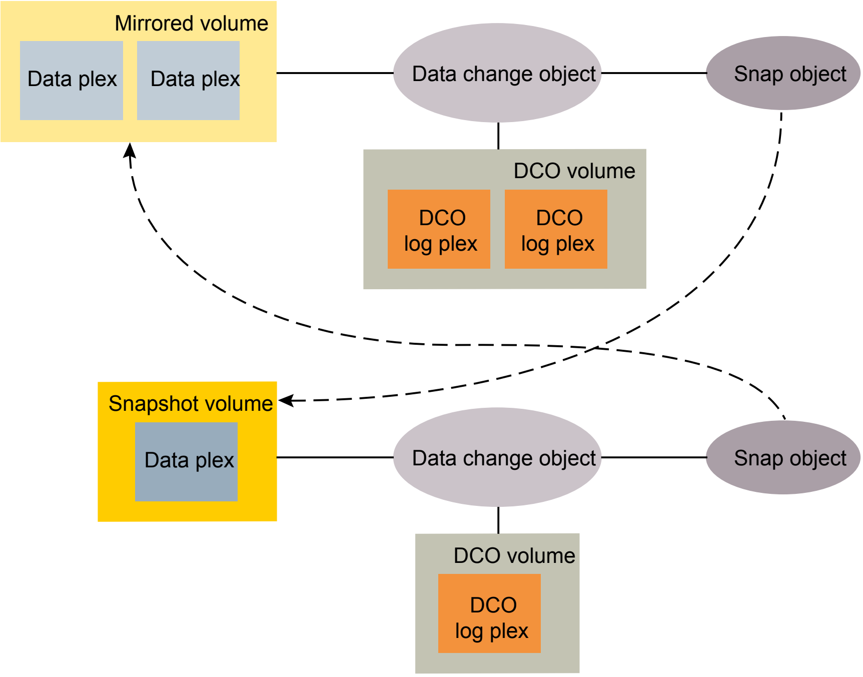


Multiple snapshot plexes and associated DCO plexes may be created in the volume by re-running the `vxassist snapstart` command for traditional snapshots, or the `vxsnap make` command for space-optimized snapshots. You can create up to a total of 32 plexes (data and log) in a volume.

A traditional snapshot volume is created from a snapshot plex by running the `vxassist snapshot` operation on the volume. For instant snapshots, however, the `vxsnap make` command makes an instant snapshot volume immediately available for use. There is no need to run an additional command.

Figure 3-27 shows how the creation of the snapshot volume also sets up a DCO object and a DCO volume for the snapshot volume.

Figure 3-27 Mirrored volume and snapshot volume after completion of a snapshot operation



The DCO volume contains the single DCO plex that was associated with the snapshot plex. If two snapshot plexes were taken to form the snapshot volume, the DCO volume would contain two plexes. For space-optimized instant snapshots, the DCO object and DCO volume are associated with a snapshot volume that is created on a cache object and not on a VxVM disk.

Associated with both the original volume and the snapshot volume are snap objects. The snap object for the original volume points to the snapshot volume, and the snap object for the snapshot volume points to the original volume. This allows VxVM to track the relationship between volumes and their snapshots even if they are moved into different disk groups.

The snap objects in the original volume and snapshot volume are automatically deleted in the following circumstances:

- For traditional snapshots, the `vxassist snapback` operation is run to return all of the plexes of the snapshot volume to the original volume.
- For traditional snapshots, the `vxassist snapclear` operation is run on a volume to break the association between the original volume and the snapshot volume.

If the volumes are in different disk groups, the command must be run separately on each volume.

- For full-sized instant snapshots, the `vxsnap reattach` operation is run to return all of the plexes of the snapshot volume to the original volume.
- For full-sized instant snapshots, the `vxsnap dis` or `vxsnap split` operations are run on a volume to break the association between the original volume and the snapshot volume. If the volumes are in different disk groups, the command must be run separately on each volume.

Note: The `vxsnap reattach`, `dis` and `split` operations are not supported for space-optimized instant snapshots.

See the `vxassist`(1M) manual page.

See the `vxsnap`(1M) manual page.

DCO volume versioning

Persistent FastResync uses a data change object (DCO) and a DCO volume to hold the FastResync maps.

This release of Veritas Volume Manager (VxVM) supports the following DCO volume versions:

Version 30 DCO volume layout	An extension of the Version 20 DCO volume layout with the same functionality. Version 30 is the default version with this release.
Instant snap DCO volume layout	Previously known as Version 20 DCO volume layout, this version of the DCO layout supports instant snapshots of volumes. This type of DCO manages the FastResync maps, and also manages DRL recovery maps and special maps called copymaps that allow instant snapshot operations to resume correctly following a system crash.
Version 0 DCO volume layout	This version of the DCO volume layout only supports legacy snapshots (vxassist snapshots). The DCO object manages information about the FastResync maps. These maps track writes to the original volume and to each of up to 32 snapshot volumes since the last `snapshot` operation. Each plex of the DCO volume on disk holds 33 maps, each of which is 4 blocks in size by default. VxVM software continues to support the version 0 (zero) layout for legacy volumes.

Instant snap (version 20) DCO volume layout

The instant snap data change object (DCO) supports full-sized and space-optimized instant snapshots. Traditional third-mirror volume snapshots that are administered using the `vxassist` command are not supported with this DCO layout.

Introduced in Veritas Volume Manager (VxVM) 4.0, the instant snap DCO volume layout is also known as a version 20 DCO volume layout. This type of DCO is used not only to manage the FastResync maps, but also to manage DRL recovery maps and special maps called copymaps that allow instant snapshot operations to resume correctly following a system crash.

Introduced in InfoScale 8.0, the version 30 DCO volume layout is an extension of the version 20 DCO volume layout, which continues to support all the existing functionality. This volume layout is now the default.

See "Dirty region logging" on page 77.

Each bit in a map represents a region (a contiguous number of blocks) in a volume's address space. A region represents the smallest portion of a volume for which changes are recorded in a map. A write to a single byte of storage anywhere within a region is treated in the same way as a write to the entire region.

The layout of an instant snap DCO volume uses dynamic creation of maps on the preallocated storage. The size of the DRL (Dirty region logging) map does not depend on volume size. You can configure the size of the DRL by using the option `drlmapsz` while creating the DCO volume. By default, the size of the DRL is 1MB.

For CVM configurations, each node has a dedicated DRL map that gets allocated during the first write on that node. By default, the size of the DCO volume accommodates 32 DRL maps, an accumulator, and 16 per-volume maps (including a DRL recovery map, a detach map to track detached plexes, and the remaining 14 maps for tracking snapshots).

The size of the DCO plex can be estimated using the following formula:

```
DCO_volume_size = (32*drlmapsize + acmsize + 16*per-volume_map_size)
```

where:

```
acmsize = (volume_size / (region_size*4))
```

```
per-volume_map_size = (volume_size/region_size*8)
```

```
drlmapsize = 1M, by default
```

For a 100GB volume, the size of the DCO volume with the default *regionsize* of 64KB is approximately 36MB.

Create the DCOs for instant snapshots by using the `vxsnap prepare` command or by specifying the options `logtype=dco dcoversion=20` while creating a volume with the `vxassist make` command.

Version 0 DCO volume layout

The version 0 DCO volume layout supports only traditional (third-mirror) volume snapshots that are administered using the `vxassist` command. Full-sized and space-optimized instant snapshots are not supported with this DCO layout.

The size of each map can be changed by specifying the `dcolen` attribute to the `vxassist` command when the volume is created. The default value of `dcolen` is 132 blocks (the plex contains 33 maps, each of length 4 blocks). To use a larger map size, multiply the desired map size by 33 to calculate the value of `dcolen`. For example, to use an 8-block map, specify `dcolen=264`. The maximum possible map size is 64 blocks, which corresponds to a `dcolen` value of 2112 blocks.

The size of a DCO plex is rounded up to the nearest integer multiple of the disk group alignment value. The alignment value is 8KB for disk groups that support the Cross-platform Data Sharing (CDS) feature. Otherwise, the alignment value is 1 block.

Effect of growing a volume on the FastResync map

It is possible to grow the replica volume, or the original volume, and still use FastResync. According to the DCO volume layout, growing the volume has the following different effects on the map that FastResync uses to track changes to the original volume:

- For an instant snap DCO volume, the size of the map is increased and the size of the region that is tracked by each bit in the map stays the same.
- For a version 0 DCO volume, the size of the map remains the same and the region size is increased.

In either case, the part of the map that corresponds to the grown area of the volume is marked as “dirty” so that this area is resynchronized. The `snapback` operation fails if it attempts to create an incomplete snapshot plex. In such cases, you must grow the replica volume, or the original volume, before invoking any of the commands `vxsnap reattach`, `vxsnap restore`, or `vxassist snapback`. Growing the two volumes separately can lead to a snapshot that shares physical disks with another mirror in the volume. To prevent this, grow the volume after the `snapback` command is complete.

See the `vxsnap`(1M) and the `vxassist`(1M) manual pages.

FastResync limitations

The following limitations apply to FastResync:

- Persistent FastResync is supported for RAID-5 volumes, but this prevents the use of the relayout or resize operations on the volume while a DCO is associated with it.
- Neither non-persistent nor persistent FastResync can be used to resynchronize mirrors after a system crash. Dirty region logging (DRL), which can coexist with FastResync, should be used for this purpose. In VxVM 4.0 and later releases, DRL logs may be stored in an instant snap DCO volume.
- When a subdisk is relocated, the entire plex is marked “dirty” and a full resynchronization becomes necessary.
- If a snapshot volume is split off into another disk group, non-persistent FastResync cannot be used to resynchronize the snapshot plexes with the original volume when the disk group is rejoined with the original volume’s disk group. Persistent FastResync must be used for this purpose.
- If you move or split an original volume (on which persistent FastResync is enabled) into another disk group, and then move or join it to a snapshot volume’s disk group, you cannot use `vxassist snapback` to resynchronize traditional snapshot plexes with the original volume. This restriction arises because a snapshot volume references the original volume by its record ID at the time that the snapshot volume was created. Moving the original volume to a different disk group changes the volume’s record ID, and so breaks the association. However, in such a case, you can use the `vxplex snapback` command with the `-f` (force) option to perform the snapback.

  Note: This restriction only applies to traditional snapshots. It does not apply to instant snapshots.

- Any operation that changes the layout of a replica volume can mark the FastResync change map for that snapshot “dirty” and require a full resynchronization during snapback. Operations that cause this include subdisk split, subdisk move, and online relayout of the replica. It is safe to perform these operations after the snapshot is completed.

  See the `vxassist` (1M) manual page.

  See the `vxplex` (1M) manual page.

  See the `vxvol` (1M) manual page.

Volume sets

Volume sets are an enhancement to Veritas Volume Manager (VxVM) that allow several volumes to be represented by a single logical object. All I/O from and to the underlying volumes is directed by way of the I/O interfaces of the volume set. Veritas File System (VxFS) uses volume sets to manage multi-volume file systems and the SmartTier feature. This feature allows VxFS to make best use of the different performance and availability characteristics of the underlying volumes. For example, file system metadata can be stored on volumes with higher redundancy, and user data on volumes with better performance.

See "Creating a volume set" on page 505.

How VxVM handles hardware clones or snapshots

Advanced disk arrays provide methods to create copies of physical volumes (disks or LUNs) from the hardware side.

You can create a hardware snapshot (such as an EMC BCV™ or Hitachi ShadowImage™), a hardware mirror, or a hardware clone. You can also use `dd` or a similar command to clone the disk content.

If the physical volumes are VxVM disks, using a hardware copy method also copies the configuration data stored in the private region of the VxVM managed disk. The hardware disk copy becomes a duplicate of the original VxVM disk. For VxVM to handle the duplicated disk images correctly, VxVM must distinguish between the original and duplicate disk images.

VxVM detects that a disk is a hardware copy, to ensure that the duplicate disks are not confused with the original disks. This functionality enables the server to import a consistent set of disks. By default, VxVM imports the original physical volume but VxVM also enables you to work with the hardware copies on the same server. VxVM provides special options to import a disk group with the cloned images and make a cloned disk group with a unique identity. With care, you can manage multiple sets of hardware copies, even from the same server.

See "Importing a disk group containing hardware cloned disks " on page 694.

VxVM provides the following functionality to handle hardware copies:

Functionality	Description
Distinguishes between the hardware copy and the original data disk.	VxVM discovers a unique disk identifier (UDID) for each disk from the attributes of the hardware disk and stores this value. VxVM compares the discovered UDID to the stored value to detect if a disk is a hardware copy.
Prevents inadvertent sharing over the SAN of an original LUN and one or more of its point-in time copies, mirrors, or replicated copies.	By default, when you import a VxVM disk group, VxVM prevents disks that are identified as clones or copies from being imported. This behavior prevents mistakenly importing a mix of original disks and hardware copies.
Imports the hardware copies as a clone disk group or as a new standard disk group.	If you choose to import the hardware copies of the disks of a VxVM disk group, VxVM identifies the disks as clone disks. You can choose whether to maintain the clone disk status or create a new standard disk group.
Detects the LUN class of the array.	VxVM detects the extended attributes of the array, including the LUN class. The LUN class can help to identify which disks are hardware copies of the VxVM disks.
Provides disk tagging to label and manage sets of disks.	If you create multiple copies of the same set of volumes, you as administrator need to identify which disk copies make up a consistent set of disks. You can use VxVM disk tags to label the sets of disks. For example, if you have multiple point in time snapshots of the same LUN, you can label each with a separate disk tag. Specify the tag to the import operation to import the tagged snapshot LUN.

How VxVM uses the unique disk identifier (UDID)

Veritas Volume Manager (VxVM) uses a unique disk identifier (UDID) to detect hardware copies of the VxVM disks. Before using a physical volume, VxVM always verifies whether the disk already has a UDID and whether that UDID matches the expected value.

When you initialize a VxVM disk, the Device Discovery Layer (DDL) of VxVM determines the UDID from hardware attributes such as the vendor ID (vid), the product ID (pid), the cabinet serial number, and the LUN serial number. VxVM stores the UDID in the private region of the disk when VxVM first sees a disk that does not have a UDID, or when VxVM initializes the disk. The exact make-up of the UDID depends on the array storage library (ASL). Future versions of VxVM may use different formats for new arrays.

When VxVM discovers a disk with a UDID, VxVM compares the current UDID value (the value determined from the hardware attributes) to the UDID that is already stored on the disk. If the UDID values do not match between the UDID value determined by the DDL and the on-disk UDID, VxVM sets the `udid_mismatch` flag for the disk.

The `udid_mismatch` flag generally indicates that the disk is a hardware copy of a VxVM disk. The hardware copy has a copy of the VxVM private region of the original disk, including the UDID. The UDID already stored in the VxVM private region matches the attributes of the original hardware disk, but does not match the value on the hardware disk that is the copy.

With the UDID matching feature, VxVM can prevent the situation where the inconsistent set of disks is presented to the host. This functionality enables you to import a disk group composed of LUN snapshots on the same host as the original LUNs. When you import the disks identified with the `udid_mismatch` flag, VxVM sets the `clone_disk` flag on the disk. With care, multiple hardware images of the original LUN can be simultaneously managed and imported on the same host as the original LUN.

See “Importing a disk group containing hardware cloned disks ” on page 694.

If a system only sees the copy (or clone) devices, you can remove the `clone_disk` flags. Only remove the `clone_disk` flags if you are sure there is no risk. For example, you must make sure that there are not two physical volumes that are copies of the same base physical volume at different times.

If the `udid_mismatch` flag is set incorrectly on a disk that is not a clone disk, you can remove the `udid_mismatch` flag and treat the disk as a standard disk.

See the *Veritas InfoScale Troubleshooting Guide*.

Volume encryption

VxVM provides advanced security for data at rest through encryption of VxVM data volumes. Encryption is a technology that converts data or information into a code that can be decrypted only by authorized users.

Veritas InfoScale supports encrypting data at rest and data encryption over wire. It helps you to take regular backups of the encrypted volumes. See the *Veritas InfoScale™ 7.3.1 Replication Administrator's Guide*

You can encrypt VxVM data volumes to:

- Protect sensitive data from unauthorized access
- Retire disks from use or ship them for replacement without the overhead of secure wiping of content

The implementation uses the Advanced Encryption Standard (AES) cryptographic algorithm with 256-bit key size validated by the Federal Information Processing Standard (FIPS) Publication 140-2, (FIPS PUB 140-2) security standard.

You can encrypt volumes or disk groups in your storage environment. VxVM generates a volume encryption key at the time of volume creation. The volume encryption key is secured (wrapped) using a key wrap. The wrapped key is stored with the volume record. The volume encryption key is not stored on disk.

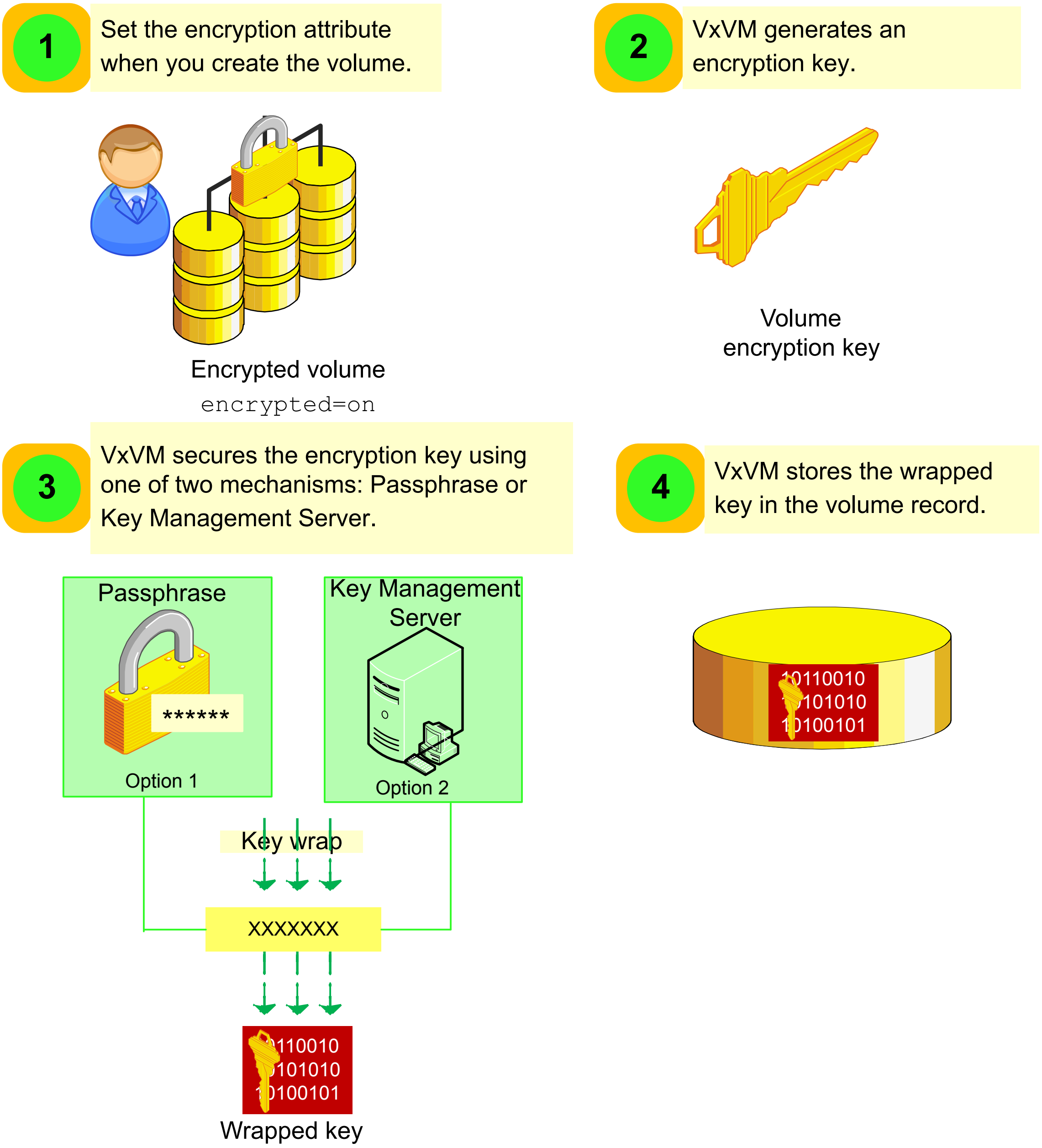
For the encrypted volumes that are created with disk group version 300 or later, InfoScale uses the FIPS 140-2 standards to validate the wrapping key (KEK) that secures volume encryption. For the encrypted volumes that are created with disk group version 290 or earlier, you must first upgrade to disk group version 300 or later. Thereafter, you can use the re-key operation to ensure that InfoScale uses the FIPS standards to validate the KEK.

You can secure the volume encryption key using one of the following methods:

Using Passphrases (PBE)	See "Using passphrases for encryption" on page 98.
Using Key Management Server (KMS)	See "Using Key Management Server for encryption" on page 98.

Figure 3-28 describes the encryption process.

Figure 3-28 Encryption volume creation


1
Set the encryption attribute when you create the volume.
Encrypted volume
encrypted=on
2
VxVM generates an encryption key.
Volume encryption key
3
VxVM secures the encryption key using one of two mechanisms: Passphrase or Key Management Server.
Passphrase
******
Option 1
Key Management Server
Option 2
Key wrap
XXXXXXX
0110010
0101010
10100101
Wrapped key
4
VxVM stores the wrapped key in the volume record.
10110010
10101010
10100101

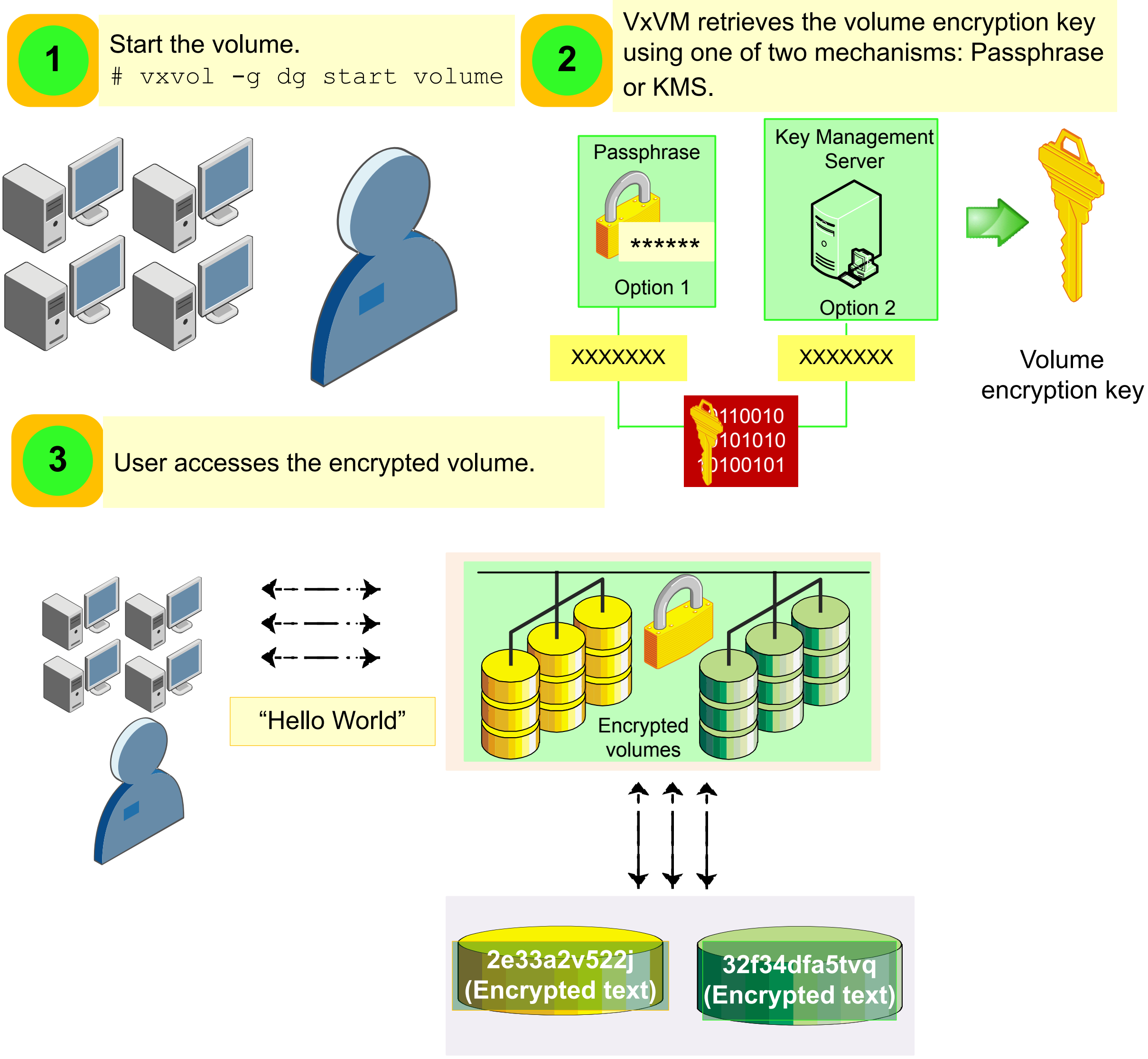
If you encrypt a disk group, all volumes in the disk group are encrypted. Any volume created later on the disk group will also be encrypted by default.

Only new volumes that are created using disk group version 220 or later can be encrypted by VxVM.

When you start an encrypted volume, VxVM uses the key wrap to retrieve the volume encryption key and enable access to the volume.

Figure 3-29 illustrates the decryption process.

Figure 3-29 Encryption and decryption process



1 Start the volume.

```
# vxvol -g dg start volume
```

2 VxVM retrieves the volume encryption key using one of two mechanisms: Passphrase or KMS.

3 User accesses the encrypted volume.

The following capabilities are not supported by VxVM encryption:

- Encryption of root and swap volumes
- Encryption of volumes that use RAID-5 layout
- Encryption of existing volumes

- Linked break-off snapshots

Using passphrases for encryption

When you encrypt a volume, VxVM generates a volume encryption key. The volume encryption key needs to be secured using a key wrap. If you choose to use the passphrase mechanism, VxVM prompts for a passphrase, then uses a hash algorithm to derive the key wrap from the specified passphrase. No additional hardware or software is required to use this mechanism. The passphrase must be randomly generated and must have high entropy.

Volumes that are encrypted using passphrases must be manually started whenever the system boots up or is restarted. This is because the volume prompts for authentication when the system starts. However, you can enable automated startup for encrypted volumes by providing the required passphrases in a file.

Passphrase-based encryption is suitable for environments that do not depend heavily on automated configurations.

Using Key Management Server for encryption

VxVM supports the use of a Key Management Server (KMS) that conforms to the OASIS Key Management Interoperability Protocol (KMIP) specification.

During creation of encrypted volumes:

- VxVM sends a key generation request to the configured KMS using the KMIP protocol.
- KMS responds with a unique identifier. VxVM sends the identifier to KMS to obtain the key generated by KMS.
- KMS responds with the key; VxVM generates the random volume encryption key, and encrypts it using the key provided by KMS.
- VxVM stores the encrypted key and the KMS identifier in the volume record.

During startup of encrypted volumes:

- VxVM retrieves the encrypted key and the KMS identifier from the volume record.
- VxVM sends the identifier to KMS to obtain the key.
- KMS responds with the key; VxVM decrypts the encrypted key (stored in the volume record) with the key provided by KMS.

KMS-based encryption is suitable for environments that support high availability and automated configurations.

With a Key Management Server, you can:

- Eliminate the need to remember complex passphrases
- Back up or replicate keys for disaster recovery
- Use a single KMS key to encrypt or to retrieve the volume encryption key for all the volumes in a disk group.
- Change or rotate the KMS keys for volumes

VxVM supports Key Management Servers that conform to the OASIS KMIP specification.

VxVM configures the server using the configuration information in the file `/etc/vx/enc-kms-kmip.conf` located on the KMIP client.

See "Configuring a Key Management Server" on page 155.

Recommendations for encryption

It is recommended to use CPUs designed to support Advanced Encryption Standard Instruction Set (or the Intel Advanced Encryption Standard New Instructions (AES-NI) to improve performance.

You can use the following command to verify whether or not the processor supports encryption acceleration:

```
$ grep -o aes /proc/cpuinfo
aes
aes
aes
aes
```

If the command does not produce any output, the processor does not support encryption acceleration.

Disk group level encryption key management and key rotation (Re-Key)

InfoScale supports the use of a single KMS key for all the volumes in a disk group. Consequently, you can maintain a common KMS key at the disk group level instead of maintaining an individual KMS key for each volume. When you start an encrypted volume that has a common KMS key with the disk group, VxVM needs to fetch only one key to enable access to the volume. Thus, a common KMS key reduces the network load that is sent to the KMS in the form of multiple requests based on the number of volumes. A single request to KMS lets you to start all the volumes in a single operation.

To make the use of this single key more secure, InfoScale provides the option to re-key the volumes that change the KMS key when needed. This option is also known as key rotation. You can use an external scheduler based on your policy to schedule the re-key operation.

To use a single key for all the encrypted volumes in a disk group, set the value of the `same_enckey` tunable to yes as follows:

At the time of disk group creation, set:

```
vxdg -o same_enckey=yes init DiskGroupName diskName1 diskName2 …
diskNameN
```

The re-key operation behaves as follows:

- It does not change the volume encryption key.
- It retrieves the encrypted volume encryption key using the existing KMS key, and then encrypts it again with the new KMS key.
- It stores the newly encrypted volume encryption key and the new KMS identifier of the changed KMS key in the volume record.

Note: The disk group level encryption key management and key rotation feature does not support VVR configuration and disk group operations like join, split, move.

How Veritas File System works

This chapter includes the following topics:

- Veritas File System features
- Veritas File System performance enhancements
- Using Veritas File System

Veritas File System features

Table 4-1 lists the Veritas File System (VxFS) features.

Table 4-1 Veritas File System features

Feature	Description
Access Control Lists	An Access Control List (ACL) stores a series of entries that identify specific users or groups and their access privileges for a directory or file. A file may have its own ACL or may share an ACL with other files. ACLs have the advantage of specifying detailed access permissions for multiple users and groups. On Linux, ACLs are supported on cluster file systems. See the `getfacl`(1) and `setfacl`(1) manual pages.

Table 4-1 Veritas File System features *(continued)*

Feature	Description
Cluster file systems	Storage Foundation Cluster File System High Availability (SFCFSHA) allows clustered servers to mount and use a file system simultaneously as if all applications using the file system were running on the same server. The Veritas Volume Manager cluster functionality (CVM) makes logical volumes and raw device applications accessible through a cluster. SFCFSHA uses a symmetric architecture in which all nodes in the cluster can simultaneously function as metadata servers. SFCFSHA still has some remnants of the old master/slave or primary/secondary concept. The first server to mount each cluster file system becomes its primary; all other nodes in the cluster become secondaries. Applications access the user data in files directly from the server on which they are running. Each SFCFSHA node has its own intent log. File system operations, such as allocating or deleting files, can originate from any node in the cluster. In some environments that use shared disk groups with Flexible Storage Sharing feature turned on, the data may be fetched from other nodes using CVM I/O Shipping. Installing VxFS and enabling the cluster feature does not create a cluster file system configuration. File system clustering requires other Veritas products to enable communication services and provide storage resources. These products are packaged with VxFS in SFCFSHA to provide a complete clustering environment. See the *Storage Foundation Cluster File System High Availability Administrator's Guide*. To be a cluster mount, a file system must be mounted using the `mount -o cluster` option. File systems mounted without the `-o cluster` option are termed local mounts. See the `mount_vxfs`(1M) manual page. SFCFSHA functionality is available with both Veritas InfoScale Storage and Veritas InfoScale Enterprise licenses.

Table 4-1 Veritas File System features *(continued)*

Feature	Description
Cross-platform data sharing	Cross-platform data sharing (CDS) allows data to be serially shared among heterogeneous systems where each system has direct access to the physical devices that hold the data. This feature can be used only in conjunction with Veritas Volume Manager (VxVM). See the *Veritas InfoScale Solutions Guide*.
Data deduplication	You can perform post-process periodic deduplication in a file system to eliminate duplicate data without any continuous cost. You can verify whether data is duplicated on demand, and then efficiently and securely eliminate the duplicates. This feature is available with both Veritas InfoScale Storage and Veritas InfoScale Enterprise licenses. See "About deduplicating data" on page 614.
Defragmentation	You can perform defragmentation to remove unused space from directories, make all small files contiguous, and consolidate free blocks for file system use. See "About defragmentation" on page 114.
Enhanced data integrity modes	VxFS has the following `mount` command options to enable the enhanced data integrity modes: ■ `blkclear` See "blkclear mount option" on page 170. ■ `closesync` See "mincache mount option" on page 170. ■ `log` See "log mount option" on page 168.
Enhanced performance mode	The default VxFS logging mode, `mount -o delaylog`, increases performance by delaying the logging of some structural changes. However, `delaylog` does not provide the equivalent data integrity as the enhanced data integrity modes because recent changes may be lost during a system failure. This option provides at least the same level of data accuracy that traditional UNIX file systems provide for system failures, along with fast file system recovery. See the `mount_vxfs`(1M) manual page. See "delaylog mount option" on page 168.

Table 4-1 Veritas File System features *(continued)*

Feature	Description
Enhanced security	RHEL provides user-level security functionalities and features for file systems. These security functionalities can be availed if you enable SELinux at the OS-level. See the `mount_vxfs(1M)` manual page.
Extent attributes	VxFS allocates disk space to files in groups of one or more adjacent blocks called extents. VxFS defines an application interface that allows programs to control various aspects of the extent allocation for a given file. The extent allocation policies associated with a file are referred to as extent attributes. See "About extent attributes" on page 185.
Extent-based allocation	An extent is a contiguous area of storage in a computer file system, reserved for a file. When starting to write to a file, a whole extent is allocated. When writing to the file again, the data continues where the previous write left off. This reduces or eliminates file fragmentation. An extent is presented as an address-length pair, which identifies the starting block address and the length of the extent (in file system or logical blocks). Since VxFS is an extent-based file system, addressing is done through extents (which can consist of multiple blocks) rather than in single-block segments. Extents can therefore enhance file system throughput. See "About extents" on page 28.
Extended mount options	The VxFS file system provides the following enhancements to the `mount` command: ■ Enhanced data integrity modes ■ Enhanced performance mode ■ Temporary file system mode ■ Improved synchronous writes ■ Support for large file sizes See "Mounting a VxFS file system" on page 165.

Table 4-1 Veritas File System features *(continued)*

Feature	Description
Fast file system recovery	Most file systems rely on full structural verification by the `fsck` utility as the only means to recover from a system failure. For large disk configurations, this involves a time-consuming process of checking the entire structure, verifying that the file system is intact, and correcting any inconsistencies. VxFS provides fast recovery with the VxFS intent log and VxFS intent log resizing features. See "About the Veritas File System intent log" on page 27.
File Change Log	The VxFS File Change Log (FCL) tracks changes to files and directories in a file system. The File Change Log can be used by applications such as backup products, webcrawlers, search and indexing engines, and replication software that typically scan an entire file system searching for modifications since a previous scan. FCL functionality is available on all the four Veritas InfoScale licenses: Veritas InfoScale™ Storage, Veritas InfoScale™ Availability, Veritas InfoScale™ Foundation, and Veritas InfoScale™ Enterprise . See "About Veritas File System File Change Log" on page 766.
File compression	Compressing files reduces the space used by files, while retaining the accessibility of the files and being transparent to applications. Compressed files look and behave almost exactly like uncompressed files: the compressed files have the same name, and can be read and written as with uncompressed files. Reads cause data to be uncompressed in memory, only; the on-disk copy of the file remains compressed. In contrast, after a write, the new data is uncompressed on disk. See "About compressing files" on page 626.
File replication	You can perform cost-effective periodic replication of data over IP networks, giving organizations an extremely flexibile storage independent data availability solution for disaster recovery and off-host processing. See the *Veritas InfoScale Replication Administrator's Guide.*.

Table 4-1 Veritas File System features *(continued)*

Feature	Description
File system snapshots	VxFS provides online data backup using the snapshot feature. An image of a mounted file system instantly becomes an exact read-only copy of the file system at a specific point in time. The original file system is called the snapped file system, while the copy is called the snapshot. When changes are made to the snapped file system, the old data is copied to the snapshot. When the snapshot is read, data that has not changed is read from the snapped file system, changed data is read from the snapshot. Backups require one of the following methods: ■ Copying selected files from the snapshot file system (using `find` and `cpio`) ■ Backing up the entire file system (using `fscat`) ■ Initiating a full or incremental backup (using `vxdump`) See "About snapshot file systems" on page 376.
FileSnaps	A FileSnap is a space-optimized copy of a file in the same name space, stored in the same file system. VxFS supports FileSnaps on file systems with disk layout Version 8 or later. See "About FileSnaps" on page 373.
Freezing and thawing file systems	Freezing a file system is a necessary step for obtaining a stable and consistent image of the file system at the volume level. Consistent volume-level file system images can be obtained and used with a file system snapshot tool. See "Freezing and thawing a file system" on page 336.
Improved synchronous writes	VxFS provides superior performance for synchronous write applications. The `mount -o datainlog` option greatly improves the performance of small synchronous writes. The `mount -o convosync=dsync` option improves the performance of applications that require synchronous data writes but not synchronous inode time updates. See the `mount_vxfs`(1M) manual page. **Warning:** The use of the `-o convosync=dsync` option violates POSIX semantics. See "convosync mount option" on page 171.

Table 4-1 Veritas File System features *(continued)*

Feature	Description

maxlink support	Added support for more than 64K sub-directories. If `maxlink` is disabled on a file system, the sub-directory limit will be 32K by default. If `maxlink` is enabled on a file system, this allows you to create up to 4294967295(2^32 – 1) sub-directories.
	By default `maxlink` is enabled.
	To enable the `maxlink` option at `mkfs` time. For example:
	`# mkfs -t vxfs -o maxlink /dev/vx/rdsk/testdg/vol1`
	To disable the `maxlink` option at `mkfs` time. For example:
	`# mkfs -t vxfs -o nomaxlink /dev/vx/rdsk/testdg/vol1`
	To enable the `maxlink` option through the `fsadm` command on a mounted files system. For example:
	`# fsadm -t vxfs -o maxlink /mnt1`
	To disable the `maxlink` option through the `fsadm` command on a mounted file system. For example:
	`# fsadm -t vxfs -o nomaxlink /mnt1`
	See the `mkfs_vxfs`(1M) and `fsadm_vxfs`(1M) manual pages.

Multi-volume file systems	The multi-volume file system (MVFS) feature allows several volumes to be represented by a single logical object. All I/O to and from an underlying logical volume is directed by way of volume sets. You can create a single VxFS file system on this multi-volume set. This feature can be used only in conjunction with VxVM. MVFS functionality is available on all the four Veritas InfoScale licenses: Veritas InfoScale™ Storage, Veritas InfoScale™ Availability, Veritas InfoScale™ Foundation, and Veritas InfoScale™ Enterprise. See "About multi-volume file systems" on page 512.

Table 4-1 Veritas File System features *(continued)*

Feature	Description
Partitioned directories	Parallel threads that access a large volume and perform access and updates on a directory that commonly exist in a file system, suffer from an exponentially longer wait time for the threads. This feature creates partitioned directories to improve the directory performance of file systems. When any directory crosses the tunable threshold, this feature takes an exclusive lock on the directory inode and redistributes the entries into various respective hash directories. These hash directories are not visible in the name-space view of the user or operating system. For every new create, delete, or lookup thread, this feature performs a lookup for the respective hashed directory (depending on the target name) and performs the operation in that directory. This leaves the parent directory inode and its other hash directories unobstructed for access, which vastly improves file system performance. This feature operates only on disk layout Version 8 or later file systems. See "Partitioned directories" on page 795. See the vxtunefs(1M) and fsadm_vxfs(1M) manual pages.
Quotas	VxFS supports quotas, which allocate per-user and per-group quotas and limit the use of two principal resources: files and data blocks. You can assign quotas for each of these resources. Each quota consists of two limits for each resource: hard limit and soft limit. The hard limit represents an absolute limit on data blocks or files. A user can never exceed the hard limit under any circumstances. The soft limit is lower than the hard limit and can be exceeded for a limited amount of time. This allows users to exceed limits temporarily as long as they fall under those limits before the allotted time expires. See "About Veritas File System quota limits" on page 758.

Table 4-1 Veritas File System features *(continued)*

Feature	Description
Reverse path name lookup	The reverse path name lookup feature obtains the full path name of a file or directory from the inode number of that file or directory. The reverse path name lookup feature can be useful for a variety of applications, such as for clients of the VxFS File Change Log feature, in backup and restore utilities, and for replication products. Typically, these applications store information by inode numbers because a path name for a file or directory can be very long, thus the need for an easy method of obtaining a path name. See "About reverse path name lookup" on page 791.
SmartIO	The SmartIO feature of Storage Foundation and High Availability Solutions (SFHA Solutions) enables data efficiency on SSDs or other supported devices through I/O caching. Using SmartIO to improve efficiency, you can optimize the cost per IOPS. SmartIO uses advanced, customizable heuristics to determine what data to cache and how that data gets removed from the cache. The heuristics take advantage of SFHA Solutions' knowledge of the characteristics of the workload. SmartIO uses a cache area on the target device or devices. The cache area is the storage space that SmartIO uses to store the cached data and the metadata about the cached data. The type of the cache area determines whether it supports VxFS caching or VxVM caching. See the *Veritas InfoScale SmartIO for Solid State Drives Solutions Guide*.
SmartTier	The SmartTier option is built on a multi-volume file system. Using SmartTier, you can map more than one volume to a single file system. You can then configure policies that automatically relocate files from one volume to another, or relocate files by running file relocation commands. Having multiple volumes lets you determine where files are located, which can improve performance for applications that access specific types of files. SmartTier functionality is available with both Veritas InfoScale Storage and Veritas InfoScale Enterprise licenses. **Note:** In the previous VxFS 5.x releases, SmartTier was known as Dynamic Storage Tiering. See "About SmartTier" on page 528.

Table 4-1 Veritas File System features *(continued)*

Feature	Description
Storage Checkpoints	To increase availability, recoverability, and performance, VxFS offers on-disk and online backup and restore capabilities that facilitate frequent and efficient backup strategies. Backup and restore applications can leverage a Storage Checkpoint, a disk- and I/O-efficient copying technology for creating periodic frozen images of a file system. Storage Checkpoints present a view of a file system at a point in time, and subsequently identifies and maintains copies of the original file system blocks. Instead of using a disk-based mirroring method, Storage Checkpoints save disk space and significantly reduce I/O overhead by using the free space pool available to a file system. Storage Checkpoint functionality is available with both Veritas InfoScale Storage and Veritas InfoScale Enterprise licenses. See "About Storage Checkpoints" on page 435.
Support for large files and large file systems	VxFS supports files larger than two gigabytes and large file systems up to 256 terabytes. **Warning:** Some applications and utilities might not work on large files. See "largefiles and nolargefiles mount options" on page 174.
Temporary file system mode	On most UNIX systems, temporary file system directories, such as `/tmp` and `/usr/tmp`, often hold files that do not need to be retained when the system reboots. The underlying file system does not need to maintain a high degree of structural integrity for these temporary directories. VxFS provides the `mount -o tmplog` option, which allows the user to achieve higher performance on temporary file systems by delaying the logging of most operations. See the `mount_vxfs`(1M) manual page. See "tmplog mount option" on page 169.
Thin Reclamation	The Thin Reclamation feature allows you to release free data blocks of a VxFS file system to the free storage pool of a Thin Storage LUN. This feature is only supported on file systems created on a VxVM volume. See "About Thin Reclamation of a file system" on page 481.

Veritas File System performance enhancements

Traditional file systems employ block-based allocation schemes that provide adequate random access and latency for small files, but limit throughput for larger files. As a result, they are less than optimal for commercial environments.

Veritas File System (VxFS) addresses this file system performance issue through an alternative allocation method and increased user control over allocation, I/O, and caching policies.

See "Using Veritas File System" on page 113.

VxFS provides the following performance enhancements:

- Data synchronous I/O
  See "Data synchronous I/O" on page 334.
- Direct I/O and discovered direct I/O
  See "Direct I/O" on page 333.
  See "Discovered Direct I/O" on page 334.
- Delayed allocation for extending writes
  See "Delayed allocation for extending writes" on page 113.
- Enhanced I/O performance
  See "Enhanced I/O performance" on page 112.
- Caching advisories
  See "Cache advisories" on page 336.
- Enhanced directory features
- Explicit file alignment, extent size, and preallocation controls
  See "Extent attribute alignment" on page 188.
  See "Fixed extent size" on page 186.
  See "Reservation: preallocating space to a file" on page 186.
- Tunable I/O parameters
  See "Tuning the VxFS file system" on page 793.
- Integration with Veritas Volume Manager (VxVM)
  See "About Veritas Volume Manager" on page 26.
- Support for large directories

  Note: VxFS reduces the file lookup time in directories with an extremely large number of files.

- Partitioned directories

See the `vxtunefs`(1M) and `fsadm_vxfs`(1M) manual pages.

Enhanced I/O performance

Veritas File System (VxFS) provides enhanced I/O performance by applying an aggressive I/O clustering policy, integrating with Veritas Volume Manager (VxVM), and allowing application-specific parameters to be set on a per-file system basis.

See “Enhanced I/O clustering” on page 112.

See “Veritas Volume Manager integration with Veritas File System for enhanced I/O performance” on page 112.

See “Application-specific parameters for enhanced I/O performance” on page 112.

Enhanced I/O clustering

I/O clustering is a technique of grouping multiple I/O operations together for improved performance. Veritas File System (VxFS) I/O policies provide more aggressive clustering processes than other file systems and offer higher I/O throughput when using large files. The resulting performance is comparable to that provided by raw disk.

Veritas Volume Manager integration with Veritas File System for enhanced I/O performance

Veritas File System (VxFS) interfaces with Veritas Volume Manager (VxVM) to determine the I/O characteristics of the underlying volume and perform I/O accordingly. VxFS also uses this information when using `mkfs` to perform proper allocation unit alignments for efficient I/O operations from the kernel.

As part of VxFS/VxVM integration, VxVM exports a set of I/O parameters to achieve better I/O performance. This interface can enhance performance for different volume configurations such as RAID-5, striped, and mirrored volumes. Full stripe writes are important in a RAID-5 volume for strong I/O performance. VxFS uses these parameters to issue appropriate I/O requests to VxVM.

Application-specific parameters for enhanced I/O performance

You can set application specific parameters on a per-file system basis to improve I/O performance.

- Discovered Direct I/O
  All sizes above this value would be performed as direct I/O.
- Maximum Direct I/O Size

This value defines the maximum size of a single direct I/O.

See the `vxtunefs`(1M) and `tunefstab`(4) manual pages.

Delayed allocation for extending writes

Delayed allocation skips the allocations for extending writes and completes the allocations in a background thread. With this approach, Veritas File System (VxFS) performs a smaller number of large allocations instead of performing a large number of small allocations, which reduces the file system's fragmentation. Fast-moving temporary files do not have blocks allocated and thus do not add to the file system's fragmentation.

When a file is appended, the allocation to the file is skipped and the file is added to the delayed allocation list. The range for which the allocation is skipped is recorded in the inode. The `write`() system call returns immediately after the user pages are copied to the page cache. The actual allocations to the file occur when the scheduler thread picks the file for allocation. If the file is truncated or removed, allocations are not required.

Delayed allocation is enabled by default for local mount file system but disabled for cluster mount file system. You can enable this feature on cluster mount file system by using the `vxtunefs` command. You can display the delayed allocation range in the file by using the `fsmap` command.

See the `vxtunefs`(1M) and `fsmap`(1M) manual pages.

For instances where the file data must be written to the disk immediately, delayed allocation is disabled on the file. The following are the examples of such instances: direct I/O, concurrent I/O, FDD/ODM access, and synchronous I/O. Delayed allocation is not supported on memory-mapped files and BSD quotas. When BSD quotas are enabled on a file system, delayed allocation is turned off automatically for that file system.

Using Veritas File System

The following list contains the main methods to use, manage, modify, and tune VxFS:

- Online system administration
- Application program interface

Online system administration

Veritas File System (VxFS) provides command line interface (CLI) operations that are described throughout this guide and in manual pages.

VxFS allows you to run a number of administration tasks while the file system is online. Two of the more important tasks include:

- About defragmentation
- About file system resizing

About defragmentation

Free resources are initially aligned and allocated to files in an order that provides optimal performance. On an active file system, the original order of free resources is lost over time as files are created, removed, and resized. The file system is spread farther along the disk, leaving unused gaps or fragments between areas that are in use. This process is known as fragmentation and leads to degraded performance because the file system has fewer options when assigning a free extent to a file (a group of contiguous data blocks).

VxFS provides the online administration utility `fsadm` to resolve the problem of fragmentation.

The `fsadm` utility defragments a mounted file system by performing the following actions:

- Removing unused space from directories
- Making all small files contiguous
- Consolidating free blocks for file system use

This utility can run on demand and should be scheduled regularly as a cron job.

See the `fsadm_vxfs(1M)` manual page.

About file system resizing

A file system is assigned a specific size as soon as it is created; the file system may become too small or too large as changes in file system usage take place over time.

VxFS is capable of increasing or decreasing the file system size while in use. Many competing file systems can not do this. The VxFS utility `fsadm` can expand or shrink a file system without unmounting the file system or interrupting user productivity. However, to expand a file system, the underlying device on which it is mounted must be expandable.

VxVM facilitates expansion using virtual disks that can be increased in size while in use. The VxFS and VxVM components complement each other to provide online expansion capability. Use the `vxresize` command when resizing both the volume and the file system. The `vxresize` command guarantees that the file system shrinks or grows along with the volume. You can also use the the `vxassist` command combined with the `fsadm` command for this purpose; however, Veritas recommends that you use the `vxresize` command instead.

See the `vxresize`(1M) manual page.

See "Growing the existing storage by adding a new LUN" on page 118.

Application program interface

Veritas File System Developer's Kit (SDK) provides developers with the information necessary to use the application programming interfaces (APIs) to modify and tune various features and components of Veritas File System (VxFS).

See the *Veritas File System Programmer's Reference Guide*.

VxFS conforms to the System V Interface Definition (SVID) requirements and supports user access through the Network File System (NFS). Applications that require performance features not available with other file systems can take advantage of VxFS enhancements.

Expanded application facilities

Veritas File System (VxFS) provides API functions frequently associated with commercial applications that make it possible to perform the following actions:

- Preallocate space for a file
- Specify a fixed extent size for a file
- Bypass the system buffer cache for file I/O
- Specify the expected access pattern for a file

Because these functions are provided using VxFS-specific IOCTL system calls, most existing UNIX system applications do not use them. For portability reasons, these applications must check which file system type they are using before using these functions.

Section 2

Provisioning storage

Chapter 5

Provisioning new storage

This chapter includes the following topics:

Provisioning new storage

The following procedure describes how to provision new storage. If you are provisioning Storage Foundation on thin storage, you should understand how Storage Foundation works with thin storage.

See “About thin optimization solutions in Storage Foundation ” on page 469.

The procedure assumes a basic configuration for volumes and file systems. More options are available to customize your configuration.

See “Customizing allocation behavior” on page 121.

See “Creating a VxFS file system” on page 162.

To provision new storage

1 Set up the LUN. See the documentation for your storage array for information about how to create, mask, and bind the LUN.

2 Initialize the LUNs that you want to use with Veritas Volume Manager (VxVM), using one of the following commands.

The recommended method is to use the `vxdisksetup` command.

```
# vxdisksetup -i 3PARDATA0_1
# vxdisk init 3PARDATA0_1
```

3 Add the LUN to a disk group.

- If you do not have a disk group for your LUN, create the disk group:

  ```
  # vxdg init dg1 dev1=3PARDATA0_1
  ```

- If you already have a disk group for your LUN, add the LUN to the disk group:

  ```
  # vxdg -g dg1 adddisk 3PARDATA0_1
  ```

4 Create the volume on the LUN:

```
# vxassist -b -g dg1 make vol1 100g 3PARDATA0_1
```

5 Create a Veritas File System (VxFS) file system on the volume:

```
# mkfs -t vxfs /dev/vx/rdsk/dg1/vol1
```

6 Create a mount point on the file system:

```
# mkdir /mount1
```

7 Mount the file system:

```
# mount -t vxfs /dev/vx/dsk/dg1/vol1 /mount1
```

Growing the existing storage by adding a new LUN

The following procedure describes how to grow the existing storage by adding a new LUN.

To grow the existing storage by adding a new LUN

1 Create and set up the LUN.

2 Add the LUN to the disk group.

```
# vxdg -g dg1 adddisk 3PARDATA0_2
```

3 Grow the volume and the file system to the desired size. For example:

```
# vxresize -b -F vxfs -g dg1 vol1 200g
```

Growing the existing storage by growing the LUN

The following procedure describes how to grow the existing storage by growing a LUN.

To grow the existing storage by growing a LUN

1. Grow the existing LUN. See the documentation for your storage array for information about how to create, mask, and bind the LUN.
2. Make Veritas Volume Manager (VxVM) aware of the new LUN size.

   ```
   # vxdisk -g dg1 resize 3PARDATA0_1
   ```

3. Calculate the new maximum volume size:

   ```
   # vxassist -g dg1 -b maxgrow vol1
   ```

4. Grow the volume and the file system to the desired size:

   ```
   # vxresize -b -F vxfs -g dg1 vol1 200g
   ```

Displaying SF information with vxlist

The vxlist command is a display command that provides a consolidated view of the SF configuration. The vxlist command consolidates information from Veritas Volume Manager (VxVM) and Veritas File System (VxFS). The vxlist command provides various options to display information. For example, use the following form of the command to display file system information including information about the volume, disk group, and so on. In previous releases, you needed to run at least two commands to retrieve the following information.

```
# /opt/VRTSsfmh/bin/vxlist fs

TY FS  FSTYPE SIZE   FREE   %USED DEVICE_PATH          MOUNT_POINT
fs /   ext3   65.20g 51.70g 17%   /dev/sda1                /
fs mnt vxfs   19.84g  9.96g 49%   /dev/vx/dsk/bardg/vol1  /mnt
```

For help on the vxlist command, enter the following command:

```
# vxlist -H
```

See the vxlist(1m) manual page.

Chapter 6

Advanced allocation methods for configuring storage

This chapter includes the following topics:

Customizing allocation behavior

By default, the vxassist command creates volumes on any available storage that meets basic requirements. The vxassist command seeks out available disk space and allocates it in the configuration that conforms to the layout specifications and that offers the best use of free space. The vxassist command creates the required plexes and subdisks using only the basic attributes of the desired volume as input.

If you are provisioning Storage Foundation on thin storage, you should understand how Storage Foundation works with thin storage.

See "About thin optimization solutions in Storage Foundation " on page 469.

Additionally, when you modify existing volumes using the vxassist command, the vxassist command automatically modifies underlying or associated objects. The vxassist command uses default values for many volume attributes, unless you provide specific values to the command line. You can customize the default behavior of the vxassist command by customizing the default values.

See "Setting default values for vxassist" on page 122.

The vxassist command creates volumes in a default disk group according to the default rules. To use a different disk group, specify the -g diskgroup option to the vxassist command.

See "Rules for determining the default disk group" on page 639.

If you want to assign particular characteristics for a certain volume, you can specify additional attributes on the vxassist command line. These can be storage specifications to select certain types of disks for allocation, or other attributes such as the stripe unit width, number of columns in a RAID-5 or stripe volume, number of mirrors, number of logs, and log type.

If you want to create 4K sector-sized volumes by using disks that have 512 byte sectors, use the -Q option of the vxassist command.

For details of available vxassist keywords and attributes, refer to the vxassist(1M) manual page.

You can use allocation attributes to specify the types of allocation behavior shown in Table 6-1

Table 6-1 Types of allocation behavior

Allocation behavior	Procedures
Layouts for the volumes	See "Creating volumes of a specific layout" on page 143.

Table 6-1 Types of allocation behavior *(continued)*

Allocation behavior	Procedures
Media types	See "Creating volumes on specific media types" on page 152.
Specific disks, subdisks, plexes locations	See "Creating a volume on specific disks" on page 151.
Ordered allocation	See "Specifying ordered allocation of storage to volumes" on page 156.
Site-based allocation	See "Site-based allocation" on page 159.
Setting the read policy	See "Changing the read policy for mirrored volumes" on page 159.

The vxassist utility also provides various constructs to help define and manage volume allocations, with efficiency and flexibility.

See "Setting default values for vxassist" on page 122.

See "Using rules to make volume allocation more efficient" on page 124.

See "Understanding persistent attributes" on page 127.

See "Customizing disk classes for allocation" on page 129.

See "Specifying allocation constraints for vxassist operations with the use clause and the require clause" on page 132.

See "Management of the use and require type of persistent attributes " on page 140.

Setting default values for vxassist

The default values that the vxassist command uses may be specified in the file /etc/default/vxassist. The defaults listed in this file take effect if you do not override them on the command line, or in an alternate defaults file that you specify using the -d option. A default value specified on the command line always takes precedence. vxassist also has a set of built-in defaults that it uses if it cannot find a value defined elsewhere.

You must create the /etc/default directory and the vxassist default file if these do not already exist on your system.

The format of entries in a defaults file is a list of attribute-value pairs separated by new lines. These attribute-value pairs are the same as those specified as options on the vxassist command line.

See the `vxassist`(1M) manual page.

To display the default attributes held in the file `/etc/default/vxassist`, use the following form of the `vxassist` command:

```
# vxassist help showattrs
```

The following is a sample `vxassist` defaults file:

```
#  By default:
#  create unmirrored, unstriped volumes
#  allow allocations to span drives
#  with RAID-5 create a log, with mirroring don’t create a log
#  align allocations on cylinder boundaries
    layout=nomirror,nostripe,span,nocontig,raid5log,noregionlog,
    diskalign

#  use the fsgen usage type, except when creating RAID-5 volumes
    usetype=fsgen
#  allow only root access to a volume
     mode=u=rw,g=,o=
     user=root
     group=root

#  when mirroring, create two mirrors
     nmirror=2
#  for regular striping, by default create between 2 and 8 stripe
#  columns
     max_nstripe=8
     min_nstripe=2

#  for RAID-5, by default create between 3 and 8 stripe columns
     max_nraid5stripe=8
     min_nraid5stripe=3

#  by default, create 1 log copy for both mirroring and RAID-5 volumes
    nregionlog=1
    nraid5log=1

#  by default, limit mirroring log lengths to 32Kbytes
    max_regionloglen=32k

#  use 64K as the default stripe unit size for regular volumes
    stripe_stwid=64k
```

```
#  use 16K as the default stripe unit size for RAID-5 volumes
    raid5_stwid=16k
```

Using rules to make volume allocation more efficient

The `vxassist` command lets you create a set of volume allocation rules and define it with a single name. When you specify this name in your volume allocation request, all the attributes that are defined in this rule are honored when `vxassist` creates the volume.

Creating volume allocation rules has the following benefits:

- Rules streamline your typing and reduce errors. You can define relatively complex allocation rules once in a single location and reuse them.
- Rules let you standardize behaviors in your environment, including across a set of servers.

For example, you can create allocation rules so that a set of servers can standardize their storage tiering. Suppose you had the following requirements:

Tier 1	Enclosure mirroring between a specific set of array types
Tier 2	Non-mirrored striping between a specific set of array types
Tier 0	Select solid-state drive (SSD) storage

You can create rules for each volume allocation requirement and name the rules tier1, tier2, and tier0.

You can also define rules so that each time you create a volume for a particular purpose, the volume is created with the same attributes. For example, to create the volume for a production database, you can create a rule called productiondb. To create standardized volumes for home directories, you can create a rule called homedir. To standardize your high performance index volumes, you can create a rule called dbindex.

Rule file format

When you create rules, you do not define them in the `/etc/default/vxassist` file. You create the rules in another file and add the path information to `/etc/default/vxassist`. By default, a rule file is loaded from `/etc/default/vxsf_rules`. You can override this location in `/etc/default/vxassist` with the attribute `rulefile=/`*`path`*`/`*`rule_file_name`*. You can also specify additional rule files on the command line.

A rule file uses the following conventions:

- Blank lines are ignored.
- Use the pound sign, #, to begin a comment.
- Use C language style quoting for the strings that may include embedded spaces, new lines, or tabs. For example, use quotes around the text for the `description` attribute.
- Separate tokens with a space.
- Use braces for a rule that is longer than one line.

Within the rule file, a volume allocation rule has the following format:

```
volume rule rulename vxassist_attributes
```

This syntax defines a rule named *rulename* which is a short-hand for the listed `vxassist` attributes. Rules can reference other rules using an attribute of `rule=`*`rulename`*`[,`*`rulename`*`,...]`, which adds all the attributes from that rule into the rule currently being defined. The attributes you specify in a rule definition override any conflicting attributes that are in a rule that you specify by reference. You can add a description to a rule with the attribute `description=`*`description_text`*.

The following is a basic rule file. The first rule in the file, base, defines the `logtype` and `persist` attributes. The remaining rules in the file – tier0, tier1, and tier2 – reference this rule and also define their own tier-specific attributes. Referencing a rule lets you define attributes in one place and reuse them in other rules.

```
# Create tier 1 volumes mirrored between disk arrays, tier 0 on SSD,
# and tier 2 as unmirrored. Always use FMR DCO objects.
volume rule base { logtype=dco persist=yes }
volume rule tier0 { rule=base mediatype:ssd tier=tier0 }
volume rule tier1 { rule=base mirror=enclosure tier=tier1 }
volume rule tier2 { rule=base tier=tier2 }
```

The following rule file contains a more complex definition that runs across several lines.

```
volume rule appXdb_storage {
    description="Create storage for the database of Application X"
    rule=base
    siteconsistent=yes
    mirror=enclosure
}
```

Using rules to create a volume

When you use the `vxassist` command to create a volume, you can include the rule name on the command line. For example, the content of the `vxsf_rules` file is as follows:

```
volume rule basic { logtype=dco }
volume rule tier1 {
        rule=basic
        layout=mirror
        tier=tier1
}
```

In the following example, when you create the volume `vol1` in disk group `dg3`, you can specify the `tier1` rule on the command line. In addition to the attributes you enter on the command line, `vol1` is given the attributes that you defined in `tier1`.

```
vxassist -g dg3 make vol1 200m rule=tier1
```

The following `vxprint` command displays the attributes of disk group `dg3`. The output includes the new volume, `vol1`.

```
vxprint -g dg3
TY NAME         ASSOC        KSTATE   LENGTH   PLOFFS   STATE    TUTIL0  PUTIL0
dg dg3          dg3          -        -        -        -        -       -

dm ibm_ds8x000_0266 ibm_ds8x000_0266 - 2027264 -        -        -       -
dm ibm_ds8x000_0267 ibm_ds8x000_0267 - 2027264 -        -        -       -
dm ibm_ds8x000_0268 ibm_ds8x000_0268 - 2027264 -        -        -       -

v  vol1         fsgen        ENABLED  409600   -        ACTIVE   -       -
pl vol1-01      vol1         ENABLED  409600   -        ACTIVE   -       -
sd ibm_ds8x000_0266-01 vol1-01 ENABLED 409600  0        -        -       -
pl vol1-02      vol1         ENABLED  409600   -        ACTIVE   -       -
sd ibm_ds8x000_0267-01 vol1-02 ENABLED 409600  0        -        -       -
dc vol1_dco     vol1         -        -        -        -        -       -
v  vol1_dcl     gen          ENABLED  144      -        ACTIVE   -       -
pl vol1_dcl-01  vol1_dcl     ENABLED  144      -        ACTIVE   -       -
sd ibm_ds8x000_0266-02 vol1_dcl-01 ENABLED 144 0        -        -       -
pl vol1_dcl-02  vol1_dcl     ENABLED  144      -        ACTIVE   -       -
sd ibm_ds8x000_0267-02 vol1_dcl-02 ENABLED 144 0        -        -       -
```

The following `vxassist` command confirms that `vol1` is in the tier `tier1`. The application of rule `tier1` was successful.

```
vxassist -g dg3 listtag
TY   NAME              DISKGROUP         TAG
=========================================================
v    vol1              dg3               vxfs.placement_class.tier1
```

Understanding persistent attributes

The `vxassist` command lets you record certain volume allocation attributes for a volume. These attributes are called persistent attributes. You can record the attributes which would be useful in later allocation operations on the volume, such as volume grow and enclosure mirroring. You can also restrict allocation to storage that has a particular property (such as the enclosure type, disk tag, or media type). On the other hand, volume length is not useful, and generally neither is a specific list of disks.

The persistent attributes can be retrieved and applied to the allocation requests (with possible modifications) for the following operations:

- volume grow or shrink
- move
- relayout
- mirror
- add a log

Persistent attributes let you record carefully-described allocation attributes at the time of volume creation and retain them for future allocation operations on the volume. Also, you can modify, enhance, or discard the persistent attributes. For example, you can add and retain a separation rule for a volume that is originally not mirrored. Alternatively, you can temporarily suspend a volume allocation rule which has proven too restrictive or discard it to allow a needed allocation to succeed.

You can use the persist attribute to record allocation attributes on the command line or in a rule file.

See "Using persistent attributes" on page 127.

You can manage the use and require type of persistent attributes with the intent management operations: setrule, changerule, clearrule, and listrule.

See "Management of the use and require type of persistent attributes " on page 140.

Using persistent attributes

You can define volume allocation attributes so they can be reused in subsequent operations. These attributes are called persistent attributes, and they are stored in a set of hidden volume tags. The persist attribute determines whether an attribute persists, and how the current command might use or modify preexisting persisted attributes. You can specify persistence rules in defaults files, in rules, or on the command line. For more information, see the vxassist manual page.

To illustrate how persistent attributes work, we will use the following vxsf_rules file. It contains a rule, rule1, which defines the mediatype attribute. This rule also uses the persist attribute to make the mediatype attribute persistent.

```
# cat /etc/default/vxsf_rules
volume rule rule1 { mediatype:ssd persist=extended }
```

The following command confirms that LUNs ibm_ds8x000_0266 and ibm_ds8x000_0268 are Solid-State Drive (SSD) devices.

```
# vxdisk listtag
DEVICE              NAME                VALUE
ibm_ds8x000_0266    vxmediatype         ssd
ibm_ds8x000_0268    vxmediatype         ssd
```

The following command creates a volume, vol1, in the disk group dg3. rule1 is specified on the command line, so those attributes are also applied to vol1.

```
# vxassist -g dg3 make vol1 100m rule=rule1
```

The following command shows that the volume vol1 is created off the SSD device ibm_ds8x000_0266 as specified in rule1.

```
# vxprint -g dg3
TY NAME         ASSOC        KSTATE   LENGTH   PLOFFS   STATE    TUTIL0  PUTIL0
dg dg3          dg3          -        -        -        -        -       -

dm ibm_ds8x000_0266 ibm_ds8x000_0266 - 2027264 -        -        -       -
dm ibm_ds8x000_0267 ibm_ds8x000_0267 - 2027264 -        -        -       -
dm ibm_ds8x000_0268 ibm_ds8x000_0268 - 2027264 -        -        -       -

v  vol1         fsgen        ENABLED  204800   -        ACTIVE   -       -
pl vol1-01      vol1         ENABLED  204800   -        ACTIVE   -       -
sd ibm_ds8x000_0266-01 vol1-01 ENABLED 204800  0        -        -       -
```

The following command displays the attributes that are defined in rule1.

```
# vxassist -g dg3 help showattrs rule=rule1
alloc=mediatype:ssd
persist=extended
```

If no persistent attributes are defined, the following command grows vol1 on Hard Disk Drive (HDD) devices. However, at the beginning of this section, mediatype:ssd was defined as a persistent attribute. Therefore, the following command honors this original intent and grows the volume on SSD devices.

```
# vxassist -g dg3 growby vol1 1g
```

The following vxprint command confirms that the volume was grown on SSD devices.

```
# vxprint -g dg3
TY NAME          ASSOC         KSTATE    LENGTH    PLOFFS    STATE     TUTIL0   PUTIL0
dg dg3           dg3           -         -         -         -         -        -

dm ibm_ds8x000_0266 ibm_ds8x000_0266 - 2027264 -             -         -        -
dm ibm_ds8x000_0267 ibm_ds8x000_0267 - 2027264 -             -         -        -
dm ibm_ds8x000_0268 ibm_ds8x000_0268 - 2027264 -             -         -        -

v  vol1          fsgen         ENABLED  2301952    -         ACTIVE    -        -
pl vol1-01       vol1          ENABLED  2301952    -         ACTIVE    -        -
sd ibm_ds8x000_0266-01 vol1-01 ENABLED 2027264 0             -         -        -
sd ibm_ds8x000_0268-01 vol1-01 ENABLED 274688  2027264 -               -        -
```

Customizing disk classes for allocation

The vxassist command accepts disk classes to indicate storage specifications for allocation. The disk classes are internally-discovered attributes that are automatically associated with the disks. You can specify disk classes to an allocation request with vxassist to indicate the type of storage to allocate.

For more information about the disk classes, see the Storage Specifications section of the vxassist(1M) manual page.

You can customize the disk classes in the following ways:

- Create a customized alias name.
  See "User-defined alias names for disk classes" on page 130.
- Customize the priority order for the disk classes.
  See "User-defined precedence order for disk classes" on page 130.

You can also create customized, user-defined disk classes.

See "User-defined disk classes" on page 131.

User-defined alias names for disk classes

For convenience, you can define alias names for existing storage-specification disk classes. Typically, an alias is a shorter or more user-friendly name. You can use the alias instead of its corresponding disk class, to specify `vxassist` allocation constraints. Define the alias names in rule files.

For example, to define “atyp” as an alias for the base disk class “arraytype”, include the following statement in a rule file.

```
class alias atyp=arraytype
```

When the above rule file is used, you can specify the alias “atyp” for allocation. For example, the following constraint specification allocates storage from A/A arrays for the volume creation.

```
# vxassist -g dgname make volname volsize use=atyp:A/A
```

User-defined precedence order for disk classes

The `vxassist` command applies a default priority order for the disk classes that are specified in the mirror confinement (`mirrorconfine`, `wantmirrorconfine`), mirror separation (`mirror`, `wantmirror`), and stripe separation (`stripe`, `wantstripe`) constraints. The higher priority class is honored for allocation when mirroring or striping. If a different priority order is required, you can change the default order for these disk classes.

Note: The “site” class always has the highest precedence, and its order cannot be overridden.

Define the customized precedence order in a rule file. The higher the order number, the higher is the class precedence.

The following shows the default precedence order, for the class names supported with mirror and stripe separation or confinement constraints.

site	order=1000
vendor	order=900
arrayproduct	order=800
array	order=700
arrayport	order=600
hostport	order=400

The acceptable range for the precedence order is between 0 and 1000.

For example, the array class has a higher priority than the hostport class by default. To make the hostport class have a higher priority, assign the hostport class a higher order number. To define the order for the classes, include the following statement in a rule file:

```
class define array order=400
class define hostport order=700
```

When the above rule is used, the following command mirrors across hostport class rather than the array class.

```
# vxassist -g dgname make volname volsize mirror=array,hostport
```

User-defined disk classes

You can define customized disk classes to use in storage specifications for the `vxassist` command. Customized disk classes allow for user-defined device classification and grouping. You can use these disk classes to control allocations. A customized disk class is a user-defined property that is associated with a set of disks. The property is attached as a disk class to the disks that satisfy a particular constraint.

You can use the custom disk classes like other storage-specification disk classes, to specify `vxassist` allocation constraints. Define the custom disk classes in a rule file.

Example

With the following definition in the rule file, the user-defined property “poolname” is associated to the referenced disks. All devices that have the array vendor property defined as HITACHI or IBM, are marked as poolname “finance”. All devices that have the array vendor property defined as DGC or EMC, are marked as poolname “admin”.

```
disk properties vendor:HITACHI {
     poolname:finance
}
disk properties vendor:IBM {
     poolname:finance
}
disk properties vendor:DGC {
     poolname:admin
}
disk properties vendor:EMC {
     poolname:admin
}
```

You can now use the user-defined disk class “poolname”for allocation. For example, the following constraint specification allocates disks from the poolname “admin” for the volume creation.

```
# vxassist -g dgname make volname volsize poolname:admin
```

Specifying allocation constraints for vxassist operations with the use clause and the require clause

The `vxassist` command accepts a variety of storage specifications for allocations. The require constraint and the use constraint are methods to specify detailed storage specifications for allocations. These constraints enable you to select disks from an intersection set or a union set of intended properties. You can specify the set of disks for allocations with more precision than the existing methods `alloc` and `logdisk` clauses. The use and require constraints can apply to data, log, or both data and log.

The constraints can be one of the following types:

- The require constraints
  All of the specifications in the constraint must be satisfied, or the allocation fails. A require constraint behaves as an intersection set. For example, allocate disks from a particular array vendor AND with a particular array type.
- The use constraints
  At least one of the specifications in the constraint must be satisfied, or the allocation fails. A use constraint behaves as a union set. For example, allocate disks from any of the specified enclosures: enclrA or enclrB.

For disk group version of 180 or above, the use and require type of constraints are persistent for the volume by default. The default preservation of these clauses enables further allocation operations like grow, without breaking the specified intents.

You can specify multiple storage specifications, separated by commas, in a use or require clause on the `vxassist` command line. You can also specify multiple use or require clauses on the `vxassist`command line.

See “Interaction of multiple require and use constraints” on page 134.

Use the `vxassist` intent management operations (`setrule`, `changerule`, `clearrule`, `listrule`) to manage persistent require and use constraints.

See “Management of the use and require type of persistent attributes ” on page 140.

About require constraints

The “require” type of constraints specify that the allocation must select storage that matches all the storage specifications in the constraint. Therefore, the require constraint acts like an intersection set, or a logical AND operation. If any of the specifications cannot be met, the operation fails. The attribute names to specify require constraints are:

- `require`

  The constraint applies to both data and log allocation.

- `logrequire`

  The constraint applies to log allocations only.

- `datarequire`

  The constraint applies to data allocations only.

If any storage-specification is negated with !, the allocation excludes the storage that matches that storage specification

---

Note: If the require type of constraint is provided with the same class but different instances, then these instances are unionized rather than intersected. That is, the allocation selects storage that satisfies any of these storage specifications (similar to use type of constraint).

See “Interaction of multiple require and use constraints” on page 134.

About use constraints

The “use” type of constraints specify that the allocation must select storage that matches at least one of the storage specifications in the constraint. Therefore, the use constraint acts like a union set, or a logical OR operation. If none of the specifications can be met, the operation fails. The attribute names to specify use constraints are:

- `use`

  The constraint applies to both data and log allocation.

- `loguse`

  The constraint applies to log allocations only.

- `datause`

  The constraint applies to data allocations only.

See “Interaction of multiple require and use constraints” on page 134.

If the storage specification is negated with !, then the allocation excludes the storage that matches that storage specification.

Interaction of multiple require and use constraints

You can specify multiple use or require clauses on the `vxassist` command line. Not all combinations are supported. However, all possible constraint specifications can be achieved with the supported combinations.

The scope for a constraint can be data-specific (datause or datarequire), log-specific (loguse or logrequire) or general, which applies to both data and log (use or require).

Note: Veritas recommends that you do not combine use or require constraints with direct storage-specifications or other clauses like `alloc` or `logdisk`.

The following rules apply when multiple use or require clauses are specified:

- Multiple use constraints of the same scope are unionized, so that at least one of the storage specifications is satisfied. That is, multiple `use` clauses; multiple `datause` clauses; or multiple `loguse` clauses.
- Multiple require constraints of the same scope are intersected, so that all the storage specifications are satisfied. That is, multiple `require` clauses; multiple `datarequire` clauses; or multiple `logrequire` clauses.
- Require and use constraints of the same scope are mutually intersected. That is, `require` clauses and `use` clauses; `datarequire` clauses and `datause` clauses; or `logrequire` clauses and `loguse` clauses. At least one of the use storage specifications must be satisfied and all of the require storage specifications are satisfied. For example, if a `datause` clause and a `datarequire` clause are used together, the allocation for the data must meet at least one of the `datause` specifications and all of the `datarequire` specifications.
- Data-specific constraints and log-specific constraints can be used together. They are applied independently to data and logs respectively. That is, `datause` clause with `loguse` clause or `logrequire` clause; `datarequire` clause with `loguse` clause or `logrequire` clause . For example, a `datarequire` clause can be used to control data allocation along with a `logrequire` clause to control log allocation.
- The `vxassist` command does not support a mix of general scope constraints with data-specific or log-specific constraints. For example, a `require` clause cannot be used along with the `logrequire` clause or a `datarequire` clause. However, all possible constraint specifications can be achieved with the supported combinations.

Table 6-2 summarizes these rules for the interaction of each type of constraint if multiple constraints are specified.

Table 6-2 Combinations of require and use constraints

Scope	Mutually unionized	Mutually intersected	Applied independently
Data	datause - datause	datarequire - datause datarequire - datarequire	datause - loguse datause - logrequire datarequire - loguse datarequire - logrequire
Log	loguse - loguse	logrequire - loguse logrequire - logrequire	loguse - datause loguse - datarequire logrequire -datause logrequire - datarequire
General - log and data	use - use	use - require require - require	N/A

Examples of use and require constraints

The following examples show use and require constraints for storage allocation.

Example 1 - require constraint

This example shows the require constraint in a disk group that has disks from two arrays: `emc_clariion0` and `ams_wms0`. Both arrays are connected through the same HBA hostportid (06-08-02), but the arrays have different arraytype (A/A and A/A-A respectively).

The following output shows the disk group information:

```
# vxprint -g testdg
TY NAME         ASSOC        KSTATE   LENGTH   PLOFFS   STATE    TUTIL0  PUTIL0
dg testdg       testdg       -        -        -        -        -       -

dm ams_wms0_359 ams_wms0_359 -        2027264  -        -        -       -
dm ams_wms0_360 ams_wms0_360 -        2027264  -        -        -       -
dm ams_wms0_361 ams_wms0_361 -        2027264  -        -        -       -
dm ams_wms0_362 ams_wms0_362 -        2027264  -        -        -       -
dm emc_clariion0_0 emc_clariion0_0 -  4120320  -        -        -       -
dm emc_clariion0_1 emc_clariion0_1 -  4120320  -        -        -       -
dm emc_clariion0_2 emc_clariion0_2 -  4120320  -        -        -       -
dm emc_clariion0_3 emc_clariion0_3 -  4120320  -        -        -       -
```

To allocate both the data and the log on the disks that are attached to the particular HBA and that have the array type A/A:

```
# vxassist -g testdg make v1 1G logtype=dco dcoversion=20 \
require=hostportid:06-08-02,arraytype:A/A
```

The following output shows the results of the above command. The command allocated disk space for the data and the log on `emc_clariion0` array disks, which satisfy all the storage specifications in the `require` constraint:

```
# vxprint -g testdg
TY NAME         ASSOC         KSTATE   LENGTH   PLOFFS   STATE    TUTIL0  PUTIL0
dg testdg       testdg        -        -        -        -        -       -

dm ams_wms0_359 ams_wms0_359 -        2027264  -        -        -       -
dm ams_wms0_360 ams_wms0_360 -        2027264  -        -        -       -
dm ams_wms0_361 ams_wms0_361 -        2027264  -        -        -       -
dm ams_wms0_362 ams_wms0_362 -        2027264  -        -        -       -
dm emc_clariion0_0 emc_clariion0_0 -  4120320  -        -        -       -
dm emc_clariion0_1 emc_clariion0_1 -  4120320  -        -        -       -
dm emc_clariion0_2 emc_clariion0_2 -  4120320  -        -        -       -
dm emc_clariion0_3 emc_clariion0_3 -  4120320  -        -        -       -

v  v1           fsgen         ENABLED  2097152  -        ACTIVE   -       -
pl v1-01        v1            ENABLED  2097152  -        ACTIVE   -       -
sd emc_clariion0_0-01 v1-01   ENABLED  2097152  0        -        -       -
dc v1_dco       v1            -        -        -        -        -       -
v  v1_dcl       gen           ENABLED  67840    -        ACTIVE   -       -
pl v1_dcl-01    v1_dcl        ENABLED  67840    -        ACTIVE   -       -
sd emc_clariion0_0-02 v1_dcl-01 ENABLED 67840   0        -        -       -
```

Example 2 - use constraint

This example shows the use constraint in a disk group that has disks from three arrays: `ams_wms0`, `emc_clariion0`, and `hitachi_vsp0`.

The following output shows the disk group information:

```
# vxprint -g testdg
TY NAME         ASSOC         KSTATE   LENGTH   PLOFFS   STATE    TUTIL0  PUTIL0
dg testdg       testdg        -        -        -        -        -       -

dm ams_wms0_359 ams_wms0_359 -        2027264  -        -        -       -
dm ams_wms0_360 ams_wms0_360 -        2027264  -        -        -       -
dm ams_wms0_361 ams_wms0_361 -        2027264  -        -        -       -
dm ams_wms0_362 ams_wms0_362 -        2027264  -        -        -       -
dm emc_clariion0_0 emc_clariion0_0 -  4120320  -        -        -       -
dm hitachi_vsp0_3 hitachi_vsp0_3 -    4120320  -        -        -       -
```

To allocate both the data and the log on the disks that belong to the array `ams_wms0` or the array `emc_clariion0`:

```
# vxassist -g testdg make v1 3G logtype=dco dcoversion=20 \
use=array:ams_wms0,array:emc_clariion0
```

The following output shows the results of the above command. The command allocated disk space for the data and the log on disks that satisfy the arrays specified in the `use` constraint.

```
# vxprint -g testdg
TY NAME         ASSOC        KSTATE   LENGTH   PLOFFS   STATE    TUTIL0  PUTIL0
dg testdg       testdg       -        -        -        -        -       -

dm ams_wms0_359 ams_wms0_359 -        2027264  -        -        -       -
dm ams_wms0_360 ams_wms0_360 -        2027264  -        -        -       -
dm ams_wms0_361 ams_wms0_361 -        2027264  -        -        -       -
dm ams_wms0_362 ams_wms0_362 -        2027264  -        -        -       -
dm emc_clariion0_0 emc_clariion0_0 -  4120320  -        -        -       -
dm hitachi_vsp0_3 hitachi_vsp0_3 -    4120320  -        -        -       -

v  v1           fsgen        ENABLED  6291456  -        ACTIVE   -       -
pl v1-01        v1           ENABLED  6291456  -        ACTIVE   -       -
sd ams_wms0_359-01 v1-01     ENABLED  2027264  0        -        -       -
sd ams_wms0_360-01 v1-01     ENABLED  143872   2027264  -        -       -
sd emc_clariion0_0-01 v1-01  ENABLED  4120320  2171136  -        -       -
dc v1_dco       v1           -        -        -        -        -       -
v  v1_dcl       gen          ENABLED  67840    -        ACTIVE   -       -
pl v1_dcl-01    v1_dcl       ENABLED  67840    -        ACTIVE   -       -
sd ams_wms0_360-02 v1_dcl-01 ENABLED  67840    0        -        -       -
```

Example 3: datause and logrequire combination

This example shows the combination of a `datause` constraint and a `logrequire` constraint. The disk group has disks from three arrays: `ams_wms0`, `emc_clariion0`, and `hitachi_vsp0`, which have different array types.

The following output shows the disk group information:

```
# vxprint -g testdg
 TY NAME          ASSOC         KSTATE   LENGTH   PLOFFS   STATE    TUTIL0  PUTIL0
 dg testdg        testdg        -        -        -        -        -       -
dm ams_wms0_359 ams_wms0_359 -        2027264  -        -        -       -
dm ams_wms0_360 ams_wms0_360 -        2027264  -        -        -       -
dm ams_wms0_361 ams_wms0_361 -        2027264  -        -        -       -
dm ams_wms0_362 ams_wms0_362 -        2027264  -        -        -       -
dm emc_clariion0_0 emc_clariion0_0 -  4120320  -        -        -       -
dm emc_clariion0_1 emc_clariion0_1 -  4120320  -        -        -       -
dm emc_clariion0_2 emc_clariion0_2 -  4120320  -        -        -       -
dm emc_clariion0_3 emc_clariion0_3 -  4120320  -        -        -       -
dm hitachi_vsp0_3 hitachi_vsp0_3 -    4120320  -        -        -       -
```

To allocate data on disks from `ams_wms0` or `emc_clariion0` array, and to allocate log on disks from arraytype A/A-A:

```
# vxassist -g testdg make v1 1G logtype=dco dcoversion=20 \
datause=array:ams_wms0,array:emc_clariion0 logrequire=arraytype:A/A-A
```

The following output shows the results of the above command. The command allocated disk space for the data and the log independently. The data space is allocated on `emc_clariion0` disks that satisfy the `datause` constraint. The log space is allocated on `ams_wms0` disks that are A/A-A arraytype and that satisfy the `logrequire` constraint:

```
# vxprint -g testdg
TY NAME          ASSOC         KSTATE   LENGTH   PLOFFS   STATE    TUTIL0  PUTIL0
dg testdg        testdg        -        -        -        -        -       -
dm ams_wms0_359 ams_wms0_359 -        2027264  -        -        -       -
dm ams_wms0_360 ams_wms0_360 -        2027264  -        -        -       -
dm ams_wms0_361 ams_wms0_361 -        2027264  -        -        -       -
dm ams_wms0_362 ams_wms0_362 -        2027264  -        -        -       -
dm emc_clariion0_0 emc_clariion0_0 -  4120320  -        -        -       -
dm emc_clariion0_1 emc_clariion0_1 -  4120320  -        -        -       -
dm emc_clariion0_2 emc_clariion0_2 -  4120320  -        -        -       -
dm emc_clariion0_3 emc_clariion0_3 -  4120320  -        -        -       -
dm hitachi_vsp0_3 hitachi_vsp0_3 -    4120320  -        -        -       -

v  v1            fsgen         ENABLED  2097152  -        ACTIVE   -       -
pl v1-01         v1            ENABLED  2097152  -        ACTIVE   -       -
sd emc_clariion0_0-01 v1-01  ENABLED  2097152  0        -        -       -
dc v1_dco        v1            -        -        -        -        -       -
v  v1_dcl        gen           ENABLED  67840    -        ACTIVE   -       -
```

```
pl v1_dcl-01    v1_dcl       ENABLED  67840    -        ACTIVE   -       -
sd ams_wms0_359-01 v1_dcl-01 ENABLED  67840    0        -        -       -
```

Example 4 - use and require combination

This example shows the combination of a `use` constraint and a `require` constraint. The disk group has disks from three arrays: `ams_wms0`, `emc_clariion0`, and `hitachi_vsp0`. Only the disks from `ams_wms0` array are multi-pathed.

The following output shows the disk group information:

```
# vxprint -g testdg
TY NAME         ASSOC        KSTATE   LENGTH   PLOFFS   STATE    TUTIL0  PUTIL0
dg testdg       testdg       -        -        -        -        -       -
dm ams_wms0_359 ams_wms0_359 -        2027264  -        -        -       -
dm ams_wms0_360 ams_wms0_360 -        2027264  -        -        -       -
dm ams_wms0_361 ams_wms0_361 -        2027264  -        -        -       -
dm ams_wms0_362 ams_wms0_362 -        2027264  -        -        -       -
dm emc_clariion0_0 emc_clariion0_0 -  4120320  -        -        -       -
dm emc_clariion0_1 emc_clariion0_1 -  4120320  -        -        -       -
dm emc_clariion0_2 emc_clariion0_2 -  4120320  -        -        -       -
dm emc_clariion0_3 emc_clariion0_3 -  4120320  -        -        -       -
dm hitachi_vsp0_3 hitachi_vsp0_3 -    4120320  -        -        -       -
```

To allocate data and log space on disks from `emc_clariion0` or `ams_wms0` array, and disks that are multi-pathed:

```
# vxassist -g testdg make v1 1G logtype=dco dcoversion=20 \
use=array:emc_clariion0,array:ams_wms0 require=multipathed:yes
```

The following output shows the results of the allocation. The data and log space is on `ams_wms0` disks, which satisfy the `use` as well as the `require` constraints:

```
# vxprint -g testdg
TY NAME         ASSOC        KSTATE   LENGTH   PLOFFS   STATE    TUTIL0  PUTIL0
dg testdg       testdg       -        -        -        -        -       -
dm ams_wms0_359 ams_wms0_359 -        2027264  -        -        -       -
dm ams_wms0_360 ams_wms0_360 -        2027264  -        -        -       -
dm ams_wms0_361 ams_wms0_361 -        2027264  -        -        -       -
dm ams_wms0_362 ams_wms0_362 -        2027264  -        -        -       -
dm emc_clariion0_0 emc_clariion0_0 -  4120320  -        -        -       -
dm emc_clariion0_1 emc_clariion0_1 -  4120320  -        -        -       -
dm emc_clariion0_2 emc_clariion0_2 -  4120320  -        -        -       -
dm emc_clariion0_3 emc_clariion0_3 -  4120320  -        -        -       -
dm hitachi_vsp0_3 hitachi_vsp0_3 -    4120320  -        -        -       -
v  v1           fsgen        ENABLED  2097152  -        ACTIVE   -       -
```

```
pl v1-01        v1           ENABLED  2097152  -        ACTIVE   -       -
sd ams_wms0_359-01 v1-01     ENABLED  2027264  0        -        -       -
sd ams_wms0_360-01 v1-01     ENABLED  69888    2027264  -        -       -
dc v1_dco       v1           -        -        -        -        -       -
v  v1_dcl       gen          ENABLED  67840    -        ACTIVE   -       -
pl v1_dcl-01    v1_dcl       ENABLED  67840    -        ACTIVE   -       -
sd ams_wms0_360-02 v1_dcl-01 ENABLED  67840    0        -        -       -
```

Management of the use and require type of persistent attributes

Persistent attributes are the saved volume intents that should be honored for subsequent allocation operations for that volume. The intent management operations enable you to manage the use and require type of persistent intents for volumes. These operations allow you to independently manage the intents after the volume creation. When you change the persistent intents for a volume, the changed intents are not checked for validity or enforced for the current allocation of the volume.

You can set, change, clear, or list the persistent intents for the volume with the following `vxassist` operations:

- `setrule`

  Replaces any existing saved intents with the specified intents for the specified volume.

- `changerule`

  Appends the specified intents to the existing saved intents for the specified volume.

- `clearrule`

  Removes any existing saved intents for the specified volume.

- `listrule`

  Lists any saved intents for the specified volume. If no volume name is specified, the command shows the intents for all of the volumes.

The intent management operations only apply to the use or require type of persistent constraints. The other type of persistent constraints are managed with the `persist` attribute.

See “Using persistent attributes” on page 127.

To display the intents that are currently associated to a volume

- To display the intents that are currently associated to a volume, use the following command:

  ```
  # vxassist [options] listrule [volume]
  ```

  For example, to display the existing saved intents for the volume v1:

  ```
  # vxassist -g testdg listrule v1
  volume rule v1 {
       require=array:ams_wms0
  }
  ```

To replace the intents that are currently associated to a volume

1. Display the intents that are currently associated to the volume:

   ```
   # vxassist [options] listrule [volume]
   ```

   In this example, the volume v1 has an existing saved intent that requires the array to be ams_wms0. For example, to display the existing saved intents for the volume v1:

   ```
   # vxassist -g testdg listrule v1
   volume rule v1 {
        require=array:ams_wms0
   }
   ```

2. Specify the new intent with the following command:

   ```
   # vxassist [options] setrule volume attributes...
   ```

   For example, to replace the array with the ds4100-0 array, specify the new intent with the following command:

   ```
   # vxassist -g testdg setrule v1 require=array:ds4100-0
   ```

3. Verify the new intent with the display command.

   For example, the following command shows that the intent has changed:

   ```
   # vxassist -g testdg listrule v1
   volume rule v1 {
        require=array:ds4100-0
   }
   ```

To add to the intents that are currently associated to a volume

1. Display the intents that are currently associated to the volume:

   ```
   # vxassist [options] listrule [volume]
   ```

   In this example, the volume v1 has an existing saved intent that requires the array to be ds4100-0. For example, to display the existing saved intents for the volume v1:

   ```
   # vxassist -g testdg listrule v1
   volume rule v1 {
        use=array:ds4100-0
   }
   ```

2. Add the new intent with the following command:

   ```
   # vxassist [options] changerule volume attributes...
   ```

   For example, to add the ams_wms0 array in the use constraint, specify the new intent with the following command:

   ```
   # vxassist -g testdg changerule v1 use=array:ams_wms0
   ```

3. Verify the new intent with the display command.

   For example, the following command shows that the intent has changed:

   ```
   # vxassist -g testdg listrule v1
   volume rule v1 {
        use=array:ds4100-0,array:ams_wms0
   }
   ```

To clear the intents that are currently associated to a volume

1 Display the intents that are currently associated to the volume:

```
# vxassist [options] listrule [volume]
```

For example, to display the existing saved intents for the volume v1:

```
# vxassist -g testdg listrule v1
volume rule v1 {
     require=multipathed:yes
     use=array:emc_clariion0,array:ams_wms0
}
```

2 Clear the existing intents with the following command:

```
# vxassist [options] clearrule volume
```

For example, to clear the intents for the volume v1:

```
# vxassist -g testdg clearrule v1
```

3 Verify that the volume has no saved intents.

For example, the following command shows that the volume v1 has no saved intents:

```
# vxassist -g testdg listrule v1
volume rule v1 {}
```

Creating volumes of a specific layout

Veritas Volume Manager (VxVM) enables you to create volumes of various layouts. You can specify an attribute to indicate the type of layout you want to create. The following sections include details for each of the following types:

- mirrored volumes
  See “Creating a mirrored volume” on page 145.
- striped volumes
  See “Creating a striped volume” on page 147.
- RAID-5 volumes
  See “Creating a RAID-5 volume” on page 149.

Types of volume layouts

Veritas Volume Manager (VxVM) allows you to create volumes with several layout types. Table 6-3 describes the layout types for VxVM volumes.

Table 6-3 Types of volume layout

Layout type	Description
Concatenated	A volume whose subdisks are arranged both sequentially and contiguously within a plex. Concatenation allows a volume to be created from multiple regions of one or more disks if there is not enough space for an entire volume on a single region of a disk. If a single LUN or disk is split into multiple subdisks, and each subdisk belongs to a unique volume, this is called carving. See "Concatenation, spanning, and carving" on page 56.
Striped	A volume with data spread evenly across multiple disks. Stripes are equal-sized fragments that are allocated alternately and evenly to the subdisks of a single plex. There must be at least two subdisks in a striped plex, each of which must exist on a different disk. Throughput increases with the number of disks across which a plex is striped. Striping helps to balance I/O load in cases where high traffic areas exist on certain subdisks. See "Striping (RAID-0)" on page 58.
Mirrored	A volume with multiple data plexes that duplicate the information contained in a volume. Although a volume can have a single data plex, at least two are required for true mirroring to provide redundancy of data. For the redundancy to be useful, each of these data plexes should contain disk space from different disks. See "Mirroring (RAID-1)" on page 61.
RAID-5	A volume that uses striping to spread data and parity evenly across multiple disks in an array. Each stripe contains a parity stripe unit and data stripe units. Parity can be used to reconstruct data if one of the disks fails. In comparison to the performance of striped volumes, write throughput of RAID-5 volumes decreases since parity information needs to be updated each time data is modified. However, in comparison to mirroring, the use of parity to implement data redundancy reduces the amount of space required. See "RAID-5 (striping with parity)" on page 64.

Table 6-3 Types of volume layout *(continued)*

Layout type	Description
Mirrored-stripe	A volume that is configured as a striped plex and another plex that mirrors the striped one. This requires at least two disks for striping and one or more other disks for mirroring (depending on whether the plex is simple or striped). The advantages of this layout are increased performance by spreading data across multiple disks and redundancy of data. See "Striping plus mirroring (mirrored-stripe or RAID-0+1)" on page 62.
Layered Volume	A volume constructed from other volumes. Non-layered volumes are constructed by mapping their subdisks to VxVM disks. Layered volumes are constructed by mapping their subdisks to underlying volumes (known as storage volumes), and allow the creation of more complex forms of logical layout. See "About layered volumes" on page 69. The following layouts are examples of layered volumes: ■ Striped-mirror A striped-mirror volume is created by configuring several mirrored volumes as the columns of a striped volume. This layout offers the same benefits as a non-layered mirrored-stripe volume. In addition, it provides faster recovery as the failure of single disk does not force an entire striped plex offline. See "Mirroring plus striping (striped-mirror, RAID-1+0, or RAID-10)" on page 63. ■ Concatenated-mirror A concatenated-mirror volume is created by concatenating several mirrored volumes. This provides faster recovery as the failure of a single disk does not force the entire mirror offline.

Creating a mirrored volume

A mirrored volume provides data redundancy by containing more than one copy of its data. Each copy (or mirror) is stored on different disks from the original copy of the volume and from other mirrors. Mirroring a volume ensures that its data is not lost if a disk in one of its component mirrors fails.

A mirrored volume requires space to be available on at least as many disks in the disk group as the number of mirrors in the volume.

If you specify `layout=mirror`, `vxassist` determines the best layout for the mirrored volume. Because the advantages of the layouts are related to the size of the volume, vxassist selects the layout based on the size of the volume. For smaller volumes, vxassist uses the simpler mirrored concatenated (mirror-concat) layout. For larger volumes, vxassist uses the more complex concatenated mirror (concat-mirror) layout. The attribute stripe-mirror-col-split-trigger-pt controls the selection. Volumes that are smaller than stripe-mirror-col-split-trigger-pt are created as mirror-concat, and volumes that are larger are created as concat-mirror. By default, the attribute stripe-mirror-col-split-trigger-pt is set to one gigabyte. The value can be set in /etc/default/vxassist. If there is a reason to implement a particular layout, you can specify layout=mirror-concat or layout=concat-mirror to implement the desired layout.

To create a new mirrored volume

- Create a new mirrored volume, using the following command:

  ```
  # vxassist [-b] [-g diskgroup] make volume length \
    layout=mirror [nmirror=number] [init=active]
  ```

  Specify the -b option if you want to make the volume immediately available for use.

  For example, to create the mirrored volume, volmir, in the disk group, mydg, use the following command:

  ```
  # vxassist -b -g mydg make volmir 5g layout=mirror
  ```

  The following example shows how to create a volume with 3 mirrors instead of the default of 2 mirrors:

  ```
  # vxassist -b -g mydg make volmir 5g layout=mirror nmirror=3
  ```

Creating a mirrored-concatenated volume

A mirrored-concatenated volume mirrors several concatenated plexes.

To create a mirrored-concatenated volume

- Create the volume as a mirrored-concatenated volume, using the following command:

  ```
  # vxassist [-b] [-g diskgroup] make volume length \
    layout=mirror-concat [nmirror=number]
  ```

  Specify the -b option if you want to make the volume immediately available for use.

Alternatively, first create a concatenated volume, and then mirror it.

See "Adding a mirror to a volume " on page 667.

Creating a concatenated-mirror volume

A concatenated-mirror volume is an example of a layered volume which concatenates several underlying mirror volumes.

To create a concatenated-mirror volume

- Create a concatenated-mirror volume, using the following command:

  ```
  # vxassist [-b] [-g diskgroup] make volume length \
    layout=concat-mirror [nmirror=number]
  ```

  Specify the `-b` option if you want to make the volume immediately available for use.

Creating a striped volume

A striped volume contains at least one plex that consists of two or more subdisks located on two or more physical disks. A striped volume requires space to be available on at least as many disks in the disk group as the number of columns in the volume.

See "Striping (RAID-0)" on page 58.

To create a striped volume, use the following command:

```
# vxassist [-b] [-g diskgroup] make volume length layout=stripe
```

Specify the `-b` option if you want to make the volume immediately available for use.

For example, to create the 10-gigabyte striped volume `volzebra`, in the disk group, `mydg`, use the following command:

```
# vxassist -b -g mydg make volzebra 10g layout=stripe
```

This creates a striped volume with the default stripe unit size (64 kilobytes) and the default number of stripes (2).

You can specify the disks on which the volumes are to be created by including the disk names on the command line. For example, to create a 30-gigabyte striped volume on three specific disks, `mydg03`, `mydg04`, and `mydg05`, use the following command:

```
# vxassist -b -g mydg make stripevol 30g layout=stripe \
  mydg03 mydg04 mydg05
```

To change the number of columns or the stripe width, use the `ncolumn` and `stripeunit` modifiers with `vxassist`. For example, the following command creates a striped volume with 5 columns and a 32-kilobyte stripe size:

```
# vxassist -b -g mydg make stripevol 30g layout=stripe \
  stripeunit=32k ncol=5
```

Creating a mirrored-stripe volume

A mirrored-stripe volume mirrors several striped data plexes. A mirrored-stripe volume requires space to be available on at least as many disks in the disk group as the number of mirrors multiplied by the number of columns in the volume.

To create a mirrored-stripe volume, use the following command:

```
# vxassist [-b] [-g diskgroup] make volume length \
  layout=mirror-stripe [nmirror=number_of_mirrors] \
  [ncol=number_of_columns] [stripewidth=size]
```

Specify the `-b` option if you want to make the volume immediately available for use.

Alternatively, first create a striped volume, and then mirror it. In this case, the additional data plexes may be either striped or concatenated.

See "Adding a mirror to a volume " on page 667.

Creating a striped-mirror volume

A striped-mirror volume is an example of a layered volume that stripes several underlying mirror volumes. A striped-mirror volume requires space to be available on at least as many disks in the disk group as the number of columns multiplied by the number of mirrors in the volume.

To create a striped-mirror volume, use the following command:

```
# vxassist [-b] [-g diskgroup] make volume length \
  layout=stripe-mirror [nmirror=number_of_mirrors] \
  [ncol=number_of_columns] [stripewidth=size]
```

Specify the `-b` option if you want to make the volume immediately available for use.

By default, Veritas Volume Manager (VxVM) attempts to create the underlying volumes by mirroring subdisks rather than columns if the size of each column is greater than the value for the attribute `stripe-mirror-col-split-trigger-pt` that is defined in the `vxassist` defaults file.

If there are multiple subdisks per column, you can choose to mirror each subdisk individually instead of each column. To mirror at the subdisk level, specify the layout as `stripe-mirror-sd` rather than `stripe-mirror`. To mirror at the column level, specify the layout as `stripe-mirror-col` rather than `stripe-mirror`.

Creating a RAID-5 volume

A RAID-5 volume requires space to be available on at least as many disks in the disk group as the number of columns in the volume. Additional disks may be required for any RAID-5 logs that are created.

Note: Veritas Volume Manager (VxVM) supports the creation of RAID-5 volumes in private disk groups, but not in shareable disk groups in a cluster environment.

You can create RAID-5 volumes by using either the `vxassist` command (recommended) or the `vxmake` command. This section describes using the preferred method, the `vxassist` command.

For information about using the `vxmake` command, see the `vxmake`(1M) manual page.

A RAID-5 volume contains a RAID-5 data plex that consists of three or more subdisks located on three or more physical disks. Only one RAID-5 data plex can exist per volume. A RAID-5 volume can also contain one or more RAID-5 log plexes, which are used to log information about data and parity being written to the volume.

See "RAID-5 (striping with parity)" on page 64.

Warning: Do not create a RAID-5 volume with more than 8 columns because the volume will be unrecoverable in the event of the failure of more than one disk.

To create a RAID-5 volume, use the following command:

```
# vxassist [-b] [-g diskgroup] make volume length layout=raid5 \
  [ncol=number_of_columns] [stripewidth=size] [nlog=number] \
  [loglen=log_length]
```

Specify the `-b` option if you want to make the volume immediately available for use.

For example, to create the RAID-5 volume `volraid` together with 2 RAID-5 logs in the disk group, `mydg`, use the following command:

```
# vxassist -b -g mydg make volraid 10g layout=raid5 nlog=2
```

This creates a RAID-5 volume with the default stripe unit size on the default number of disks. It also creates two RAID-5 logs rather than the default of one log.

If you require RAID-5 logs, you must use the logdisk attribute to specify the disks to be used for the log plexes.

RAID-5 logs can be concatenated or striped plexes, and each RAID-5 log associated with a RAID-5 volume has a complete copy of the logging information for the volume. To support concurrent access to the RAID-5 array, the log should be several times the stripe size of the RAID-5 plex.

It is suggested that you configure a minimum of two RAID-5 log plexes for each RAID-5 volume. These log plexes should be located on different disks. Having two RAID-5 log plexes for each RAID-5 volume protects against the loss of logging information due to the failure of a single disk.

If you use ordered allocation when creating a RAID-5 volume on specified storage, you must use the logdisk attribute to specify on which disks the RAID-5 log plexes should be created. Use the following form of the vxassist command to specify the disks from which space for the logs is to be allocated:

```
# vxassist [-b] [-g diskgroup] -o ordered make volume length \
  layout=raid5 [ncol=number_columns] [nlog=number] \
  [loglen=log_length] logdisk=disk[,disk,...] \
  storage_attributes
```

For example, the following command creates a 3-column RAID-5 volume with the default stripe unit size on disks mydg04, mydg05 and mydg06. It also creates two RAID-5 logs on disks mydg07 and mydg08.

```
# vxassist -b -g mydg -o ordered make volraid 10g layout=raid5 \
  ncol=3 nlog=2 logdisk=mydg07,mydg08 mydg04 mydg05 mydg06
```

The number of logs must equal the number of disks that is specified to logdisk.

See “Specifying ordered allocation of storage to volumes” on page 156.

See the vxassist(1M) manual page.

You can add more logs to a RAID-5 volume at a later time.

To add a RAID-5 log to an existing volume, use the following command:

```
# vxassist [-b] [-g diskgroup] addlog volume [loglen=length]
```

If you specify the -b option, adding the new log is a background task.

When you add the first log to a volume, you can specify the log length. Any logs that you add subsequently are configured with the same length as the existing log.

For example, to create a log for the RAID-5 volume volraid, in the disk group mydg, use the following command:

```
# vxassist -g mydg addlog volraid
```

Creating a volume on specific disks

Veritas Volume Manager (VxVM) automatically selects the disks on which each volume resides, unless you specify otherwise. If you want to select a particular type of disks for a certain volume, you can provide the storage specifications to `vxassist` for storage allocation.

For more infornation, see the Storage Specifications section of the `vxassist`(1M) manual page.

See "Customizing disk classes for allocation" on page 129.

See "Specifying allocation constraints for vxassist operations with the use clause and the require clause" on page 132.

If you want a volume to be created on specific disks, you must designate those disks to VxVM. More than one disk can be specified.

To create a volume on a specific disk or disks, use the following command:

```
# vxassist [-b] [-g diskgroup] make volume length \
  [layout=layout] diskname ...
```

Specify the `-b` option if you want to make the volume immediately available for use.

For example, to create the volume `volspec` with length 5 gigabytes on disks `mydg03` and `mydg04`, use the following command:

```
# vxassist -b -g mydg make volspec 5g mydg03 mydg04
```

The `vxassist` command allows you to specify storage attributes. These give you control over the devices, including disks and controllers, which `vxassist` uses to configure a volume.

For example, you can specifically exclude the disk `mydg05`.

Note: The ! character is a special character in some shells. The following examples show how to escape it in a bash shell.

```
# vxassist -b -g mydg make volspec 5g \!mydg05
```

The following example excludes all disks that are on controller c2:

```
# vxassist -b -g mydg make volspec 5g \!ctlr:c2
```

If you want a volume to be created using only disks from a specific disk group, use the -g option to vxassist, for example:

```
# vxassist -g bigone -b make volmega 20g bigone10 bigone11
```

or alternatively, use the diskgroup attribute:

```
# vxassist -b make volmega 20g diskgroup=bigone bigone10 \
   bigone11
```

Any storage attributes that you specify for use must belong to the disk group. Otherwise, vxassist will not use them to create a volume.

You can also use storage attributes to control how vxassist uses available storage, for example, when calculating the maximum size of a volume, when growing a volume or when removing mirrors or logs from a volume. The following example excludes disks mydg07 and mydg08 when calculating the maximum size of a RAID-5 volume that vxassist can create using the disks in the disk group mydg:

```
# vxassist -b -g mydg maxsize layout=raid5 nlog=2 \!mydg07 \!mydg08
```

It is also possible to control how volumes are laid out on the specified storage.

See "Specifying ordered allocation of storage to volumes" on page 156.

vxassist also lets you select disks based on disk tags. The following command only includes disks that have a tier1 disktag.

```
# vxassist -g mydg make vol3 1g disktag:tier1
```

See the vxassist(1M) manual page.

Creating volumes on specific media types

When you create a volume, you can specify the media type for the volume. The supported media types are Hard Disk Drives (HDD) or Solid State Devices (SSD). The SSD media type requires disk group 150 or greater. The default is HDD.

To specify a media type, specify the vxassist command with the mediatype attribute. If no mediatype is specified, the volume allocates storage only from the HDD devices.

Creating encrypted volumes

Set the attribute encrypted to on with the vxassist command to create encrypted volumes.

If you encrypt the volume with a password or passphrase, VxVM displays the volume status *encrypted with password* for the volume. If you have configured the Key Management Server and encrypt the volume using it, VxVM displays the volume status *encrypted* for the volume.

The following example encrypts a volume using a Key Management Server:

```
# vxassist -g mydg make vol01 1g encrypted=on
# vxassist -g mydg make vol02 1g
# vxencrypt list
Disk group: mydg

 VOLUME       STATUS
 vol01         encrypted
 vol02         not encrypted
```

Changing the encryption password

Use the `vxencrypt` command to change the encryption password at any time.

```
# vxencrypt -g mydg passwd vol01
Enter current password: xxxx
Enter new password: xxxx
Confirm new password: xxxxx
```

Changing the KEK using the re-key operation

InfoScale provides the option to re-key the volumes that change the KMS key when needed. This option is also known as key rotation. You can use an external scheduler based on your policy to schedule the re-key operation.

- On a single volume, use the `vxencrypt` command to change the key encryption key at any time.

```
vxencrypt -g DiskGroupName rekey volumeName
```

If the rekey operation for any volume fails due to any reason, run the `vxencrypt` command again. When the `vxencrypt` command is executed, InfoScale requests the KMS for a new identifier for the encrypted volumes.

Viewing encrypted volumes

Use the `vxencrypt` command to view the list of encrypted volumes.

```
# vxencrypt list
Disk group: mydg
        VOLUME       STATUS
        vol01         encrypted
        vol02         encrypted

Disk group: mydg1
        VOLUME       STATUS
        vol03         encrypted
        vol04         not encrypted
```

```
# vxencrypt -g mydg1 list
        VOLUME       STATUS
        vol03         encrypted
        vol04         not encrypted
```

Automating startup for encrypted volumes

By default, encrypted volumes can not start automatically when the system boots up because it requires the user to provide an access passphrase. However, you can automate the startup of encrypted volumes by storing the required passphrases in the `/etc/vx/encryption/password_file` file.

Caution: The password file is stored on the disk. Set secure file permissions to prevent unauthorized users from reading the file. You must also secure physical access to the hosts and storage on which the file is located.

The passphrase file must contain one line for each encrypted volume; each line contains the following information in three columns of text:

First column	Name of the disk group or disk group ID Use the * wildcard character to indicate any disk group.
Second column	Name of the encrypted volume Use the * wildcard character to indicate any volume.

Third column Passphrase

The passphrase must be specified as plain text in the `password_file` file.

VxVM does not impose any limitation on the size of the passphrase or the characters in the passphrase; however, new line or NULL character must not be specified in the passphrase.

At the time of system startup, VxVM queries the passphrase file for encrypted volumes. If the volume is listed in the file, it uses the corresponding passphrase for the volume instead of prompting the user for a manual entry.

A sample passphrase file is as follows:

```
datadg1    datavol1   mypassphrase1
datadg1    datavol2   mypassphrase2
```

Configuring a Key Management Server

You can configure a Key Management Server for volume encryption by creating the configuration file `/etc/vx/enc-kms-kmip.conf` on the KMIP client.

The configuration file must have the following information:

`host` The hostname or IP address of the Key Management Server

`port` The port number at which the Key Management Server accepts Key Management Interoperability Protocol (KMIP) clients

`keyfile` The location of the private key to be used by the KMIP client, in Privacy Enhanced Mail (PEM) format

`certfile` The location of the certificate to be used by the KMIP client, in PEM format

`cacerts` The location of the root certificate to be used for mutual authentication, in PEM format

`ssl_version` The SSL version to be used by the KMIP client.

A sample configuration file is as follows:

```
[client]
host = kms-enterprise.example.com
port = 5696
keyfile= /etc/vx/client-key.pem
certfile= /etc/vx/client-crt.pem
```

```
cacerts= /etc/vx/cacert.pem
ssl_version = PROTOCOL_TLSv1_2
```

Specifying ordered allocation of storage to volumes

Ordered allocation gives you complete control of space allocation. It requires that the number of disks that you specify to the `vxassist` command must match the number of disks that are required to create a volume. The order in which you specify the disks to `vxassist` is also significant.

If you specify the `-o ordered` option to `vxassist` when creating a volume, any storage that you also specify is allocated in the following order:

- Concatenate disks
- Form columns
- Form mirrors

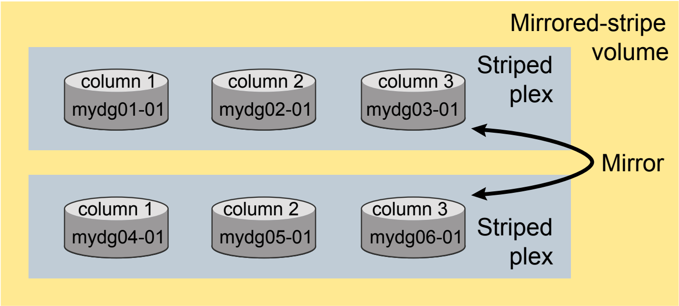
For example, the following command creates a mirrored-stripe volume with 3 columns and 2 mirrors on 6 disks in the disk group, `mydg`:

```
# vxassist -b -g mydg -o ordered make mirstrvol 10g \
  layout=mirror-stripe ncol=3 mydg01 mydg02 mydg03 \
  mydg04 mydg05 mydg06
```

This command places columns 1, 2, and 3 of the first mirror on disks `mydg01`, `mydg02`, and `mydg03` respectively, and columns 1, 2, and 3 of the second mirror on disks `mydg04`, `mydg05`, and `mydg06` respectively.

Figure 6-1 shows an example of using ordered allocation to create a mirrored-stripe volume.

Figure 6-1 Example of using ordered allocation to create a mirrored-stripe volume



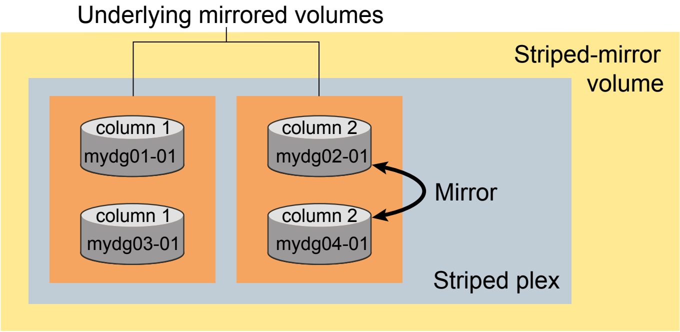
For layered volumes, `vxassist` applies the same rules to allocate storage as for non-layered volumes. For example, the following command creates a striped-mirror volume with 2 columns:

```
# vxassist -b -g mydg -o ordered make strmirvol 10g \
  layout=stripe-mirror ncol=2 mydg01 mydg02 mydg03 mydg04
```

This command mirrors column 1 across disks `mydg01` and `mydg03`, and column 2 across disks `mydg02` and `mydg04`.

Figure 6-2 shows an example of using ordered allocation to create a striped-mirror volume.

Figure 6-2 Example of using ordered allocation to create a striped-mirror volume



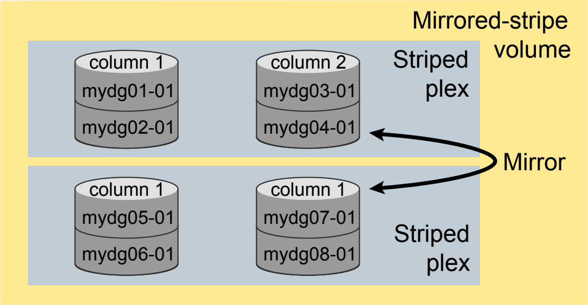
Additionally, you can use the `col_switch` attribute to specify how to concatenate space on the disks into columns. For example, the following command creates a mirrored-stripe volume with 2 columns:

```
# vxassist -b -g mydg -o ordered make strmir2vol 10g \
  layout=mirror-stripe ncol=2 col_switch=3g,2g \
  mydg01 mydg02 mydg03 mydg04 mydg05 mydg06 mydg07 mydg08
```

This command allocates 3 gigabytes from `mydg01` and 2 gigabytes from `mydg02` to column 1, and 3 gigabytes from `mydg03` and 2 gigabytes from `mydg04` to column 2. The mirrors of these columns are then similarly formed from disks `mydg05` through `mydg08`.

Figure 6-3 shows an example of using concatenated disk space to create a mirrored-stripe volume.

Figure 6-3 Example of using concatenated disk space to create a mirrored-stripe volume



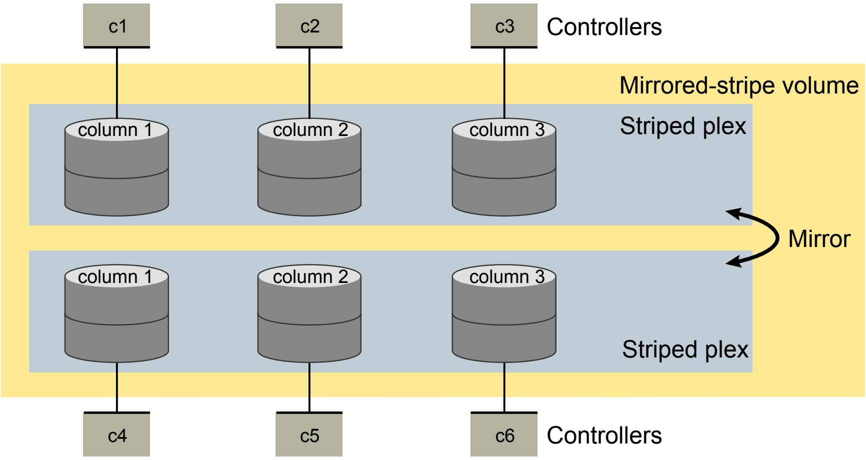
Other storage specification classes for controllers, enclosures, targets and trays can be used with ordered allocation. For example, the following command creates a 3-column mirrored-stripe volume between specified controllers:

```
# vxassist -b -g mydg -o ordered make mirstr2vol 80g \
  layout=mirror-stripe ncol=3 \
  ctlr:c1 ctlr:c2 ctlr:c3 ctlr:c4 ctlr:c5 ctlr:c6
```

This command allocates space for column 1 from disks on controllers `c1`, for column 2 from disks on controller `c2`, and so on.

Figure 6-4 shows an example of using storage allocation to create a mirrored-stripe volume across controllers.

Figure 6-4 Example of storage allocation used to create a mirrored-stripe volume across controllers



There are other ways in which you can control how `vxassist` lays out mirrored volumes across controllers.

Site-based allocation

In a Remote Mirror configuration (also known as a campus cluster or stretch cluster), the hosts and storage of a cluster are divided between two or more sites. These sites are typically connected through a redundant high-capacity network that provides access to storage and private link communication between the cluster nodes.

Configure the disk group in a Remote Mirror site to be site-consistent. When you create volumes in such a disk group, the volumes are mirrored across all sites by default.

See “About sites and remote mirrors” on page 313.

Changing the read policy for mirrored volumes

For a mirrored volume, Veritas Volume Manager (VxVM) uses the read policy to determine which data plex in the volume to use for reads. By default, VxVM chooses a plex using the following criteria, in order:

- Site
  Plexes on the same site are chosen over plexes on another site.
- Connectivity

Locally connected plexes are chosen over remotely connected plexes. This criterion applies for shared disk groups.

- Media type
  SSD devices are chosen over HDDs.
- Layout
  Striped plexes are chosen over other layouts

To customize the read policy, you can choose one of the following VxVM read policies:

`prefer`

Uses a particular named plex to satisfy read requests. Specify one preferred plex when you set the `prefer` policy.

If a read request cannot be satisfied by the preferred plex, VxVM applies the plex order in the `select` policy.

`round`

Distributes the non-sequential read operations in “round-robin” fashion across all of the available plexes. For example, given three plexes, VxVM switches between each of the three plexes, so that each plex receives one third of the read requests. Sequential read operations access only one plex. This approach takes advantage of the drive or controller read-ahead caching policies.

`select`

Chooses a plex based on the characteristics of the plex. The `select` policy is the default read policy, unless site consistency is enabled. If sites are configured, VxVM internally switches to the siteread policy.

The `select` policy chooses a plex in the following order:

- Locally connected striped SSD plexes
- Locally connected SSD plexes
- Locally connected striped plexes
- Locally connected plexes
- Remotely connected striped SSD plexes
- Remotely connected SSD plexes

If VxVM cannot find a plex with the above characteristics, VxVM uses the `round` policy.

`siteread`

Reads preferentially from plexes at the locally defined site. This method is the default policy for volumes in disk groups where site consistency is enabled.

The `siteread` policy chooses a plex in the following order:

- Local site, locally connected striped SSD plexes
- Local site, locally connected SSD plexes
- Local site, locally connected striped plexes
- Local site, locally connected plexes
- Local site, remotely connected striped SSD plexes
- Local site, remotely connected SSD plexes
- Local site, remotely connected striped plexes
- Local site, remotely connected plexes

If VxVM cannot find a plex with the above characteristics, VxVM refers to the plex order in the `select` policy.

`split`

Divides the read requests and distributes them across all the available plexes.

Note: You cannot set the read policy on a RAID-5 volume.

To set the read policy to `round`, use the following command:

```
# vxvol [-g diskgroup] rdpol round volume
```

For example, to set the read policy for the volume `vol01` in disk group `mydg` to round-robin, use the following command:

```
# vxvol -g mydg rdpol round vol01
```

To set the read policy to `prefer`, use the following command:

```
# vxvol [-g diskgroup] rdpol prefer volume preferred_plex
```

For example, to set the policy for `vol01` to read preferentially from the plex `vol01-02`, use the following command:

```
# vxvol -g mydg rdpol prefer vol01 vol01-02
```

To set the read policy to `select`, use the following command:

```
# vxvol [-g diskgroup] rdpol select volume
```

Chapter 7

Creating and mounting VxFS file systems

This chapter includes the following topics:

Creating a VxFS file system

The `mkfs` command creates a VxFS file system by writing to a special character device file. The special character device must be a Veritas Volume Manager (VxVM) volume. The `mkfs` command builds a file system with a root directory and a `lost+found` directory.

Before running `mkfs`, you must create the target device.

See to your operating system documentation.

If you are using a logical device (such as a VxVM volume), see the VxVM documentation.

Note: Creating a VxFS file system on a Logical Volume Manager (LVM) or Multiple Device (MD) driver volume is not supported in this release. You also must convert an underlying LVM to a VxVM volume before converting an `ext2` or `ext3` file system to a VxFS file system. See the `vxvmconvert`(1M) manual page.

See the `mkfs`(1M) and `mkfs_vxfs`(1M) manual pages.

When you create a file system with the `mkfs` command, you can select the following characteristics:

- File system block size
- Intent log size

To create a file system

◆ Use the `mkfs` command to create a file system:

```
mkfs [-t vxfs] [generic_options]
[-o specific_options] -m special [size]
```

`-t vxfs` Specifies the VxFS file system type.

`-m` Displays the command line that was used to create the file system. The file system must already exist. This option enables you to determine the parameters used to construct the file system.

generic_options Options common to most other file system types.

`-o` *specific_options* Options specific to VxFS.

`-o N` Displays the geometry of the file system and does not write to the device.

`-o largefiles` Allows users to create files larger than two gigabytes. The default option is largefiles.

`-o nomaxlink` Support is added for more than 64 K sub-directory. If maxlink is disabled on the file system, the sub-directory limit is 32 K by default.

special Specifies the special device file location or character device node of a particular storage device. The device must be a Veritas Volume Manager volume.

size Specifies the number of 512-byte sectors in the file system. If *size* is not specified, `mkfs` determines the size of the special device.

The following example creates a VxFS file system on a VxVM volume.

To create a VxFS file system

1 Create the file system:

```
# /opt/VRTS/bin/mkfs /dev/vx/rdsk/diskgroup/volume

version 17 layout
12288 sectors, 6144 blocks of size 1024, log size 256 blocks
rcq size 1024 blocks
largefiles supported
maxlink supported
WORM not supported
maxts supported
```

2 Mount the newly created file system:

```
# mount -t vxfs /dev/vx/dsk/diskgroup/volume /mnt1
```

File system block size

The unit of allocation in VxFS is an extent. Unlike some other UNIX file systems, VxFS does not make use of block fragments for allocation because storage is allocated in extents that consist of one or more blocks. You specify the block size when creating a file system by using the `mkfs -o bsize` option. The block size cannot be altered after the file system is created. The smallest available block size for VxFS is 1 KB.

The default block size is 1024 bytes for file systems smaller than 1 TB, and 8192 bytes for file systems 1 TB or larger.

Choose a block size based on the type of application being run. For example, if there are many small files, a 1 KB block size may save space. For large file systems, with relatively few files, a larger block size is more appropriate. Larger block sizes use less disk space in file system overhead, but consume more space for files that are not a multiple of the block size. The easiest way to judge which block sizes provide the greatest system efficiency is to try representative system loads against various sizes and pick the fastest.

Intent log size

You specify the intent log size when creating a file system by using the `mkfs -o logsize` option.You can dynamically increase or decrease the intent log size using the `logsize` option of the `fsadm` command. The `mkfs` utility uses a default intent log size of 64 megabytes. The default size is sufficient for most workloads. If the system is used as an NFS server or for intensive synchronous write workloads, performance may be improved using a larger log size.

See the `fsadm_vxfs`(1M) and `mkfs_vxfs`(1M) manual pages.

With larger intent log sizes, recovery time is proportionately longer and the file system may consume more system resources (such as memory) during normal operation.

There are several system performance benchmark suites for which VxFS performs better with larger log sizes. As with block sizes, the best way to pick the log size is to try representative system loads against various sizes and pick the fastest.

Converting a file system to VxFS

To meet your storage needs you may want to convert your file system to Veritas File System (VxFS).

InfoScale provides an offline as well as online solution to convert a native file system (Ext2, Ext3, and Ext4) to VxFS.

The online conversion of file system is a real-time migration process. This process requires additional separate storage to copy the application data from the existing LVM volume to the VxVM volumes. This conversion process involves migration of the file system with limited application downtime. Use the `fsmigadm` command to perform the online migration of a native file system to VxFS.

The offline conversion of a file system is a process that allows you to convert your existing file system while your applications are offline. The offline conversion involves two steps:

1. Conversion of Logical Volume Manager (LVM) volume groups and objects to the equivalent VxVM disk groups and objects. Use the `vxvmconvert` utility to convert the LVM groups to VxVM disk groups.
2. Conversion of file system to VxFS. Use the `vxfsconvert` utility to convert the native file system to VxFS.

For more details about converting a file system, refer to, *Veritas InfoScale Solutions Guide*.

Mounting a VxFS file system

You can mount a VxFS file system by using the `mount` command. When you enter the `mount` command, the generic `mount` command parses the arguments and the `-t FSType` option executes the `mount` command specific to that file system type. If the `-t` option is not supplied, the command searches the file `/etc/fstab` for a file system and an FSType matching the special file or mount point provided. If no file system type is specified, mount uses the default file system.

The mount command automatically runs the VxFS fsck command to clean up the intent log if the mount command detects a dirty log in the file system. This functionality is only supported on file systems mounted on a Veritas Volume Manager (VxVM) volume.

On RHEL, you can use the context, defcontext, fscontext, and rootcontext mount options to specify a mode for VxFS operation.

In addition to the standard mount mode (delaylog mode), Veritas File System (VxFS) provides the following mount options for you to specify other modes of operation:

- log mount option
- delaylog mount option
- tmplog mount option
- logiosize mount option
- nodatainlog mount option
- blkclear mount option
- mincache mount option
- convosync mount option
- ioerror mount option
- largefiles and nolargefiles mount options
- cio mount option
- mntlock mount option
- ckptautomnt mount option

Caching behavior can be altered with the mincache option, and the behavior of O_SYNC and D_SYNC writes can be altered with the convosync option.

See the fcntl(2) manual page.

The delaylog and tmplog modes can significantly improve performance. The improvement over log mode is typically about 15 to 20 percent with delaylog; with tmplog, the improvement is even higher. Performance improvement varies, depending on the operations being performed and the workload. Read/write intensive loads should show less improvement, while file system structure intensive loads, such as mkdir, create, and rename, may show over 100 percent improvement.

The best way to select a mode is to test representative system loads against the logging modes and compare the performance results.

Most of the modes can be used in combination. For example, a desktop machine might use both the `blkclear` and `mincache=closesync` modes.

The `mount` command automatically runs the VxFS `fsck` command to clean up the intent log if the `mount` command detects a dirty log in the file system. This functionality is only supported on file systems mounted on a Veritas Volume Manager (VxVM) volume.

See the `mount_vxfs`(1M) manual page.

To mount a file system

- Use the `mount` command to mount a file system:

```
mount [-t vxfs] [generic_options] [-r] [-o specific_options] \
special mount_point
```

`vxfs`	File system type.
generic_options	Options common to most other file system types.
specific_options	Options specific to VxFS.
`-o ckpt=`*`ckpt_name`*	Mounts a Storage Checkpoint.
`-o cluster`	Mounts a file system in shared mode. Available only with the VxFS cluster file system feature.
special	A VxFS block special device.
mount_point	Directory on which to mount the file system.
`-r`	Mounts the file system as read-only.

The following example mounts the file system `/dev/vx/dsk/fsvol/vol1` on the `/mnt1` directory with read/write access and delayed logging.

Example of mounting a file system

- Mount the file system `/dev/vx/dsk/fsvol/vol1` on the `/mnt1` directory with read/write access and delayed logging:

```
# mount -t vxfs -o delaylog /dev/vx/dsk/fsvol/vol1 /mnt1
```

log mount option

File systems are typically asynchronous in that structural changes to the file system are not immediately written to disk, which provides better performance. However, recent changes made to a system can be lost if a system failure occurs. Specifically, attribute changes to files and recently created files may disappear. In log mode, all system calls other than `write`(2), `writev`(2), and `pwrite`(2) are guaranteed to be persistent after the system call returns to the application.

The `rename`(2) system call flushes the source file to disk to guarantee the persistence of the file data before renaming it. In both the `log` and `delaylog` modes, the rename is also guaranteed to be persistent when the system call returns. This benefits shell scripts and programs that try to update a file atomically by writing the new file contents to a temporary file and then renaming it on top of the target file.

delaylog mount option

The default logging mode is `delaylog`, in which writing to a file is delayed, or buffered, meaning that the data to be written is copied to the file system cache and later flushed to disk. In `delaylog` mode, the effects of most system calls other than `write`(2), `writev`(2), and `pwrite`(2) are guaranteed to be persistent approximately three seconds after the system call returns to the application. Contrast this with the behavior of most other file systems in which most system calls are not persistent until approximately 30 seconds or more after the call has returned. Fast file system recovery works with this mode.

A delayed write provides much better performance than synchronously writing the data to disk. However, in the event of a system failure, data written shortly before the failure may be lost since it was not flushed to disk. In addition, if space was allocated to the file as part of the write request, and the corresponding data was not flushed to disk before the system failure occurred, uninitialized data can appear in the file.

For the most common type of write, delayed extending writes (a delayed write that increases the file size), VxFS avoids the problem of uninitialized data appearing in the file by waiting until the data has been flushed to disk before updating the new file size to disk. If a system failure occurs before the data has been flushed to disk, the file size has not yet been updated, thus no uninitialized data appears in the file. The unused blocks that were allocated are reclaimed.

The `rename`(2) system call flushes the source file to disk to guarantee the persistence of the file data before renaming it. In the `log` and `delaylog` modes, the `rename` is also guaranteed to be persistent when the system call returns. This benefits shell scripts and programs that try to update a file atomically by writing the new file contents to a temporary file and then renaming it on top of the target file.

tmplog mount option

In tmplog mode, the effects of system calls have persistence guarantees that are similar to those in delaylog mode. In addition, enhanced flushing of delayed extending writes is disabled, which results in better performance but increases the chances of data being lost or uninitialized data appearing in a file that was being actively written at the time of a system failure. This mode is only recommended for temporary file systems. Fast file system recovery works with this mode.

Note: The term "effects of system calls" refers to changes to file system data and metadata caused by a system call, excluding changes to st_atime.

See the stat(2) manual page.

Logging mode persistence guarantees

In all logging modes, VxFS is fully POSIX compliant. The effects of the fsync(2) and fdatasync(2) system calls are guaranteed to be persistent after the calls return. The persistence guarantees for data or metadata modified by write(2), writev(2), or pwrite(2) are not affected by the logging mount options. The effects of these system calls are guaranteed to be persistent only if the O_SYNC, O_DSYNC, VX_DSYNC, or VX_DIRECT flag, as modified by the convosync= mount option, has been specified for the file descriptor.

The behavior of NFS servers on a VxFS file system is unaffected by the log and tmplog mount options, but not delaylog. In all cases except for tmplog, VxFS complies with the persistency requirements of the NFSv3 and NFSv4 standard. Unless a UNIX application has been developed specifically for the VxFS file system in log mode, it expects the persistence guarantees offered by most other file systems and experiences improved robustness when used with a VxFS file system mounted in delaylog mode. Applications that expect better persistence guarantees than that offered by most other file systems can benefit from the log, mincache=, and closesync mount options. However, most commercially available applications work well with the default VxFS mount options, including the delaylog mode.

See the mount_vxfs(1M) manual page.

logiosize mount option

The logiosize=size option enhances the performance of storage devices that employ a read-modify-write feature. If you specify logiosize when you mount a file system, VxFS writes the intent log in the least *size* bytes or a multiple of *size* bytes to obtain the maximum performance from such devices.

See the `mount_vxfs`(1M) manual page.

The values for *size* can be 512, 1024, 2048, 4096, or 8192.

nodatainlog mount option

Use the `nodatainlog` mode on systems with disks that do not support bad block revectoring. Usually, a VxFS file system uses the intent log for synchronous writes. The inode update and the data are both logged in the transaction, so a synchronous write only requires one disk write instead of two. When the synchronous write returns to the application, the file system has told the application that the data is already written. If a disk error causes the metadata update to fail, then the file must be marked bad and the entire file is lost.

If a disk supports bad block revectoring, then a failure on the data update is unlikely, so logging synchronous writes should be allowed. If the disk does not support bad block revectoring, then a failure is more likely, so the `nodatainlog` mode should be used.

A `nodatainlog` mode file system is approximately 50 percent slower than a standard mode VxFS file system for synchronous writes. Other operations are not affected.

blkclear mount option

The `blkclear` mode is used in increased data security environments. The `blkclear` mode guarantees that uninitialized storage never appears in files. The increased integrity is provided by clearing extents on disk when they are allocated within a file. This mode does not affect extending writes. A `blkclear` mode file system is approximately 10 percent slower than a standard mode VxFS file system, depending on the workload.

mincache mount option

The `mincache` mode has the following suboptions:

- `mincache=closesync`
- `mincache=direct`
- `mincache=dsync`
- `mincache=unbuffered`
- `mincache=tmpcache`

The `mincache=closesync` mode is useful in desktop environments where users are likely to shut off the power on the machine without halting it first. In this mode, any changes to the file are flushed to disk when the file is closed.

To improve performance, most file systems do not synchronously update data and inode changes to disk. If the system crashes, files that have been updated within the past minute are in danger of losing data. With the mincache=closesync mode, if the system crashes or is switched off, only open files can lose data. A mincache=closesync mode file system could be approximately 15 percent slower than a standard mode VxFS file system, depending on the workload.

The following describes where to use the mincache modes:

- The mincache=direct, mincache=unbuffered, and mincache=dsync modes are used in environments where applications have reliability problems caused by the kernel buffering of I/O and delayed flushing of non-synchronous I/O.
- The mincache=direct and mincache=unbuffered modes guarantee that all non-synchronous I/O requests to files are handled as if the VX_DIRECT or VX_UNBUFFERED caching advisories had been specified.
- The mincache=dsync mode guarantees that all non-synchronous I/O requests to files are handled as if the VX_DSYNC caching advisory had been specified. Refer to the vxfsio(7) manual page for explanations of VX_DIRECT, VX_UNBUFFERED, and VX_DSYNC, as well as for the requirements for direct I/O.
- The mincache=direct, mincache=unbuffered, and mincache=dsync modes also flush file data on close as mincache=closesync does.

Because the mincache=direct, mincache=unbuffered, and mincache=dsync modes change non-synchronous I/O to synchronous I/O, throughput can substantially degrade for small to medium size files with most applications. Since the VX_DIRECT and VX_UNBUFFERED advisories do not allow any caching of data, applications that normally benefit from caching for reads usually experience less degradation with the mincache=dsync mode. mincache=direct and mincache=unbuffered require significantly less CPU time than buffered I/O.

If performance is more important than data integrity, you can use the mincache=tmpcache mode. The mincache=tmpcache mode disables special delayed extending write handling, trading off less integrity for better performance. Unlike the other mincache modes, tmpcache does not flush the file to disk when the file is closed. When the mincache=tmpcache option is used, bad data can appear in a file that was being extended when a crash occurred.

See the mount_vxfs(1M) manual page.

convosync mount option

The convosync (convert osync) mode has the following suboptions:

- convosync=closesync

Note: The convosync=closesync mode converts synchronous and data synchronous writes to non-synchronous writes and flushes the changes to the file to disk when the file is closed.

- convosync=delay
- convosync=direct
- convosync=dsync

Note: The convosync=dsync option violates POSIX guarantees for synchronous I/O.

- convosync=unbuffered

The convosync=delay mode causes synchronous and data synchronous writes to be delayed rather than to take effect immediately. No special action is performed when closing a file. This option effectively cancels any data integrity guarantees normally provided by opening a file with O_SYNC.

See the open(2), fcntl(2), and vxfsio(7) manual pages.

Warning: Be very careful when using the convosync=closesync or convosync=delay mode because they actually change synchronous I/O into non-synchronous I/O. Applications that use synchronous I/O for data reliability may fail if the system crashes and synchronously-written data is lost.

The convosync=dsync mode converts synchronous writes to data synchronous writes.

As with closesync, the direct, unbuffered, and dsync modes flush changes to the file to disk when it is closed. These modes can be used to speed up applications that use synchronous I/O. Many applications that are concerned with data integrity specify the O_SYNC fcntl in order to write the file data synchronously. However, this has the undesirable side effect of updating inode times and therefore slowing down performance. The convosync=dsync, convosync=unbuffered, and convosync=direct modes alleviate this problem by allowing applications to take advantage of synchronous writes without modifying inode times as well.

Before using convosync=dsync, convosync=unbuffered, or convosync=direct, make sure that all applications that use the file system do not require synchronous inode time updates for O_SYNC writes.

ioerror mount option

This mode sets the policy for handling I/O errors on a mounted file system. I/O errors can occur while reading or writing file data or metadata. The file system can respond to these I/O errors either by halting or by gradually degrading. The `ioerror` option provides five policies that determine how the file system responds to the various errors. All policies limit data corruption, either by stopping the file system or by marking a corrupted inode as bad.

The policies are as follows:

- disable policy
- wdisable policy and mwdisable policy
- mdisable policy

disable policy

If `disable` is selected, VxFS disables the file system after detecting any I/O error. You must then unmount the file system and correct the condition causing the I/O error. After the problem is repaired, run `fsck` and mount the file system again. In most cases, replay `fsck` to repair the file system. A full `fsck` is required only in cases of structural damage to the file system's metadata. Select `disable` in environments where the underlying storage is redundant, such as RAID-5 or mirrored disks.

wdisable policy and mwdisable policy

If `wdisable` (write disable) or `mwdisable` (metadata-write disable) is selected, the file system is disabled or degraded, depending on the type of error encountered. Select `wdisable` or `mwdisable` for environments where read errors are more likely to persist than write errors, such as when using non-redundant storage. `mwdisable` is the default `ioerror` mount option for local mounts.

Note: The mirrored volume file system is not disabled when `wdisable` or `mwdisable` is selected, if the problem occurs when there is only one plex.

See the `mount_vxfs`(1M) manual page.

Note: If the `nodisable` option is selected, the behavior will be same as the `mwdisable ioerror` policy. For more information see the `mwdisable` option.

mdisable policy

If mdisable (metadata disable) is selected, the file system is disabled if a metadata read or write fails. However, the file system continues to operate if the failure is confined to data extents. mdisable is the default ioerror mount option for cluster mounts.

largefiles and nolargefiles mount options

Veritas File System (VxFS) supports sparse files up to 16 terabytes, and non-sparse files up to 2 terabytes - 1 kilobyte.

Note: Applications and utilities such as backup may experience problems if they are not aware of large files. In such a case, create your file system without large file capability.

See "Creating a file system with large files" on page 174.

See "Mounting a file system with large files" on page 174.

See "Managing a file system with large files" on page 175.

Creating a file system with large files

To create a file system with large file capability:

```
# mkfs -t vxfs -o largefiles special_device size
```

Specifying largefiles sets the largefiles flag. This enables the file system to hold files that are two gigabytes or larger. This is the default option.

To clear the flag and prevent large files from being created:

```
# mkfs -t vxfs -o nolargefiles special_device size
```

The largefiles flag is persistent and stored on disk.

Mounting a file system with large files

If a mount succeeds and nolargefiles is specified, the file system cannot contain or create any large files. If a mount succeeds and largefiles is specified, the file system may contain and create large files.

The mount command fails if the specified largefiles|nolargefiles option does not match the on-disk flag.

Because the mount command defaults to match the current setting of the on-disk flag if specified without the largefiles or nolargefiles option, the best practice is not to specify either option. After a file system is mounted, you can use the fsadm utility to change the large files option.

Managing a file system with large files

Managing a file system with large files includes the following tasks:

- Determining the current status of the large files flag
- Switching capabilities on a mounted file system
- Switching capabilities on an unmounted file system

To determine the current status of the largefiles flag, type either of the following commands:

```
# mkfs -t vxfs -m special_device
# /opt/VRTS/bin/fsadm mount_point | special_device
```

To switch capabilities on a mounted file system:

```
# /opt/VRTS/bin/fsadm -o [no]largefiles mount_point
```

To switch capabilities on an unmounted file system:

```
# /opt/VRTS/bin/fsadm -o [no]largefiles special_device
```

You cannot change a file system to nolargefiles if it contains large files.

See the mount_vxfs(1M), fsadm_vxfs(1M), and mkfs_vxfs(1M) manual pages.

cio mount option

The cio (Concurrent I/O) option specifies the file system to be mounted for concurrent reads and writes. If cio is specified, but the license is not present, the mount command prints an error message and terminates the operation without mounting the file system. The cio option cannot be disabled through a remount. To disable the cio option, the file system must be unmounted and mounted again without the cio option.

mntlock mount option

The mntlock option prevents a file system from being unmounted by an application. This option is useful for applications that do not want the file systems that the applications are monitoring to be improperly unmounted by other applications or administrators.

The mntunlock option of the vxumount command reverses the mntlock option if you previously locked the file system.

ckptautomnt mount option

The ckptautomnt option enables the Storage Checkpoint visibility feature, which makes Storage Checkpoints easier to access.

See "Storage Checkpoint visibility" on page 369.

Combining mount command options

Although mount options can be combined arbitrarily, some combinations do not make sense. The following examples provide some common and reasonable mount option combinations.

To mount a desktop file system using options:

```
# mount -t vxfs -o log,mincache=closesync \
/dev/vx/dsk/diskgroup/volume /mnt
```

This guarantees that when a file is closed, its data is synchronized to disk and cannot be lost. Thus, after an application has exited and its files are closed, no data is lost even if the system is immediately turned off.

To mount a temporary file system or to restore from backup:

```
# mount -t vxfs -o tmplog,convosync=delay,mincache=tmpcache \
/dev/vx/dsk/diskgroup/volume /mnt
```

This combination might be used for a temporary file system where performance is more important than absolute data integrity. Any O_SYNC writes are performed as delayed writes and delayed extending writes are not handled. This could result in a file that contains corrupted data if the system crashes. Any file written 30 seconds or so before a crash may contain corrupted data or be missing if this mount combination is in effect. However, such a file system does significantly less disk writes than a log file system, and should have significantly better performance, depending on the application.

To mount a file system for synchronous writes:

```
# mount -t vxfs -o log,convosync=dsync \
/dev/vx/dsk/diskgroup/volume /mnt
```

This combination can be used to improve the performance of applications that perform O_SYNC writes, but only require data synchronous write semantics. Performance can be significantly improved if the file system is mounted using convosync=dsync without any loss of data integrity.

Unmounting a file system

Use the umount command to unmount a currently mounted file system.

See the vxumount(1M) manual page.

To unmount a file system

◆ Use the umount command to unmount a file system:

  Specify the file system to be unmounted as a *mount_point* or *special*. *special* is the VxFS block special device on which the file system resides.

The following is an example of unmounting a file system.

Example of unmounting a file system

◆ Unmount the file system /dev/vx/dsk/fsvol/vol1:

```
# umount /dev/vx/dsk/fsvol/vol1
```

Resizing a file system

You can extend or shrink mounted VxFS file systems using the fsadm command. The size to which a file system can be increased depends on the file system disk layout version. A file system with version 7 or later can be increased up to 256 terabytes. The size to which a file system with version 7 or later disk layout can be increased depends on the file system block size.

See the fsadm_vxfs(1M) and fdisk(8) manual pages.

Extending a file system using fsadm

You can resize a file system by using the fsadm command.

To resize a VxFS file system

◆ Use the fsadm command to extend a VxFS file system:

```
fsadm [-t vxfs] [-b newsize] [-r rawdev] \
mount_point
```

`vxfs`	The file system type.
newsize	The size to which the file system will increase. The default units is sectors, but you can specify `k` or `K` for kilobytes, `m` or `M` for megabytes, or `g` or `G` for gigabytes.
mount_point	The file system's mount point.
`-r rawdev`	Specifies the path name of the raw device if there is no entry in `/etc/fstab` and `fsadm` cannot determine the raw device.

The following example extends a file system mounted at `/mnt1` to 22528 sectors.

Example of extending a file system to 22528 sectors

- Extend the VxFS file system mounted on `/mnt1` to 22528 sectors:

```
# fsadm -t vxfs -b 22528 /mnt1
```

The following example extends a file system mounted at `/mnt1` by 500 gigabytes.

Example of extending a file system by 500 gigabytes

- Extend the VxFS file system mounted on `/mnt1` by 500 gigabytes:

```
# fsadm -t vxfs -b +500g /mnt1
```

Shrinking a file system

You can decrease the size of the file system using `fsadm`, even while the file system is mounted.

Warning: After this operation, there is unused space at the end of the device. You can then resize the device, but be careful not to make the device smaller than the new size of the file system.

To decrease the size of a VxFS file system

- Use the `fsadm` command to decrease the size of a VxFS file system:

```
fsadm  [-t vxfs] [-b newsize] [-r rawdev] mount_point
```

`vxfs`	The file system type.

newsize	The size to which the file system will shrink. The default units is sectors, but you can specify `k` or `K` for kilobytes, `m` or `M` for megabytes, or `g` or `G` for gigabytes.
mount_point	The file system's mount point.
`-r rawdev`	Specifies the path name of the raw device if there is no entry in `/etc/fstab` and `fsadm` cannot determine the raw device.

The following example shrinks a VxFS file system mounted at `/mnt1` to 20480 sectors.

Example of shrinking a file system to 20480 sectors

◆ Shrink a VxFS file system mounted at `/mnt1` to 20480 sectors:

```
# fsadm -t vxfs -b 20480 /mnt1
```

The following example shrinks a file system mounted at `/mnt1` by 450 gigabytes.

Example of shrinking a file system by 450 gigabytes

◆ Shrink the VxFS file system mounted on `/mnt1` by 450 gigabytes:

```
# fsadm -t vxfs -b -450g /mnt1
```

Reorganizing a file system

You can reorganize or compact a fragmented file system using `fsadm`, even while the file system is mounted. This may help shrink a file system that could not previously be decreased.

To reorganize a VxFS file system

◆ Use the `fsadm` command to reorganize a VxFS file system:

```
fsadm [-t vxfs] [-e] [-d] [-E] [-i] [-D] [-H] [-r rawdev] mount_point
```

`vxfs`	The file system type.
`-d`	Reorders directory entries to put subdirectory entries first, then all other entries in decreasing order of time of last access. Also compacts directories to remove free space.
`-D`	Reports on directory fragmentation.

`-e` Minimizes file system fragmentation. Files are reorganized to have the minimum number of extents.

`-E` Reports on extent fragmentation.

`i` Indicates that the files that are used in last 60 seconds must not be reorganized.

During the file system reorganization, if an application is actively accessing a particular file or has accessed it in the last 60 seconds, then the file system reorganization must not affect the file read-write operation and must not reorganize it.

`-H` Displays the storage size in human-friendly units (KB/MB/GB/TB/PB/EB), when used with the `-E` and `-D` options.

mount_point The file system's mount point.

`-r` *rawdev* Specifies the path name of the raw device if there is no entry in `/etc/fstab` and `fsadm` cannot determine the raw device.

To perform free space defragmentation

- Use the `fsadm` command to perform free space defragmentation of a VxFS file system:

```
fsadm [-t vxfs] [-C] mount_point
```

`vxfs` The file system type.

`-C` Minimizes file system free space fragmentation. This attempts to generate bigger chunks of free space in the device.

mount_point The file system's mount point.

The following example reorganizes the file system mounted at `/mnt1`.

Example of reorganizing a VxFS file system

- Reorganize the VxFS file system mounted at `/mnt1`:

```
# fsadm -t vxfs -EeDd /mnt1
```

The following example minimizes the free space fragmentation of the file system mounted at `/mnt1`.

Example of running free space defragmentation

◆ Minimize the free space of the the VxFS file system mounted at `/mnt1`:

```
# fsadm -t vxfs -C /mnt1
```

Displaying information on mounted file systems

Use the `mount` command to display a list of currently mounted file systems.

See the `mount_vxfs`(1M) and `mount`(8) manual pages.

To view the status of mounted file systems

◆ Use the `mount` command to view the status of mounted file systems:

```
mount
```

This shows the file system type and `mount` options for all mounted file systems.

The following example displays information on mounted file systems by invoking the `mount` command without options.

To display information on mounted file systems

◆ Invoke the `mount` command without options:

```
# mount
/dev/sda3 on / type ext3 (rw,acl,user_xattr)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
/dev/vx/dsk/testdg/vol01 on /vol01_testdg type vxfs
 (rw,delaylog,largefiles,ioerror=mwdisable)
```

Identifying file system types

Use the `fstyp` command to determine the file system type for a specified file system. This is useful when a file system was created elsewhere and you want to know its type.

See the `fstyp_vxfs`(1M) manual page.

To determine a file system's type

◆ Use the `fstyp` command to determine a file system's type:

```
fstyp -v special
```

special The block or character (raw) device.

`-v` Specifies the device that needs to be checked.

The following example uses the `fstyp` command to determine the file system type of the `/dev/vx/dsk/fsvol/vol1` device.

To determine the file system's type

◆ Use the `fstyp` command to determine the file system type of the device `/dev/vx/dsk/fsvol/vol1`:

```
  # fstyp -v /dev/vx/dsk/fsvol/vol1
```

The output indicates that the file system type is `vxfs`, and displays file system information similar to the following:

```
 vxfs
magic a501fcf5  version 16  ctime Mon 04 May 2020 04:05:56 PM IST
volguid 3f03b5b2-8c55-11ea-b66f-abd4f2521f81
logstart 0  logend 0
bsize  1024 size  41943040 dsize  41943040  ninode 0  nau 0
defiextsize 0  ilbsize 0  immedlen 96  ndaddr 10
aufirst 0  emap 0  imap 0  iextop 0  istart 0
bstart 0  femap 0  fimap 0  fiextop 0  fistart 0  fbstart 0
nindir 2048  aulen 32768  auimlen 0  auemlen 8
auilen 0  aupad 0  aublocks 32768  maxtier 15
inopb 4  inopau 0  ndiripau 0  iaddrlen 8   bshift 10
inoshift 2  bmask fffffc00  boffmask 3ff  checksum 103b34fd9
oltext1 32  oltext2 4714  oltsize 1  checksum2 0
free 41862017  ifree 0
efree  1 2 3 2 2 0 1 2 2 2 1 1 1 1 2 0 0 1 1 1 1 1 1 2 1 0 0 0 0 0 0 0
```

Monitoring free space

In general, Veritas File System (VxFS) works best if the percentage of free space in the file system does not get below 10 percent. This is because file systems with 10 percent or more free space have less fragmentation and better extent allocation. Regular use of the `df` command to monitor free space is desirable.

See the `df_vxfs`(1M) manual page.

Full file systems may have an adverse effect on file system performance. Full file systems should therefore have some files removed, or should be expanded.

See the fsadm_vxfs(1M) manual page.

VxFS supports reclamation of free storage on a Thin Storage LUN.

See “About Thin Reclamation of a file system” on page 481.

Monitoring fragmentation

Fragmentation reduces performance and availability. Veritas recommends regular use of the fragmentation reporting and reorganization facilities of the fsadm command.

The easiest way to ensure that fragmentation does not become a problem is to schedule regular defragmentation runs using the cron command.

Defragmentation scheduling should range from weekly (for frequently used file systems) to monthly (for infrequently used file systems). Extent fragmentation should be monitored with the fsadm command.

To determine the degree of fragmentation, use the following factors:

- Percentage of free space in extents of less than 8 blocks in length
- Percentage of free space in extents of less than 64 blocks in length
- Percentage of free space in extents of length 64 blocks or greater

An unfragmented file system has the following characteristics:

- Less than 1 percent of free space in extents of less than 8 blocks in length
- Less than 5 percent of free space in extents of less than 64 blocks in length
- More than 5 percent of the total file system size available as free extents in lengths of 64 or more blocks

A badly-fragmented file system has one or more of the following characteristics:

- Greater than 5 percent of free space in extents of less than 8 blocks in length
- More than 50 percent of free space in extents of less than 64 blocks in length
- Less than 5 percent of the total file system size available as free extents in lengths of 64 or more blocks

Fragmentation can also be determined based on the fragmentation index. Two types of indices are generated by the fsadm command: the file fragmentation index and the free space fragmentation index. Both of these indices range between 0 and 100, and give an idea about the level of file fragmentation and free space fragmentation, respectively. A value of 0 for the fragmentation index means that the file system has no fragmentation, and a value of 100 means that the file system has the highest level of fragmentation. Based on the index, you should use the appropriate defragmentation option with the fsadm command. For example if the file fragmentation index is high, the `fsadm` command should be run with the `-e` option. If the free space fragmentation index is high, the `fsadm` command should be run with `-C` option. When the `fsadm` command is run with the `-e` option, internally it performs free space defragmentation before performing file defragmentaion.

The optimal period for scheduling of extent reorganization runs can be determined by choosing a reasonable interval, scheduling `fsadm` runs at the initial interval, and running the extent fragmentation report feature of `fsadm` before and after the reorganization.

The “before" result is the degree of fragmentation prior to the reorganization. If the degree of fragmentation is approaching the figures for bad fragmentation, reduce the interval between `fsadm` runs. If the degree of fragmentation is low, increase the interval between `fsadm` runs.

The “after" result is an indication of how well the reorganizer has performed. The degree of fragmentation should be close to the characteristics of an unfragmented file system. If not, it may be a good idea to resize the file system; full file systems tend to fragment and are difficult to defragment. It is also possible that the reorganization is not being performed at a time during which the file system in question is relatively idle.

Directory reorganization is not nearly as critical as extent reorganization, but regular directory reorganization improves performance. It is advisable to schedule directory reorganization for file systems when the extent reorganization is scheduled. The following is a sample script that is run periodically at 3:00 A.M. from `cron` for a number of file systems:

```
outfile=/var/spool/fsadm/out.`/bin/date +'%m%d'`
for i in /home /home2 /project /db
do
  /bin/echo "Reorganizing $i"
  /usr/bin/time /opt/VRTS/bin/fsadm -t vxfs -e -E -s $i
  /usr/bin/time /opt/VRTS/bin/fsadm -t vxfs -s -d -D $i
done > $outfile 2>&1
```

Extent attributes

This chapter includes the following topics:

- About extent attributes
- Commands related to extent attributes

About extent attributes

Veritas File System (VxFS) allocates disk space to files in groups of one or more adjacent blocks called extents. VxFS defines an application interface that allows programs to control various aspects of the extent allocation for a given file. The extent allocation policies associated with a file are referred to as extent attributes.

The VxFS `getext` and `setext` commands let you view or manipulate file extent attributes.

See the `setext`(1) and `getext`(1) manual pages.

The two basic extent attributes associated with a file are its reservation and its fixed extent size. You can preallocate space to the file by manipulating a file's reservation, or override the default allocation policy of the file system by setting a fixed extent size.

See "Reservation: preallocating space to a file" on page 186.

See "Fixed extent size" on page 186.

Other policies determine the way these attributes are expressed during the allocation process.

You can specify the following criteria:

- The space reserved for a file must be contiguous
- No allocations will be made for a file beyond the current reservation
- An unused reservation will be released when the file is closed

- Space will be allocated, but no reservation will be assigned
- The file size will be changed to incorporate the allocated space immediately

Some of the extent attributes are persistent and become part of the on-disk information about the file, while other attributes are temporary and are lost after the file is closed or the system is rebooted. The persistent attributes are similar to the file's permissions and are written in the inode for the file. When a file is copied, moved, or archived, only the persistent attributes of the source file are preserved in the new file.

See "Other extent attribute controls" on page 187.

In general, the user will only set extent attributes for reservation. Many of the attributes are designed for applications that are tuned to a particular pattern of I/O or disk alignment.

See "About Veritas File System I/O" on page 332.

Reservation: preallocating space to a file

Veritas File System (VxFS) makes it possible to preallocate space to a file at the time of the request rather than when data is written into the file. This space cannot be allocated to other files in the file system. VxFS prevents any unexpected out-of-space condition on the file system by ensuring that a file's required space will be associated with the file before it is required.

A persistent reservation is not released when a file is truncated. The reservation must be cleared or the file must be removed to free the reserved space.

Fixed extent size

The Veritas File System (VxFS) default allocation policy uses a variety of methods to determine how to make an allocation to a file when a write requires additional space. The policy attempts to balance the two goals of optimum I/O performance through large allocations and minimal file system fragmentation. VxFS accomplishes these goals by allocating from space available in the file system that best fits the data.

Setting a fixed extent size overrides the default allocation policies for a file and always serves as a persistent attribute. Be careful to choose an extent size appropriate to the application when using fixed extents. An advantage of the VxFS extent-based allocation policies is that they rarely use indirect blocks compared to block-based file systems; VxFS eliminates many instances of disk access that stem from indirect references. However, a small extent size can eliminate this advantage.

Files with large extents tend to be more contiguous and have better I/O characteristics. However, the overall performance of the file system degrades because the unused space fragments free space by breaking large extents into smaller pieces. By erring on the side of minimizing fragmentation for the file system, files may become so non-contiguous that their I/O characteristics would degrade.

Fixed extent sizes are particularly appropriate in the following situations:

- If a file is large and sparse and its write size is fixed, a fixed extent size that is a multiple of the write size can minimize space wasted by blocks that do not contain user data as a result of misalignment of write and extent sizes. The default extent size for a sparse file is 8K.
- If a file is large and contiguous, a large fixed extent size can minimize the number of extents in the file.

Custom applications may also use fixed extent sizes for specific reasons, such as the need to align extents to cylinder or striping boundaries on disk.

How the fixed extent size works with the shared extents

Veritas File System (VxFS) allows the user to set the fixed extent size option on a file that controls the minimum allocation size of the file. If a file has shared extents that must be unshared, the allocation that is done as a part of the unshare operation ignores the fixed extent size option that is set on the file. The allocation size during the unshare operation, is dependent on the size of the write operation on the shared region.

Other extent attribute controls

The auxiliary controls on extent attributes determine the following conditions:

- Whether allocations are aligned
  See “Extent attribute alignment” on page 188.
- Whether allocations are contiguous
  See “Extent attribute contiguity” on page 188.
- Whether the file can be written beyond its reservation
  See “Write operations beyond extent attribute reservation” on page 188.
- Whether an unused reservation is released when the file is closed
  See “Extent attribute reservation trimming” on page 188.
- Whether the reservation is a persistent attribute of the file
  See “Extent attribute reservation persistence” on page 188.
- When the space reserved for a file will actually become part of the file
  See “Including an extent attribute reservation in the file” on page 188.

Extent attribute alignment

Specific alignment restrictions coordinate a file's allocations with a particular I/O pattern or disk alignment. Alignment can only be specified if a fixed extent size has also been set. Setting alignment restrictions on allocations is best left to well-designed applications.

See the `setext`(1) manual page.

See "About Veritas File System I/O" on page 332.

Extent attribute contiguity

A reservation request can specify that its allocation remain contiguous (all one extent). Maximum contiguity of a file optimizes its I/O characteristics.

Note: Fixed extent sizes or alignment cause a file system to return an error message reporting insufficient space if no suitably sized (or aligned) extent is available. This can happen even if the file system has sufficient free space and the fixed extent size is large.

Write operations beyond extent attribute reservation

A reservation request can specify that no allocations can take place after a write operation fills the last available block in the reservation. This request can be used a way similar to the function of the `ulimit` command to prevent a file's uncontrolled growth.

Extent attribute reservation trimming

A reservation request can specify that any unused reservation be released when the file is closed. The file is not completely closed until all processes open against the file have closed it.

Extent attribute reservation persistence

A reservation request can ensure that the reservation does not become a persistent attribute of the file. The unused reservation is discarded when the file is closed.

Including an extent attribute reservation in the file

A reservation request can make sure the size of the file is adjusted to include the reservation. Normally, the space of the reservation is not included in the file until an extending write operation requires it. A reservation that immediately changes the file size can generate large temporary files. Unlike a ftruncate operation that increases the size of a file, this type of reservation does not perform zeroing of the blocks included in the file and limits this facility to users with appropriate privileges. The data that appears in the file may have been previously contained in another file. For users who do not have the appropriate privileges, there is a variant request that prevents such users from viewing uninitialized data.

Commands related to extent attributes

The Veritas File System (VxFS) commands for manipulating extent attributes are `setext` and `getext`; they allow the user to set up files with a given set of extent attributes or view any attributes that are already associated with a file.

See the `setext`(1) and `getext`(1) manual pages.

The VxFS-specific commands `vxdump` and `vxrestore` preserve extent attributes when backing up, restoring, moving, or copying files.

Most of these commands include a command-line option (`-e`) for maintaining extent attributes on files. You use this option with a VxFS file that has extent attribute information including reserved space, a fixed extent size, and extent alignment. The extent attribute information may be lost if the destination file system does not support extent attributes, has a different block size than the source file system, or lacks free extents appropriate to satisfy the extent attribute requirements.

The `-e` option takes any of the following keywords as an argument:

`warn`	Issues a warning message if extent attribute information cannot be maintained (the default)
`force`	Fails the copy if extent attribute information cannot be maintained
`ignore`	Ignores extent attribute information entirely

The following example creates a file named `file1` and preallocates 2 GB of disk space for the file.

Example of setting an extent attribute

1. Create the file `file1`:

   ```
   # touch file1
   ```

2. Preallocate 2 GB of disk space for the file `file1`:

   ```
   # setext -t vxfs -r 2g -f chgsize file1
   ```

   Since the example specifies the `-f chgsize` option, VxFS immediately incorporates the reservation into the file and updates the file's inode with size and block count information that is increased to include the reserved space. Only users with root privileges can use the `-f chgsize` option.

The following example gets the extent atribute information of a file named `file1`.

Example of getting an extent attribute's information

◆ Get the extent attribute information for the file `file1`:

```
  # getext -t vxfs file1
 file1: Bsize  1024  Reserve 2097152  Extent Size  0
```

The file `file1` has a block size of 1024 bytes, 36 blocks reserved, a fixed extent size of 3 blocks, and all extents aligned to 3 block boundaries. The file size cannot be increased after the current reservation is exhausted. Reservations and fixed extent sizes are allocated in units of the file system block size.

About failing to preserve extent attributes

Whenever a file is copied, moved, or archived using commands that preserve extent attributes, there is the possibility of losing the attributes.

Such a failure might occur for one of the following reasons:

- The file system receiving a copied, moved, or restored file from an archive is not a VxFS file system. Since other file system types do not support the extent attributes of the VxFS file system, the attributes of the source file are lost during the migration.
- The file system receiving a copied, moved, or restored file is a VxFS type but does not have enough free space to satisfy the extent attributes. For example, consider a 50 KB file and a reservation of 1 MB. If the target file system has 500 KB free, it could easily hold the file but fail to satisfy the reservation.
- The file system receiving a copied, moved, or restored file from an archive is a VxFS type but the different block sizes of the source and target file system make extent attributes impossible to maintain. For example, consider a source file

system of block size 1024, a target file system of block size 4096, and a file that has a fixed extent size of 3 blocks (3072 bytes). This fixed extent size adapts to the source file system but cannot translate onto the target file system.

The same source and target file systems in the preceding example with a file carrying a fixed extent size of 4 could preserve the attribute; a 4 block (4096 byte) extent on the source file system would translate into a 1 block extent on the target.

On a system with mixed block sizes, a copy, move, or restoration operation may or may not succeed in preserving attributes. It is recommended that the same block size be used for all file systems on a given system.

Section 3

Administering multi-pathing with DMP

Chapter 9

Administering Dynamic Multi-Pathing

This chapter includes the following topics:

Discovering and configuring newly added disk devices

When you physically connect new disks to a host or when you zone new Fibre Channel devices to a host, you can use the `vxdctl enable` command to rebuild the volume device node directories and to update the Dynamic Multi-Pathing (DMP) internal database to reflect the new state of the system.

To reconfigure the DMP database, first make Linux recognize the new disks, and then invoke the `vxdctl enable` command.

You can also use the `vxdisk scandisks` command to scan devices in the operating system device tree, and to initiate dynamic reconfiguration of multipathed disks.

If you want SF to scan only for new devices that have been added to the system, and not for devices that have been enabled or disabled, specify the `-f` option to either of the commands, as shown here:

```
# vxdctl -f enable
# vxdisk -f scandisks
```

However, a complete scan is initiated if the system configuration has been modified by changes to:

- Installed array support libraries.
- The list of devices that are excluded from use by VxVM.
- DISKS (JBOD), SCSI3, or foreign device definitions.

See the `vxdctl`(1M) manual page.

See the `vxdisk`(1M) manual page.

Partial device discovery

Dynamic Multi-Pathing (DMP) supports partial device discovery where you can include or exclude paths to a physical disk from the discovery process.

The `vxdisk scandisks` command rescans the devices in the OS device tree and triggers a DMP reconfiguration. You can specify parameters to `vxdisk scandisks` to implement partial device discovery. For example, this command makes SF discover newly added devices that were unknown to it earlier:

```
# vxdisk scandisks new
```

The next example discovers fabric devices:

```
# vxdisk scandisks fabric
```

The following command scans for the devices `sdm` and `sdn`:

```
# vxdisk scandisks device=sdm,sdn
```

Alternatively, you can specify a ! prefix character to indicate that you want to scan for all devices except those that are listed.

Note: The ! character is a special character in some shells. The following examples show how to escape it in a bash shell.

```
# vxdisk scandisks \!device=sdm,sdn
```

You can also scan for devices that are connected (or not connected) to a list of logical or physical controllers. For example, this command discovers and configures all devices except those that are connected to the specified logical controllers:

```
# vxdisk scandisks \!ctlr=c1,c2
```

The next command discovers only those devices that are connected to the specified physical controller:

```
# vxdisk scandisks pctlr=c1+c2
```

The items in a list of physical controllers are separated by + characters.

You can use the command `vxdmpadm getctlr all` to obtain a list of physical controllers.

You should specify only one selection argument to the `vxdisk scandisks` command. Specifying multiple options results in an error.

See the `vxdisk`(1M) manual page.

About discovering disks and dynamically adding disk arrays

Dynamic Multi-Pathing (DMP) uses array support libraries (ASLs) to provide array-specific support for multi-pathing. An array support library (ASL) is a dynamically loadable shared library (plug-in for DDL). The ASL implements hardware-specific logic to discover device attributes during device discovery. DMP provides the device discovery layer (DDL) to determine which ASLs should be associated to each disk array.

In some cases, DMP can also provide basic multi-pathing and failover functionality by treating LUNs as disks (JBODs).

How DMP claims devices

For fully optimized support of any array and for support of more complicated array types, Dynamic Multi-Pathing (DMP) requires the use of array-specific array support libraries (ASLs), possibly coupled with array policy modules (APMs). ASLs and APMs effectively are array-specific plug-ins that allow close tie-in of DMP with any specific array model.

Refer to the Hardware Compatibility List at:

https://www.veritas.com/content/support/en_US/doc/infoscale_hcl_8x_win

During device discovery, the DDL checks the installed ASL for each device to find which ASL claims the device.

If no ASL is found to claim the device, the DDL checks for a corresponding JBOD definition. You can add JBOD definitions for unsupported arrays to enable DMP to provide multi-pathing for the array. If a JBOD definition is found, the DDL claims the devices in the DISKS category, which adds the LUNs to the list of JBOD (physical disk) devices used by DMP. If the JBOD definition includes a cabinet number, DDL uses the cabinet number to group the LUNs into enclosures.

See "Adding unsupported disk arrays to the DISKS category" on page 206.

DMP can provide basic multi-pathing to arrays that comply with the Asymmetric Logical Unit Access (ALUA) standard, even if there is no ASL or JBOD definition. DDL claims the LUNs as part of the aluadisk enclosure. The array type is shown as ALUA. Adding a JBOD definition also enables you to group the LUNs into enclosures.

Disk categories

Disk arrays that have been certified for use with Dynamic Multi-Pathing (DMP) are supported by an array support library (ASL), and are categorized by the vendor ID string that is returned by the disks (for example, "HITACHI").

Disks in JBODs that are capable of being multi-pathed by DMP, are placed in the DISKS category. Disks in unsupported arrays can also be placed in the DISKS category.

See "Adding unsupported disk arrays to the DISKS category" on page 206.

Disks in JBODs that do not fall into any supported category, and which are not capable of being multi-pathed by DMP are placed in the OTHER_DISKS category.

Adding DMP support for a new disk array

You can dynamically add support for a new type of disk array. The support comes in the form of Array Support Libraries (ASLs) that are developed by Veritas. Veritas provides support for new disk arrays through updates to the VRTSaslapm rpm. To determine if an updated VRTSaslapm rpm is available for download, refer to the hardware compatibility list (HCL). The hardware compatibility list provides a link to the latest rpm for download and instructions for installing the VRTSaslapm rpm. You can upgrade the VRTSaslapm rpm while the system is online; you do not need to stop the applications.

Refer to the hardware compatibility list at:

https://www.veritas.com/content/support/en_US/doc/infoscale_hcl_8x_unix

Each VRTSaslapm rpm is specific for the Storage Foundation version. Be sure to install the VRTSaslapm rpm that supports the installed version of Storage Foundation.

The new disk array does not need to be already connected to the system when the `VRTSaslapm` rpm is installed.

If you need to remove the latest `VRTSaslapm` rpm, you can revert to the previously installed version. For the detailed procedure, refer to the *Veritas InfoScale Troubleshooting Guide*.

Enabling discovery of new disk arrays

The `vxdctl enable` command scans all of the disk devices and their attributes, updates the SF device list, and reconfigures DMP with the new device database. There is no need to reboot the host.

Warning: This command ensures that Dynamic Multi-Pathing is set up correctly for the array. Otherwise, VxVM treats the independent paths to the disks as separate devices, which can result in data corruption.

To enable discovery of a new disk array

◆ Type the following command:

```
# vxdctl enable
```

About third-party driver coexistence

The third-party driver (TPD) coexistence feature of Storage Foundation (SF) allows I/O that is controlled by some third-party multi-pathing drivers to bypass Dynamic Multi-Pathing (DMP) while retaining the monitoring capabilities of DMP. If a suitable Array Support Library (ASL) is available and installed, devices that use TPDs can be discovered without requiring you to set up a specification file, or to run a special command. The TPD coexistence feature of SF permits coexistence without requiring any change in a third-party multi-pathing driver.

See “Changing device naming for enclosures controlled by third-party drivers” on page 290.

See “Displaying information about devices controlled by third-party drivers” on page 226.

How to administer the Device Discovery Layer

The Device Discovery Layer (DDL) allows dynamic addition of disk arrays. DDL discovers disks and their attributes that are required for Storage Foundation (SF) operations.

The DDL is administered using the `vxddladm` utility to perform the following tasks:

- List the hierarchy of all the devices discovered by DDL including iSCSI devices.
- List all the Host Bus Adapters including iSCSI.
- List the ports configured on a Host Bus Adapter.
- List the targets configured from a Host Bus Adapter.
- List the devices configured from a Host Bus Adapter.
- Get or set the iSCSI operational parameters.
- List the types of arrays that are supported.
- Add support for an array to DDL.
- Remove support for an array from DDL.
- List information about excluded disk arrays.
- List disks that are claimed in the `DISKS` (JBOD) category.
- Add disks from different vendors to the `DISKS` category.
- Remove disks from the `DISKS` category.
- Add disks as foreign devices.

The following sections explain these tasks in more detail.

See the `vxddladm`(1M) manual page.

Listing all the devices including iSCSI

You can display the hierarchy of all the devices discovered by DDL, including iSCSI devices.

To list all the devices including iSCSI

◆ Type the following command:

```
# vxddladm list
```

The following is a sample output:

```
HBA fscsi0 (20:00:00:E0:8B:19:77:BE)
    Port fscsi0_p0 (50:0A:09:80:85:84:9D:84)
       Target fscsi0_p0_t0 (50:0A:09:81:85:84:9D:84)
           Device sda
. . .
HBA iscsi0 (iqn.1986-03.com.sun:01:0003ba8ed1b5.45220f80)
    Port  iscsi0_p0 (10.216.130.10:3260)
       Target  iscsi0_p0_t0 (iqn.1992-08.com.netapp:sn.84188548)
           Device sdb
           Device sdc
       Target  iscsi0_p0_t1 (iqn.1992-08.com.netapp:sn.84190939)
. . .
```

Listing all the Host Bus Adapters including iSCSI

You can obtain information about all the Host Bus Adapters (HBAs) configured on the system, including iSCSI adapters.

Table 9-1 shows the HBA information.

Table 9-1 HBA information

Field	Description
Driver	Driver controlling the HBA.
Firmware	Firmware version.
Discovery	The discovery method employed for the targets.
State	Whether the device is Online or Offline.
Address	The hardware address.

To list all the Host Bus Adapters including iSCSI

◆ Use the following command to list all of the HBAs, including iSCSI devices, configured on the system:

```
# vxddladm list hbas
```

Listing the ports configured on a Host Bus Adapter

You can obtain information about all the ports configured on an HBA. The display includes the following information:

HBA-ID	The parent HBA.
State	Whether the device is Online or Offline.
Address	The hardware address.

To list the ports configured on a Host Bus Adapter

◆ Use the following command to obtain the ports configured on an HBA:

```
# vxddladm list ports

PORT-ID   HBA-ID     STATE       ADDRESS
-------------------------------------------------------
c2_p0     c2         Online      50:0A:09:80:85:84:9D:84
c3_p0     c3         Online      10.216.130.10:3260
```

Listing the targets configured from a Host Bus Adapter or a port

You can obtain information about all the targets configured from a Host Bus Adapter or a port.

Table 9-2 shows the target information.

Table 9-2 Target information

Field	Description
Alias	The alias name, if available.
HBA-ID	Parent HBA or port.
State	Whether the device is Online or Offline.
Address	The hardware address.

To list the targets

- To list all of the targets, use the following command:

```
# vxddladm list targets
```

The following is a sample output:

```
TARGET-ID  ALIAS HBA-ID STATE  ADDRESS
-----------------------------------------------------------------
c2_p0_t0   -     c2     Online 50:0A:09:80:85:84:9D:84
c3_p0_t1   -     c3     Online iqn.1992-08.com.netapp:sn.84190939
```

To list the targets configured from a Host Bus Adapter or port

- You can filter based on a HBA or port, using the following command:

```
# vxddladm list targets [hba=hba_name|port=port_name]
```

For example, to obtain the targets configured from the specified HBA:

```
# vxddladm list targets hba=c2
```

```
TARGET-ID  ALIAS  HBA-ID   STATE      ADDRES
--------------------------------------------------------------
c2_p0_t0   -      c2       Online     50:0A:09:80:85:84:9D:84
```

Listing the devices configured from a Host Bus Adapter and target

You can obtain information about all the devices configured from a Host Bus Adapter.

Table 9-3 shows the device information.

Table 9-3 Device information

Field	Description
Device	The device name.
Target-ID	The parent target.
State	Whether the device is Online or Offline.
DDL status	Whether the device is claimed by DDL. If claimed, the output also displays the ASL name.

To list the devices configured from a Host Bus Adapter

- To obtain the devices configured, use the following command:

```
# vxddladm list devices

Device    Target-ID    State    DDL status (ASL)
-------------------------------------------------------
sda       fscsi0_p0_t0 Online   CLAIMED (libvxemc.so)
sdb       fscsi0_p0_t0 Online   SKIPPED (libvxemc.so)
sdc       fscsi0_p0_t0 Offline  ERROR
sdd       fscsi0_p0_t0 Online   EXCLUDED
sde       fscsi0_p0_t0 Offline  MASKED
nvme0n1   -            Online   CLAIMED (libvxnvme_nonscsi.so)
```

To list the devices configured from a Host Bus Adapter and target

- To obtain the devices configured from a particular HBA and target, use the following command:

```
# vxddladm list devices target=target_name
```

Getting or setting the iSCSI operational parameters

DDL provides an interface to set and display certain parameters that affect the performance of the iSCSI device path. However, the underlying OS framework must support the ability to set these values. The `vxddladm set` command returns an error if the OS support is not available.

Table 9-4 Parameters for iSCSI devices

Parameter	Default value	Minimum value	Maximum value
DataPDUInOrder	yes	no	yes
DataSequenceInOrder	yes	no	yes
DefaultTime2Retain	20	0	3600
DefaultTime2Wait	2	0	3600
ErrorRecoveryLevel	0	0	2
FirstBurstLength	65535	512	16777215
InitialR2T	yes	no	yes
ImmediateData	yes	no	yes

Table 9-4 Parameters for iSCSI devices *(continued)*

Parameter	Default value	Minimum value	Maximum value
MaxBurstLength	262144	512	16777215
MaxConnections	1	1	65535
MaxOutStandingR2T	1	1	65535
MaxRecvDataSegmentLength	8182	512	16777215

To get the iSCSI operational parameters on the initiator for a specific iSCSI target

- Type the following commands:

  ```
  # vxddladm getiscsi target=tgt-id {all | parameter}
  ```

  You can use this command to obtain all the iSCSI operational parameters.

  ```
  # vxddladm getiscsi target=c2_p2_t0
  ```

  The following is a sample output:

  ```
  PARAMETER               CURRENT  DEFAULT    MIN      MAX
  --------------------------------------------------------
  DataPDUInOrder              yes      yes     no      yes
  DataSequenceInOrder         yes      yes     no      yes
  DefaultTime2Retain           20       20      0     3600
  DefaultTime2Wait              2        2      0     3600
  ErrorRecoveryLevel            0        0      0        2
  FirstBurstLength          65535    65535    512 16777215
  InitialR2T                  yes      yes     no      yes
  ImmediateData               yes      yes     no      yes
  MaxBurstLength           262144   262144    512 16777215
  MaxConnections                1        1      1    65535
  MaxOutStandingR2T             1        1      1    65535
  MaxRecvDataSegmentLength  8192      8182    512 16777215
  ```

To set the iSCSI operational parameters on the initiator for a specific iSCSI target

- Type the following command:

  ```
  # vxddladm setiscsi target=tgt-id parameter=value
  ```

Listing all supported disk arrays

Use this procedure to obtain values for the `vid` and `pid` attributes that are used with other forms of the `vxddladm` command.

To list all supported disk arrays

- Use the following command:

```
# vxddladm listsupport all
```

Displaying details about an Array Support Library

Dynamic Multi-Pathing (DMP) enables you to display details about the Array Support Libraries (ASL).

To display details about an Array Support Library

- Type the following command:

```
# vxddladm listsupport libname=library_name.so
```

This command displays the vendor IDs (`VIDs`), product IDs (`PIDs`) for the arrays, array types (for example, A/A or A/P), and array names. The following is sample output.

```
# vxddladm listsupport libname=libvxfujitsu.so
ATTR_NAME            ATTR_VALUE
=================================================
LIBNAME              libvxfujitsu.so
VID                  vendor
PID                  GR710, GR720, GR730
                     GR740, GR820, GR840
ARRAY_TYPE           A/A, A/P
ARRAY_NAME           FJ_GR710, FJ_GR720, FJ_GR730
                     FJ_GR740, FJ_GR820, FJ_GR840
```

Excluding support for a disk array library

You can exclude support for disk arrays that depends on a particular disk array library. You can also exclude support for disk arrays from a particular vendor.

To exclude support for a disk array library

- To exclude support for a disk array library, specify the array library to the following command.

  ```
  # vxddladm excludearray libname=libname
  ```

  You can also exclude support for disk arrays from a particular vendor, as shown in this example:

  ```
  # vxddladm excludearray vid=ACME pid=X1
  ```

  ```
  # vxdisk scandisks
  ```

Re-including support for an excluded disk array library

If you previously excluded support for all arrays that depend on a particular disk array library, use this procedure to include the support for those arrays. This procedure removes the library from the exclude list.

To re-include support for an excluded disk array library

- If you have excluded support for all arrays that depend on a particular disk array library, you can use the `includearray` keyword to remove the entry from the exclude list.

  ```
  # vxddladm includearray libname=libname
  ```

  This command adds the array library to the database so that the library can once again be used in device discovery.

  ```
  # vxdisk scandisks
  ```

Listing excluded disk arrays

To list all disk arrays that are currently excluded from use by Veritas Volume Manager (VxVM)

- Type the following command:

  ```
  # vxddladm listexclude
  ```

Listing disks claimed in the DISKS category

To list disks that are claimed in the `DISKS` (JBOD) category

◆ Type the following command:

```
# vxddladm listjbod
```

Adding unsupported disk arrays to the DISKS category

Disk arrays should be added as JBOD devices if no Array Support Library (ASL) is available for the array.

JBODs are assumed to be Active/Active (A/A) unless otherwise specified. If a suitable ASL is not available, an A/A-A, A/P, or A/PF array must be claimed as an Active/Passive (A/P) JBOD to prevent path delays and I/O failures. If a JBOD is ALUA-compliant, it is added as an ALUA array.

See “How DMP works” on page 34.

Warning: This procedure ensures that Dynamic Multi-Pathing (DMP) is set up correctly on an array that is not supported by Veritas Volume Manager (VxVM). Otherwise, VxVM treats the independent paths to the disks as separate devices, which can result in data corruption.

To add an unsupported disk array to the DISKS category

1. Use the following command to identify the vendor ID and product ID of the disks in the array:

   ```
   # /etc/vx/diag.d/vxscsiinq device_name
   ```

   where *device_name* is the device name of one of the disks in the array. Note the values of the vendor ID (`VID`) and product ID (`PID`) in the output from this command. For Fujitsu disks, also note the number of characters in the serial number that is displayed.

   The following example output shows that the vendor ID is `SEAGATE` and the product ID is `ST318404LSUN18G`.

   ```
   Vendor id (VID)      : SEAGATE
   Product id (PID)     : ST318404LSUN18G
   Revision             : 8507
   Serial Number        : 0025T0LA3H
   ```

2. Stop all applications, such as databases, from accessing VxVM volumes that are configured on the array, and unmount all file systems and Storage Checkpoints that are configured on the array.
3. If the array is of type A/A-A, A/P, or A/PF, configure it in autotrespass mode.
4. Enter the following command to add a new JBOD category:

   ```
   # vxddladm addjbod vid=vendorid [pid=productid] \
   [serialnum=opcode/pagecode/offset/length] \
   [cabinetnum=opcode/pagecode/offset/length] policy={aa|ap}]
   ```

   where *vendorid* and *productid* are the VID and PID values that you found from the previous step. For example, *vendorid* might be `FUJITSU`, `IBM`, or `SEAGATE`. For Fujitsu devices, you must also specify the number of characters in the serial number as the `length` argument (for example, 10). If the array is of type A/A-A, A/P, or A/PF, you must also specify the `policy=ap` attribute.

   Continuing the previous example, the command to define an array of disks of this type as a JBOD would be:

   ```
   # vxddladm addjbod vid=SEAGATE pid=ST318404LSUN18G
   ```

5. Use the `vxdctl enable` command to bring the array under VxVM control.

   ```
   # vxdctl enable
   ```

   See “Enabling discovery of new disk arrays” on page 197.

6 To verify that the array is now supported, enter the following command:

```
# vxddladm listjbod
```

The following is sample output from this command for the example array:

```
VID     PID           SerialNum            CabinetNum   Policy
                 (Cmd/PageCode/off/len) (Cmd/PageCode/off/len)
==============================================================
SEAGATE ALL PIDs  18/-1/36/12           18/-1/10/11    Disk
SUN     SESS01    18/-1/36/12           18/-1/12/11    Disk
```

7. To verify that the array is recognized, use the `vxdmpadm listenclosure` command as shown in the following sample output for the example array:

```
# vxdmpadm listenclosure
```

```
ENCLR_NAME ENCLR_TYPE ENCLR_SNO STATUS    ARRAY_TYPE LUN_COUNT FIRMWARE
=======================================================================
Disk       Disk       DISKS     CONNECTED Disk       2          -
```

The enclosure name and type for the array are both shown as being set to `Disk`. You can use the `vxdisk list` command to display the disks in the array:

```
# vxdisk list
DEVICE       TYPE            DISK         GROUP        STATUS
punr710vm04_disk_1 auto:none       -            -            online invalid
punr710vm04_disk_2 auto:none       -            -            online invalid
punr710vm04_disk_3 auto:none       -            -            online invalid
punr710vm04_disk_4 auto:none       -            -            online invalid
sda                auto:none       -            -            online invalid
xiv0_9148          auto:none       -            -            online invalid thinrclm
...
```

8. To verify that the DMP paths are recognized, use the `vxdmpadm getdmpnode` command as shown in the following sample output for the example array:

```
# vxdmpadm getdmpnode enclosure=Disk
NAME                 STATE        ENCLR-TYPE   PATHS  ENBL  DSBL  ENCLR-NAME
==============================================================================
punr710vm04_disk_1   ENABLED      Disk         1      1     0     disk
punr710vm04_disk_2   ENABLED      Disk         1      1     0     disk
punr710vm04_disk_3   ENABLED      Disk         1      1     0     disk
punr710vm04_disk_4   ENABLED      Disk         1      1     0     disk
sda                  ENABLED      Disk         1      1     0     disk
...
```

The output in this example shows that there are two paths to the disks in the array.

For more information, enter the command `vxddladm help addjbod`.

See the `vxddladm`(1M) manual page.

See the `vxdmpadm`(1M) manual page.

Removing disks from the DISKS category

Use the procedure in this section to remove disks from the DISKS category.

To remove disks from the DISKS category

- ◆ Use the vxddladm command with the rmjbod keyword. The following example illustrates the command for removing disks that have the vendor id of SEAGATE:

```
# vxddladm rmjbod vid=SEAGATE
```

Foreign devices

The Device Discovery Layer (DDL) may not be able to discover some devices that are not auto-discoverable, such as RAM disks. Such foreign devices can be made available as simple disks to Veritas Volume Manager (VxVM) by using the vxddladm addforeign command. This also has the effect of bypassing DMP for handling I/O. The following example shows how to add entries for block and character devices in the specified directories:

```
# vxddladm addforeign blockdir=/dev/foo/dsk chardir=/dev/foo/rdsk
```

If a block or character device is not supported by a driver, it can be omitted from the command as shown here:

```
# vxddladm addforeign blockdir=/dev/foo/dsk
```

By default, this command suppresses any entries for matching devices in the OS-maintained device tree that are found by the autodiscovery mechanism. You can override this behavior by using the -f and -n options as described on the vxddladm(1M) manual page.

After adding entries for the foreign devices, use either the vxdisk scandisks or the vxdctl enable command to discover the devices as simple disks. These disks then behave in the same way as autoconfigured disks.

Foreign device support has the following limitations:

- A foreign device is always considered as a disk with a single path. Unlike an autodiscovered disk, it does not have a DMP node.
- It is not supported for shared disk groups in a clustered environment. Only standalone host systems are supported.
- It is not supported for Persistent Group Reservation (PGR) operations.
- It is not under the control of DMP, so enabling of a failed disk cannot be automatic, and DMP administrative commands are not applicable.
- Enclosure information is not available to VxVM. This can reduce the availability of any disk groups that are created using such devices.

- The I/O fencing and Cluster File System features are not supported for foreign devices.

Making devices invisible to VxVM

Use this procedure to exclude a device from the view of Veritas Volume Manager (VxVM). The options to prevent a device from being multi-pathed by the Dynamic Multi-Pathing (DMP) driver (`vxdmp`) are deprecated.

To make devices invisible to VxVM

1. Run the `vxdiskadm` command, and select `Prevent multipathing/Suppress devices from VxVM’s view` from the main menu. You are prompted to confirm whether you want to continue.

2. Select the operation you want to perform from the following options:

Option 1	Suppresses all paths through the specified controller from the view of VxVM.
Option 2	Suppresses specified paths from the view of VxVM.
Option 3	Suppresses disks from the view of VxVM that match a specified Vendor ID and Product ID combination. The root disk cannot be suppressed. The operation fails if the VID:PID of an external disk is the same VID:PID as the root disk and the root disk is encapsulated under VxVM.
Option 4	Suppresses all paths to a disk.
Option 5 **Deprecated**	Prevents multi-pathing for all disks on a specified controller by VxVM. This operation is deprecated, since it can lead to unsupported configurations.
Option 6 **Deprecated**	Prevents multi-pathing of a disk by VxVM. The disks that correspond to a specified path are claimed in the OTHER_DISKS category and are not multi-pathed. This operation is deprecated, since it can lead to unsupported configurations.
Option 7 **Deprecated**	Prevents multi-pathing for the disks that match a specified Vendor ID and Product ID combination. The disks that correspond to a specified Vendor ID and Product ID combination are claimed in the OTHER_DISKS category and are not multi-pathed. This operation is deprecated, since it can lead to unsupported configurations.
Option 8	Lists the devices that are currently suppressed.

Making devices visible to VxVM

Use this procedure to make a device visible to Veritas Volume Manager (VxVM) again. The options to allow multi-pathing by the Dynamic Multi-Pathing (DMP) driver (`vxdmp`) are deprecated.

To make devices visible to VxVM

1. Run the `vxdiskadm` command, and select `Allow multipathing/Unsuppress devices from VxVM’s view` from the main menu. You are prompted to confirm whether you want to continue.
2. Select the operation you want to perform from the following options:

Option 1	Unsuppresses all paths through the specified controller from the view of VxVM.
Option 2	Unsuppresses specified paths from the view of VxVM.
Option 3	Unsuppresses disks from the view of VxVM that match a specified Vendor ID and Product ID combination.
Option 4	Unsuppresses all paths to a disk.
Option 5 **Deprecated**	Allows multi-pathing of all disks that have paths through the specified controller. This operation is deprecated.
Option 6 **Deprecated**	Allows multi-pathing of a disk by VxVM. This operation is deprecated.
Option 7 **Deprecated**	Allows multi-pathing of disks that match a specified Vendor ID and Product ID combination. This operation is deprecated.
Option 8	Lists the devices that are currently suppressed.

About enabling and disabling I/O for controllers and storage processors

DMP allows you to turn off I/O through a Host Bus Adapter (HBA) controller or the array port of a storage processor so that you can perform administrative operations. This feature can be used when you perform maintenance on HBA controllers on the host, or array ports that are attached to disk arrays supported by SF. I/O operations to the HBA controller or the array port can be turned back on after the maintenance task is completed. You can accomplish these operations using the `vxdmpadm` command.

For Active/Active type disk arrays, when you disable the I/O through an HBA controller or array port, the I/O continues on the remaining paths. For Active/Passive type disk arrays, if disabling I/O through an HBA controller or array port resulted in all primary paths being disabled, DMP will failover to secondary paths and I/O will continue on them.

After the administrative operation is over, use the `vxdmpadm` command to re-enable the paths through the HBA controllers or array ports.

See "Disabling I/O for paths, controllers, array ports, or DMP nodes" on page 246.

See "Enabling I/O for paths, controllers, array ports, or DMP nodes" on page 248.

You can also perform certain reconfiguration operations dynamically online.

About displaying DMP database information

You can use the `vxdmpadm` command to list DMP database information and perform other administrative tasks. This command allows you to list all controllers that are connected to disks, and other related information that is stored in the DMP database. You can use this information to locate system hardware, and to help you decide which controllers need to be enabled or disabled.

The `vxdmpadm` command also provides useful information such as disk array serial numbers, which DMP devices (disks) are connected to the disk array, and which paths are connected to a particular controller, enclosure, or array port.

See "Administering DMP using the vxdmpadm utility" on page 217.

Displaying the paths to a disk

The `vxdisk` command is used to display the multi-pathing information for a particular metadevice. The metadevice is a device representation of a physical disk having multiple physical paths through the system's HBA controllers. In Dynamic Multi-Pathing (DMP,) all the physical disks in the system are represented as metadevices with one or more physical paths.

To display the multi-pathing information on a system

- Use the `vxdisk path` command to display the relationships between the device paths, disk access names, disk media names, and disk groups on a system as shown here:

  ```
  # vxdisk path

  SUBPATH      DANAME     DMNAME     GROUP    STATE
  sda          sda        mydg01     mydg     ENABLED
  sdi          sdi        mydg01     mydg     ENABLED
  sdb          sdb        mydg02     mydg     ENABLED
  sdj          sdj        mydg02     mydg     ENABLED
   .
   .
   .
  ```

  This shows that two paths exist to each of the two disks, `mydg01` and `mydg02`, and also indicates that each disk is in the `ENABLED` state.

To view multi-pathing information for a particular metadevice

1 Use the following command:

```
# vxdisk list devicename
```

For example, to view multi-pathing information for the device `sdl`, use the following command:

```
# vxdisk list sdl
```

The output from the `vxdisk list` command displays the multi-pathing information, as shown in the following example:

```
Device:    sdl
devicetag: sdl
type:      sliced
hostid:    sys1
.
.
.
Multipathing information:
numpaths:   2
sdl   state=enabled     type=primary
sdp   state=disabled    type=secondary
```

The `numpaths` line shows that there are 2 paths to the device. The next two lines in the "Multipathing information" section of the output show that one path is active (`state=enabled`) and that the other path has failed (`state=disabled`).

The `type` field is shown for disks on Active/Passive type disk arrays such as the EMC CLARiiON, Hitachi HDS 9200 and 9500, Sun StorEdge 6xxx, and Sun StorEdge T3 array. This field indicates the primary and secondary paths to the disk.

The `type` field is not displayed for disks on Active/Active type disk arrays such as the EMC Symmetrix, Hitachi HDS 99xx and Sun StorEdge 99xx Series, and IBM ESS Series. Such arrays have no concept of primary and secondary paths.

2 Alternately, you can use the following command to view multi-pathing information:

```
# vxdmpadm getsubpaths dmpnodename=devicename
```

For example, to view multi-pathing information for `emc_clariion0_431`, use the following command:

```
# # vxdmpadm getsubpaths dmpnodename=emc_clariion0_431
```

Typical output from the `vxdmpadm getsubpaths` command is as follows:

```
NAME   STATE[A]    PATH-TYPE[M]         CTLR-NAME  ENCLR-TYPE    ENCLR-NAME    ATTRS PRIORITY
=============================================================================================
sdac   ENABLED     Active/Non-Optimized  c6        EMC_CLARiiON  emc_clariion0  -       -
sdam   ENABLED(A)  Active/Optimized(P)   c6        EMC_CLARiiON  emc_clariion0  -       -
sdi    ENABLED     Active/Non-Optimized  c1        EMC_CLARiiON  emc_clariion0  -       -
sds    ENABLED(A)  Active/Optimized(P)   c1        EMC_CLARiiON  emc_clariion0  -       -
```

Administering DMP using the vxdmpadm utility

The `vxdmpadm` utility is a command-line administrative interface to Dynamic Multi-Pathing (DMP).

You can use the `vxdmpadm` utility to perform the following tasks:

- Retrieve the name of the DMP device corresponding to a particular path.
  See "Retrieving information about a DMP node" on page 219.
- Display consolidated information about the DMP nodes.
  See "Displaying consolidated information about the DMP nodes" on page 220.
- Display the members of a LUN group.
  See "Displaying the members of a LUN group" on page 221.
- List all paths under a DMP device node, HBA controller, enclosure, or array port.
  See "Displaying paths controlled by a DMP node, controller, enclosure, or array port" on page 221.
- Display information about the HBA controllers on the host.
  See "Displaying information about controllers" on page 224.
- Display information about enclosures.
  See "Displaying information about enclosures" on page 225.

- Display information about array ports that are connected to the storage processors of enclosures.

  See "Displaying information about array ports" on page 226.
- Display asymmetric access state for ALUA arrays.
- Display information about devices that are controlled by third-party multi-pathing drivers.

  See "Displaying information about devices controlled by third-party drivers" on page 226.
- Display extended devices attributes.

  See "Displaying extended device attributes" on page 227.
- Suppress or include devices from VxVM control.

  See "Suppressing or including devices from VxVM control" on page 230.
- Gather I/O statistics for a DMP node, enclosure, path, or controller.

  See "Gathering and displaying I/O statistics" on page 230.
- Configure the attributes of the paths to an enclosure.

  See "Setting the attributes of the paths to an enclosure" on page 237.
- Display the redundancy level of a device or enclosure.

  See "Displaying the redundancy level of a device or enclosure" on page 238.
- Specify the minimum number of active paths.

  See "Specifying the minimum number of active paths" on page 239.
- Display or set the I/O policy that is used for the paths to an enclosure.

  See "Specifying the I/O policy" on page 240.
- Enable or disable I/O for a path, HBA controller or array port on the system.

  See "Disabling I/O for paths, controllers, array ports, or DMP nodes" on page 246.
- Rename an enclosure.

  See "Renaming an enclosure" on page 249.
- Configure how DMP responds to I/O request failures.

  See "Configuring the response to I/O failures" on page 249.
- Configure the I/O throttling mechanism.

  See "Configuring the I/O throttling mechanism" on page 251.
- Control the operation of the DMP path restoration thread.

  See "Configuring DMP path restoration policies" on page 254.
- Configure array policy modules.

  See "Configuring Array Policy Modules" on page 256.
- Get or set the values of various tunables used by DMP.

See “DMP tunable parameters” on page 796.

See the vxdmpadm(1M) manual page.

Retrieving information about a DMP node

The following command displays the Dynamic Multi-Pathing (DMP) node that controls a particular physical path:

```
# vxdmpadm getdmpnode nodename=pathname
```

The physical path is specified by argument to the nodename attribute, which must be a valid path listed in the device directory.

The device directory is the /dev directory.

The command displays output similar to the following example output.

```
# vxdmpadm getdmpnode nodename=sdbc

NAME              STATE    ENCLR-TYPE   PATHS ENBL DSBL ENCLR-NAME
===================================================================
emc_clariion0_89 ENABLED EMC_CLARiiON 6     6    0    emc_clariion0
```

Use the -v option to display the LUN serial number and the array volume ID.

```
# vxdmpadm -v getdmpnode nodename=sdbc
```

```
NAME             STATE   ENCLR-TYPE   PATHS ENBL DSBL ENCLR-NAME SERIAL-NO ARRAY_VOL_ID
=====================================================================================
emc_clariion0_89 ENABLED EMC_CLARiiON 6    6    0    emc_clariion0 600601601 893
```

Use the enclosure attribute with getdmpnode to obtain a list of all DMP nodes for the specified enclosure.

```
# vxdmpadm getdmpnode enclosure=emc_clariion0
```

```
NAME              STATE    ENCLR-TYPE   PATHS  ENBL  DSBL  ENCLR-NAME
=========================================================================
emc_clariion0_429  ENABLED  EMC_CLARiiON 4      4     0     emc_clariion0
emc_clariion0_430  ENABLED  EMC_CLARiiON 4      4     0     emc_clariion0
emc_clariion0_431  ENABLED  EMC_CLARiiON 4      4     0     emc_clariion0
emc_clariion0_432  ENABLED  EMC_CLARiiON 4      4     0     emc_clariion0
```

Use the dmpnodename attribute with getdmpnode to display the DMP information for a given DMP node.

```
# vxdmpadm getdmpnode dmpnodename=emc_clariion0_158
```

```
NAME              STATE    ENCLR-TYPE    PATHS ENBL DSBL ENCLR-NAME
==================================================================
emc_clariion0_158 ENABLED EMC_CLARiiON 1      1    0    emc_clariion0
```

Displaying consolidated information about the DMP nodes

The `vxdmpadm list dmpnode` command displays the detail information of a Dynamic Multi-Pathing (DMP) node. The information includes the enclosure name, LUN serial number, port id information, device attributes, and so on.

The following command displays the consolidated information for all of the DMP nodes in the system:

```
# vxdmpadm list dmpnode all
```

Use the `enclosure` attribute with `list dmpnode` to obtain a list of all DMP nodes for the specified enclosure.

```
# vxdmpadm list dmpnode enclosure=enclosurename
```

For example, the following command displays the consolidated information for all of the DMP nodes in the `enc0` enclosure.

```
# vxdmpadm list dmpnode enclosure=enc0
```

Use the `dmpnodename` attribute with `list dmpnode` to display the DMP information for a given DMP node. The DMP node can be specified by name or by specifying a path name. The detailed information for the specified DMP node includes path information for each subpath of the listed DMP node.

The path state differentiates between a path that is disabled due to a failure and a path that has been manually disabled for administrative purposes. A path that has been manually disabled using the `vxdmpadm disable` command is listed as `disabled(m)`.

```
# vxdmpadm list dmpnode dmpnodename=dmpnodename
```

For example, the following command displays the consolidated information for the DMP node `emc_clariion0_158`.

```
# vxdmpadm list dmpnode dmpnodename=emc_clariion0_158
```

```
dmpdev      = emc_clariion0_158
state       = enabled
enclosure   = emc_clariion0
cab-sno     = CK200070400359
asl         = libvxCLARiiON.so
```

```
vid         = DGC
pid         = DISK
array-name  = EMC_CLARiiON
array-type  = CLR-A/PF
iopolicy    = MinimumQ
avid        = 158
lun-sno     = 600601601A141B001D4A32F92B49DE11
udid        = DGC%5FDISK%5FCK200070400359%5F600601601A141B001D4A32F92B49DE11
dev-attr    = lun
###path     = name state type transport ctlr hwpath aportID aportWWN attr
path        = sdck enabled(a) primary FC c2 c2 A5 50:06:01:61:41:e0:3b:33 -
path        = sdde enabled(a) primary FC c2 c2 A4 50:06:01:60:41:e0:3b:33 -
path        = sdcu enabled secondary FC c2 c2 B4 50:06:01:68:41:e0:3b:33 -
path        = sdbm enabled secondary FC c3 c3 B4 50:06:01:68:41:e0:3b:33 -
path        = sdbw enabled(a) primary FC c3 c3 A4 50:06:01:60:41:e0:3b:33 -
path        = sdbc enabled(a) primary FC c3 c3 A5 50:06:01:61:41:e0:3b:33 -
```

Displaying the members of a LUN group

The following command displays the Dynamic Multi-Pathing (DMP) nodes that are in the same LUN group as a specified DMP node:

```
# vxdmpadm getlungroup dmpnodename=dmpnode
```

For example:

```
# vxdmpadm getlungroup dmpnodename=sdq

NAME     STATE    ENCLR-TYPE  PATHS     ENBL    DSBL   ENCLR-NAME
===============================================================
sdo      ENABLED  ACME         2         2       0       enc1
sdp      ENABLED  ACME         2         2       0       enc1
sdq      ENABLED  ACME         2         2       0       enc1
sdr      ENABLED  ACME         2         2       0       enc1
```

Displaying paths controlled by a DMP node, controller, enclosure, or array port

The `vxdmpadm getsubpaths` command lists all of the paths known to Dynamic Multi-Pathing (DMP). The `vxdmpadm getsubpaths` command also provides options to list the subpaths through a particular DMP node, controller, enclosure, or array port. To list the paths through an array port, specify either a combination of enclosure name and array port id, or array port worldwide name (WWN).

To list all subpaths known to DMP:

```
# vxdmpadm getsubpaths
```

```
NAME   STATE[A]   PATH-TYPE[M]         DMPNODENAME       ENCLR-NAME   CTLR   ATTRS
=============================================================================
sdaf   ENABLED(A) PRIMARY              ams_wms0_130      ams_wms0      c2        -
sdc    ENABLED    SECONDARY            ams_wms0_130      ams_wms0      c3        -
sdb    ENABLED(A)   -                  vm04_disk_24      disk          c0        -
sda    ENABLED(A)   -                  vm04_disk_25      disk          c0        -
sdaa   ENABLED    Active/Non-Optimized emc_clariion0_438 emc_clariion0 c1        -
sdak   ENABLED(A) Active/Optimized(P)  emc_clariion0_438 emc_clariion0 c6        -
```

The `vxdmpadm getsubpaths` command combined with the `dmpnodename` attribute displays all the paths to a LUN that are controlled by the specified DMP node name from the `/dev/vx/dmp` directory:

```
# vxdmpadm getsubpaths dmpnodename=sdby
```

```
NAME   STATE[A]   PATH-TYPE[M] CTLR-NAME  ENCLR-TYPE  ENCLR-NAME  ATTRS   PRIORITY
==================================================================================
sdbp   ENABLED      -          c1          EMC        emc0           -       -
sdbs   ENABLED      -          c1          EMC        emc0           -       -
sdbv   ENABLED      -          c6          EMC        emc0           -       -
sdby   ENABLED(A)   -          c6          EMC        emc0           -       -
```

For A/A arrays, all enabled paths that are available for I/O are shown as `ENABLED(A)`.

For A/P arrays in which the I/O policy is set to `singleactive`, only one path is shown as `ENABLED(A)`. The other paths are enabled but not available for I/O. If the I/O policy is not set to `singleactive`, DMP can use a group of paths (all primary or all secondary) for I/O, which are shown as `ENABLED(A)`.

See “Specifying the I/O policy” on page 240.

Paths that are in the DISABLED state are not available for I/O operations.

A path that was manually disabled by the system administrator displays as DISABLED(M). A path that failed displays as DISABLED.

You can use `getsubpaths` to obtain information about all the paths that are connected to a particular HBA controller:

```
# vxdmpadm getsubpaths ctlr=c1
```

```
NAME STATE[A]   PATH-TYPE[M]         DMPNODENAME        ENCLR-TYPE   ENCLR-NAME     ATTRS  PRIOR
================================================================================================
sdh  ENABLED    Active/Non-Optimized emc_clariion0_429  EMC_CLARiiON  emc_clariion0     -
sdr  ENABLED(A) Active/Optimized(P)  emc_clariion0_429  EMC_CLARiiON  emc_clariion0     -
sdm  ENABLED(A) Active/Optimized(P)  emc_clariion0_430  EMC_CLARiiON  emc_clariion0     -
sdw  ENABLED    Active/Non-Optimized emc_clariion0_430  EMC_CLARiiON  emc_clariion0     -
```

You can also use `getsubpaths` to obtain information about all the paths that are connected to a port on an array. The array port can be specified by the name of the enclosure and the array port ID, or by the WWN identifier of the array port:

```
# vxdmpadm getsubpaths enclosure=enclosure portid=portid
# vxdmpadm getsubpaths pwwn=pwwn
```

For example, to list subpaths through an array port through the enclosure and the array port ID:

```
# vxdmpadm getsubpaths enclosure=emc_clariion0 portid=A7

NAME STATE[A]   PATH-TYPE[M]         DMPNODENAME       ENCLR-NAME   CTLR ATTRS PRIORITY
=========================================================================================
sdal ENABLED(A) Active/Optimized     emc_clariion0_429 emc_clariion0 c6   -      -
sdr  ENABLED(A) Active/Optimized     emc_clariion0_429 emc_clariion0 c1   -      -
sdaq ENABLED    Active/Non-Optimized emc_clariion0_430 emc_clariion0 c6   -      -
sdw  ENABLED    Active/Non-Optimized emc_clariion0_430 emc_clariion0 c1   -      -
```

For example, to list subpaths through an array port through the WWN:

```
# vxdmpadm getsubpaths pwwn=50:06:01:67:3e:a0:75:95

NAME STATE[A]   PATH-TYPE[M]         CTLR-NAME ENCLR-TYPE   ENCLR-NAME     ATTRS PRIORITY
===========================================================================================
sdal ENABLED(A) Active/Optimized     c6        EMC_CLARiiON  emc_clariion0  -       -
sdr  ENABLED(A) Active/Optimized(P)  c1        EMC_CLARiiON  emc_clariion0  -       -
sdaq ENABLED    Active/Non-Optimized c6        EMC_CLARiiON  emc_clariion0  -       -
sdw  ENABLED    Active/Non-Optimized c1        EMC_CLARiiON  emc_clariion0  -       -
```

```
# vxdmpadm getsubpaths pwwn=20:00:00:E0:8B:06:5F:19
```

You can use `getsubpaths` to obtain information about all the subpaths of an enclosure.

```
# vxdmpadm getsubpaths enclosure=enclosure_name [ctlr=ctlrname]
```

To list all subpaths of an enclosure:

```
# vxdmpadm getsubpaths enclosure=emc_clariion0
NAME          STATE[A]   PATH-TYPE[M] DMPNODENAME  ENCLR-NAME   CTLR    ATTRS
================================================================================
sdav          ENABLED(A) PRIMARY      emc_clariion0_1017 emc_clariion0 c3    -
sdbf          ENABLED    SECONDARY    emc_clariion0_1017 emc_clariion0 c3    -
sdau          ENABLED(A) PRIMARY      emc_clariion0_1018 emc_clariion0 c3    -
sdbe          ENABLED    SECONDARY    emc_clariion0_1018 emc_clariion0 c3    -
```

To list all subpaths of a controller on an enclosure:

```
# vxdmpadm getsubpaths enclosure=emc_clariion0
```

By default, the output of the `vxdmpadm getsubpaths` command is sorted by enclosure name, DMP node name, and within that, path name.

To sort the output based on the pathname, the DMP node name, the enclosure name, or the host controller name, use the `-s` option.

To sort subpaths information, use the following command:

```
# vxdmpadm -s {path | dmpnode | enclosure | ctlr} getsubpaths \
[all | ctlr=ctlr_name | dmpnodename=dmp_device_name | \
enclosure=enclr_name [ctlr=ctlr_name | portid=array_port_ID] | \
pwwn=port_WWN | tpdnodename=tpd_node_name]
```

See “Setting customized names for DMP nodes” on page 288.

Displaying information about controllers

The following Dynamic Multi-Pathing (DMP) command lists attributes of all HBA controllers on the system:

```
# vxdmpadm listctlr all

CTLR-NAME  ENCLR-TYPE     STATE    ENCLR-NAME       PATH_COUNT
===============================================================
c1         OTHER          ENABLED  other0           3
c2         X1             ENABLED  jbod0            10
c3         ACME           ENABLED  enc0             24
c4         ACME           ENABLED  enc0             24
```

This output shows that the controller `c1` is connected to disks that are not in any recognized DMP category as the enclosure type is `OTHER`.

The other controllers are connected to disks that are in recognized DMP categories.

All the controllers are in the ENABLED state, which indicates that they are available for I/O operations.

The state DISABLED is used to indicate that controllers are unavailable for I/O operations. The unavailability can be due to a hardware failure or due to I/O operations being disabled on that controller by using the `vxdmpadm disable` command.

The following forms of the command lists controllers belonging to a specified enclosure or enclosure type:

```
# vxdmpadm listctlr enclosure=emc0
```

or

```
# vxdmpadm listctlr type=EMC
```

```
# vxdmpadm listctlr type=EMC
CTLR_NAME   ENCLR_TYPE    STATE      ENCLR_NAME  PATH_COUNT
===========================================================
c1          EMC           ENABLED    emc0             6
c6          EMC           ENABLED    emc0             6
```

The `vxdmpadm getctlr` command displays HBA vendor details and the Controller ID. For iSCSI devices, the Controller ID is the IQN or IEEE-format based name. For FC devices, the Controller ID is the WWN. Because the WWN is obtained from ESD, this field is blank if ESD is not running. ESD is a daemon process used to notify DDL about occurrence of events. The WWN shown as ‘Controller ID’ maps to the WWN of the HBA port associated with the host controller.

```
# vxdmpadm getctlr c5
```

```
LNAME     PNAME VENDOR  CTLR-ID
===================================================
c5        c5    qlogic      20:07:00:a0:b8:17:e1:37
```

Displaying information about enclosures

Dynamic Multi-Pathing (DMP) can display the attributes of the enclosures, including the enclosure type, enclosure serial number, status, array type, number of LUNs, and the firmware version, if available.

To display the attributes of a specified enclosure, use the following DMP command:

```
# vxdmpadm listenclosure emc0
ENCLR_NAME ENCLR_TYPE ENCLR_SNO            STATUS    ARRAY_TYPE LUN_COUNT FIRMWARE
==================================================================================
emc0       EMC        000292601383         CONNECTED A/A        30        5875
```

To display the attrtibutes for all enclosures in a system, use the following DMP command:

```
# vxdmpadm listenclosure all

ENCLR_NAME       ENCLR_TYPE      ENCLR_SNO      STATUS    ARRAY_TYPE LUN_COUNT FIRMWARE
=======================================================================================
Disk             Disk            DISKS          CONNECTED Disk       6         -
emc0             EMC             000292601383   CONNECTED A/A        1         5875
hitachi_usp-vm0  Hitachi_USP-VM  25847          CONNECTED A/A        1         6008
emc_clariion0    EMC_CLARiiON    CK20007040035  CONNECTED CLR-A/PF   2         0324
```

Displaying information about array ports

Use the Dynamic Multi-Pathing (DMP) commands in this section to display information about array ports. The information displayed for an array port includes the name of its enclosure, its ID, and its worldwide name (WWN) identifier.

To display the attributes of an array port that is accessible through a path, DMP node or HBA controller, use one of the following commands:

```
# vxdmpadm getportids path=path_name
# vxdmpadm getportids dmpnodename=dmpnode_name
# vxdmpadm getportids ctlr=ctlr_name
```

The following form of the command displays information about all of the array ports within the specified enclosure:

```
# vxdmpadm getportids enclosure=enclr_name
```

The following example shows information about the array port that is accessible through DMP node `sdg`:

```
# vxdmpadm getportids dmpnodename=sdg

NAME        ENCLR-NAME   ARRAY-PORT-ID pWWN
==============================================================
sdg         HDS9500V0    1A            20:00:00:E0:8B:06:5F:19
```

Displaying information about devices controlled by third-party drivers

The third-party driver (TPD) coexistence feature allows I/O that is controlled by third-party multi-pathing drivers to bypass Dynamic Multi-Pathing (DMP) while retaining the monitoring capabilities of DMP. The following commands allow you to display the paths that DMP has discovered for a given TPD device, and the TPD device that corresponds to a given TPD-controlled node discovered by DMP:

```
# vxdmpadm getsubpaths tpdnodename=TPD_node_name
# vxdmpadm gettpdnode nodename=TPD_path_name
```

See “Changing device naming for enclosures controlled by third-party drivers” on page 290.

For example, consider the following disks in an EMC Symmetrix array controlled by PowerPath, which are known to DMP:

```
# vxdisk list

DEVICE       TYPE            DISK         GROUP        STATUS
emcpowerp    auto:cdsdisk    -            -            online
emcpowerq    auto:cdsdisk    -            -            online
emcpowerr    auto:cdsdisk    -            -            online
emcpowers    auto:cdsdisk    -            -            online
emcpowert    auto:cdsdisk    -            -            online
```

The following command displays the paths that DMP has discovered, and which correspond to the PowerPath-controlled node, `emcpowerp`:

```
# vxdmpadm getsubpaths tpdnodename=emcpowerp

NAME         TPDNODENAME  PATH-TYPE[-] DMPNODENAME  ENCLR-TYPE   ENCLR-NAME
================================================================================
sdt          emcpowerp        -        emcpowerp    PP_EMC_CLARiiON  pp_emc_clariion0
sdo          emcpowerp        -        emcpowerp    PP_EMC_CLARiiON  pp_emc_clariion0
sdj          emcpowerp        -        emcpowerp    PP_EMC_CLARiiON  pp_emc_clariion0
sde          emcpowerp        -        emcpowerp    PP_EMC_CLARiiON  pp_emc_clariion0
```

Conversely, the next command displays information about the PowerPath node that corresponds to the path, `sdt`, discovered by DMP:

```
# vxdmpadm gettpdnode nodename=sdt
NAME                  STATE     PATHS     ENCLR-TYPE   ENCLR-NAME
===========================================================================
emcpowerp            ENABLED      4         PP_EMC_CLARiiON  pp_emc_clariion0
```

Displaying extended device attributes

Device Discovery Layer (DDL) extended attributes are attributes or flags corresponding to a Veritas Volume Manager (VxVM) or Dynamic Multi-Pathing (DMP) LUN or disk and that are discovered by DDL. These attributes identify a LUN to a specific hardware category.

Table 9-5 describes the list of categories.

Table 9-5 Categories for extended attributes

Category	Description
Hardware RAID types	Displays what kind of Storage RAID Group the LUN belongs to
Thin Provisioning Discovery and Reclamation	Displays the LUN’s thin reclamation abilities
Device Media Type	Displays the type of media –whether SSD (Solid State Drive)
Storage-based Snapshot/Clone	Displays whether the LUN is a SNAPSHOT or a CLONE of a PRIMARY LUN
Storage-based replication	Displays if the LUN is part of a replicated group across a remote site
Transport	Displays what kind of HBA is used to connect to this LUN (FC, SATA, iSCSI)

Each LUN can have one or more of these extended attributes. DDL discovers the extended attributes during device discovery from the Array Support Library (ASL). If Veritas Operations Manager (VOM) is present, DDL can also obtain extended attributes from the VOM Management Server for hosts that are configured as managed hosts.

The `vxdisk -p list` command displays DDL extended attributes. For example, the following command shows attributes of `std`, `fc`, and `RAID_5` for this LUN:

```
# vxdisk -p list
DISK            : tagmastore-usp0_0e18
DISKID          : 1253585985.692.rx2600h11
VID             : HITACHI
UDID            : HITACHI%5FOPEN-V%5F02742%5F0E18
REVISION        : 5001
PID             : OPEN-V
PHYS_CTLR_NAME  : 0/4/1/1.0x50060e8005274246
LUN_SNO_ORDER   : 411
LUN_SERIAL_NO   : 0E18
LIBNAME         : libvxhdsusp.sl
HARDWARE_MIRROR: no
DMP_DEVICE      : tagmastore-usp0_0e18
DDL_THIN_DISK   : thick
DDL_DEVICE_ATTR: std fc RAID_5
CAB_SERIAL_NO   : 02742
```

```
ATYPE          : A/A
ARRAY_VOLUME_ID: 0E18
ARRAY_PORT_PWWN: 50:06:0e:80:05:27:42:46
ANAME          : TagmaStore-USP
TRANSPORT      : FC
```

The `vxdisk -x attribute -p list` command displays the one-line listing for the property list and the attributes. The following example shows two Hitachi LUNs that support Thin Reclamation through the attribute `hdprclm`:

```
# vxdisk -x DDL_DEVICE_ATTR -p list
DEVICE                  DDL_DEVICE_ATTR
tagmastore-usp0_0a7a    std fc RAID_5
tagmastore-usp0_065a    hdprclm fc
tagmastore-usp0_065b    hdprclm fc
```

User can specify multiple -x options in the same command to display multiple entries. For example:

```
# vxdisk -x DDL_DEVICE_ATTR -x VID -p list

DEVICE               DDL_DEVICE_ATTR  VID
tagmastore-usp0_0a7a std fc RAID_5    HITACHI
tagmastore-usp0_0a7b std fc RAID_5    HITACHI
tagmastore-usp0_0a78 std fc RAID_5    HITACHI
tagmastore-usp0_0a79 std fc RAID_5    HITACHI
tagmastore-usp0_065a hdprclm fc       HITACHI
tagmastore-usp0_065b hdprclm fc       HITACHI
tagmastore-usp0_065c hdprclm fc       HITACHI
tagmastore-usp0_065d hdprclm fc       HITACHI
```

Use the `vxdisk -e list` command to show the DLL_DEVICE_ATTR property in the last column named ATTR.

```
# vxdisk -e list
DEVICE               TYPE  DISK  GROUP  STATUS  OS_NATIVE_NAME  ATTR
tagmastore-usp0_0a7a auto  -     -      online  c10t0d2         std fc RAID_5
tagmastore-usp0_0a7b auto  -     -      online  c10t0d3         std fc RAID_5
tagmastore-usp0_0a78 auto  -     -      online  c10t0d0         std fc RAID_5
tagmastore-usp0_0655 auto  -     -      online  c13t2d7         hdprclm fc
tagmastore-usp0_0656 auto  -     -      online  c13t3d0         hdprclm fc
tagmastore-usp0_0657 auto  -     -      online  c13t3d1         hdprclm fc
```

For a list of ASLs that supports Extended Attributes, and descriptions of these attributes, refer to the hardware compatibility list (HCL).

Suppressing or including devices from VxVM control

The `vxdmpadm exclude` command suppresses devices from Veritas Volume Manager (VxVM) based on the criteria that you specify. When a device is suppressed, Dynamic Multi-Pathing (DMP) does not claim the device so that the device is not available for VxVM to use. You can add the devices back into VxVM control with the `vxdmpadm include` command. The devices can be included or excluded based on VID:PID combination, paths, controllers, or disks. You can use the bang symbol (!) to exclude or include any paths or controllers except the one specified.

The root disk cannot be suppressed. The operation fails if the VID:PID of an external disk is the same VID:PID as the root disk and the root disk is encapsulated under VxVM.

Note: The ! character is a special character in some shells. The following syntax shows how to escape it in a bash shell.

```
# vxdmpadm exclude { all | product=VID:PID |
ctlr=[\!]ctlrname | dmpnodename=diskname [ path=[\!]pathname] }
```

```
# vxdmpadm include { all | product=VID:PID |
ctlr=[\!]ctlrname | dmpnodename=diskname [ path=[\!]pathname] }
```

where:

all	all devices
product=*VID:PID*	all devices with the specified VID:PID
ctlr=*ctlrname*	all devices through the given controller
dmpnodename=*diskname*	all paths under the DMP node
dmpnodename=*diskname* path=\!*pathname*	all paths under the DMP node except the one specified

Gathering and displaying I/O statistics

You can use the `vxdmpadm iostat` command to gather and display I/O statistics for a specified DMP node, enclosure, path, port, or controller.

The statistics displayed are the CPU usage and amount of memory per CPU used to accumulate statistics, the number of read and write operations, the number of kilobytes read and written, and the average time in milliseconds per kilobyte that is read or written.

To enable the gathering of statistics, enter this command:

```
# vxdmpadm iostat start [memory=size]
```

The `memory` attribute limits the maximum amount of memory that is used to record I/O statistics for each CPU. The default limit is `32k` (32 kilobytes) per CPU.

To reset the I/O counters to zero, use this command:

```
# vxdmpadm iostat reset
```

To display the accumulated statistics at regular intervals, use the following command:

```
# vxdmpadm iostat show {filter} [interval=seconds [count=N]]
```

The above command displays I/O statistics for the devices specified by the *filter*. The *filter* is one of the following:

- `all`
- `ctlr=`*`ctlr-name`*
- `dmpnodename=`*`dmp-node`*
- `enclosure=`*`enclr-name`* `[portid=`*`array-portid`* `] [ctlr=`*`ctlr-name`*`]`
- `pathname=`*`path-name`*
- `pwwn=`*`array-port-wwn`* `[ctlr=`*`ctlr-name`*`]`

Use the `interval` and `count` attributes to specify the interval in seconds between displaying the I/O statistics, and the number of lines to be displayed. The actual interval may be smaller than the value specified if insufficient memory is available to record the statistics.

DMP also provides a *groupby* option to display cumulative I/O statistics, aggregated by the specified criteria.

See “Displaying cumulative I/O statistics” on page 231.

To disable the gathering of statistics, enter this command:

```
# vxdmpadm iostat stop
```

Displaying cumulative I/O statistics

The `vxdmpadm iostat` command provides the ability to analyze the I/O load distribution across various I/O channels or parts of I/O channels. Select the appropriate *filter* to display the I/O statistics for the DMP node, controller, array enclosure, path, port, or virtual machine. Then, use the *groupby* clause to display cumulative statistics according to the criteria that you want to analyze. If the `groupby` clause is not specified, then the statistics are displayed per path.

When you combine the *filter* and the *groupby* clause, you can analyze the I/O load for the required use case scenario. For example:

- To compare I/O load across HBAs, enclosures, or array ports, use the *groupby* clause with the specified attribute.
- To analyze I/O load across a given I/O channel (HBA to array port link), use *filter* by HBA and PWWN or enclosure and array port.
- To analyze I/O load distribution across links to an HBA, use *filter* by HBA and *groupby* array port.

Use the following format of the `iostat` command to analyze the I/O loads:

```
# vxdmpadm [-u unit] iostat show [groupby=criteria] {filter}  \
  [interval=seconds [count=N]]
```

The above command displays I/O statistics for the devices specified by the *filter*. The *filter* is one of the following:

- `all`
- `ctlr=`*`ctlr-name`*
- `dmpnodename=`*`dmp-node`*
- `enclosure=`*`enclr-name`* `[portid=`*`array-portid`* `] [ctlr=`*`ctlr-name`*`]`
- `pathname=`*`path-name`*
- `pwwn=`*`array-port-wwn`*`[ctlr=`*`ctlr-name`*`]`

You can aggregate the statistics by the following `groupby` criteria:

- `arrayport`
- `ctlr`
- `dmpnode`
- `enclosure`

By default, the read/write times are displayed in milliseconds up to 2 decimal places. The throughput data is displayed in terms of BLOCKS, and the output is scaled, meaning that the small values are displayed in small units and the larger values are displayed in bigger units, keeping significant digits constant. You can specify the units in which the statistics data is displayed. The `-u` option accepts the following options:

`h or H`	Displays throughput in the highest possible unit.
`k`	Displays throughput in kilobytes.
`m`	Displays throughput in megabytes.
`g`	Displays throughput in gigabytes.
`bytes| b`	Displays throughput in exact number of bytes.
`us`	Displays average read/write time in microseconds.

To group by DMP node:

```
# vxdmpadm [-u unit] iostat show groupby=dmpnode \
[all | dmpnodename=dmpnodename | enclosure=enclr-name]
```

To group by controller:

```
# vxdmpadm [-u unit] iostat show groupby=ctlr [ all | ctlr=ctlr ]
```

For example:

```
# vxdmpadm iostat show groupby=ctlr ctlr=c5

            OPERATIONS        BLOCKS       AVG TIME(ms)
CTLRNAME   READS   WRITES  READS  WRITES  READS   WRITES
c5         224      14     54      7       4.20    11.10
```

To group by arrayport:

```
# vxdmpadm [-u unit] iostat show groupby=arrayport [ all \
| pwwn=array_pwwn | enclosure=enclr portid=array-port-id ]
```

For example:

```
# vxdmpadm -u m iostat show groupby=arrayport \
enclosure=HDS9500-ALUA0 portid=1A

            OPERATIONS        BYTES       AVG TIME(ms)
PORTNAME   READS   WRITES  READS  WRITES  READS   WRITES
1A         743     1538    11m    24m     17.13   8.61
```

To group by enclosure:

```
# vxdmpadm [-u unit] iostat show groupby=enclosure [ all \
| enclosure=enclr ]
```

For example:

```
# vxdmpadm -u h iostat show groupby=enclosure enclosure=EMC_CLARiiON0

OPERATIONS        BLOCKS        AVG TIME(ms)
ENCLOSURENAME   READS  WRITES  READS  WRITES  READS   WRITES
EMC_CLARiiON0   743    1538    11392k  24176k 17.13   8.61
```

You can also filter out entities for which all data entries are zero. This option is especially useful in a cluster environment that contains many failover devices. You can display only the statistics for the active paths.

To filter all zero entries from the output of the `iostat show` command:

```
# vxdmpadm [-u unit] -z iostat show [all|ctlr=ctlr_name |
dmpnodename=dmp_device_name | enclosure=enclr_name [portid=portid] |
pathname=path_name|pwwn=port_WWN][interval=seconds [count=N]]
```

For example:

```
# vxdmpadm -z iostat show dmpnodename=emc_clariion0_893

cpu usage = 9852us    per cpu memory = 266240b
             OPERATIONS       BLOCKS           AVG TIME(ms)
PATHNAME   READS  WRITES    READS   WRITES    READS    WRITES
sdbc       32     0         258     0          0.04      0.00
sdbw       27     0         216     0          0.03      0.00
sdck       8      0         57      0          0.04      0.00
sdde       11     0         81      0          0.15      0.00
```

To display average read/write times in microseconds.

```
# vxdmpadm -u us iostat show pathname=sdck

cpu usage = 9865us    per cpu memory = 266240b
              OPERATIONS        BLOCKS           AVG TIME(us)
PATHNAME    READS    WRITES  READS    WRITES   READS    WRITES
sdck        8        0       57       0        43.04      0.00
```

Displaying statistics for queued or erroneous I/Os

Use the `vxdmpadm iostat show` command with the `-q` option to display the I/Os queued in Dynamic Multi-Pathing (DMP) for a specified DMP node, or for a specified path or controller. For a DMP node, the `-q` option displays the I/Os on the specified DMP node that were sent to underlying layers. If a path or controller is specified, the `-q` option displays I/Os that were sent to the given path or controller and not yet returned to DMP.

See the `vxdmpadm`(1m) manual page for more information about the `vxdmpadm iostat` command.

To display queued I/O counts on a DMP node:

```
# vxdmpadm -q iostat show [filter] [interval=n [count=m]]
```

For example:

```
# vxdmpadm -q iostat show dmpnodename=emc_clariion0_352

cpu usage = 338us    per cpu memory = 102400b
                        QUEUED I/Os      PENDING I/Os
DMPNODENAME          READS    WRITES
emc_clariion0_352    0        0          0
```

To display the count of I/Os that returned with errors on a DMP node, path, or controller:

```
# vxdmpadm -e iostat show [filter] [interval=n [count=m]]
```

For example, to show the I/O counts that returned errors on a path:

```
# vxdmpadm -e iostat show pathname=sdo

cpu usage = 637us    per cpu memory = 102400b
                  ERROR I/Os
PATHNAME      READS    WRITES
sdo           0         0
```

Examples of using the vxdmpadm iostat command

Dynamic Multi-Pathing (DMP) enables you to gather and display I/O statistics with the `vxdmpadm iostat` command. This section provides an example session using the `vxdmpadm iostat` command.

The first command enables the gathering of I/O statistics:

```
# vxdmpadm iostat start
```

The next command displays the current statistics including the accumulated total numbers of read and write operations, and the kilobytes read and written, on all paths.

```
# vxdmpadm -u k iostat show all
                     cpu usage = 7952us    per cpu memory = 8192b
           OPERATIONS           BYTES          AVG TIME(ms)
PATHNAME READS     WRITES    READS     WRITES   READS   WRITES
```

```
sdf         87          0    44544k         0    0.00     0.00
sdk          0          0        0          0    0.00     0.00
sdg         87          0    44544k         0    0.00     0.00
sdl          0          0        0          0    0.00     0.00
sdh         87          0    44544k         0    0.00     0.00
sdm          0          0        0          0    0.00     0.00
sdi         87          0    44544k         0    0.00     0.00
sdn          0          0        0          0    0.00     0.00
sdj         87          0    44544k         0    0.00     0.00
sdo          0          0        0          0    0.00     0.00
sdj         87          0    44544k         0    0.00     0.00
sdp          0          0        0          0    0.00     0.00
```

The following command changes the amount of memory that `vxdmpadm` can use to accumulate the statistics:

```
# vxdmpadm iostat start memory=4096
```

The displayed statistics can be filtered by path name, DMP node name, and enclosure name (note that the per-CPU memory has changed following the previous command):

```
# vxdmpadm -u k iostat show pathname=sdk
                     cpu usage = 8132us    per cpu memory = 4096b
               OPERATIONS             BYTES            AVG TIME(ms)
PATHNAME    READS     WRITES      READS     WRITES     READS   WRITES
sdk             0          0          0          0      0.00     0.00
```

```
# vxdmpadm -u k iostat show dmpnodename=sdf
                     cpu usage = 8501us    per cpu memory = 4096b
               OPERATIONS             BYTES            AVG TIME(ms)
PATHNAME    READS     WRITES      READS     WRITES     READS   WRITES
sdf          1088          0    557056k          0      0.00     0.00
```

```
# vxdmpadm -u k iostat show enclosure=Disk
                    cpu usage = 8626us    per cpu memory = 4096b
               OPERATIONS             BYTES            AVG TIME(ms)
PATHNAME    READS     WRITES      READS     WRITES     READS   WRITES
sdf          1088          0    557056k          0      0.00     0.00
```

You can also specify the number of times to display the statistics and the time interval. Here the incremental statistics for a path are displayed twice with a 2-second interval:

```
# vxdmpadm iostat show pathname=sdk interval=2 count=2
                   cpu usage = 9621us    per cpu memory = 266240b
              OPERATIONS            BLOCKS           AVG TIME(ms)
PATHNAME     READS    WRITES     READS    WRITES     READS    WRITES
sdk          0         0         0         0          0.00      0.00

sdk          0         0         0         0          0.00      0.00
```

Setting the attributes of the paths to an enclosure

You can use the `vxdmpadm setattr` command to set the attributes of the paths to an enclosure or disk array.

The attributes set for the paths are persistent across reboots or product upgrades.

You can set the following attributes:

`active` Changes a standby (failover) path to an active path. The following example specifies an active path for an array:

```
# vxdmpadm setattr path sde pathtype=active
```

`nomanual` Restores the original primary or secondary attributes of a path. This example restores the path to a JBOD disk:

```
# vxdmpadm setattr path sdm pathtype=nomanual
```

`nopreferred` Restores the normal priority of a path. The following example restores the default priority to a path:

```
# vxdmpadm setattr path sdk \
  pathtype=nopreferred
```

`preferred [priority=`*`N`*`]`

Specifies a path as preferred, and optionally assigns a priority number to it. If specified, the priority number must be an integer that is greater than or equal to one. Higher priority numbers indicate that a path is able to carry a greater I/O load.

See "Specifying the I/O policy" on page 240.

This example first sets the I/O policy to `priority` for an Active/Active disk array, and then specifies a preferred path with an assigned priority of 2:

```
# vxdmpadm setattr enclosure enc0 \
  iopolicy=priority
# vxdmpadm setattr path sdk pathtype=preferred \
  priority=2
```

`primary`

Defines a path as being the primary path for a JBOD disk array. The following example specifies a primary path for a JBOD disk array:

```
# vxdmpadm setattr path sdm pathtype=primary
```

`secondary`

Defines a path as being the secondary path for a JBOD disk array. The following example specifies a secondary path for a JBOD disk array:

```
# vxdmpadm setattr path sdn pathtype=secondary
```

`standby`

Marks a standby (failover) path that it is not used for normal I/O scheduling. This path is used if there are no active paths available for I/O. The next example specifies a standby path for an A/P-C disk array:

```
# vxdmpadm setattr path sde pathtype=standby
```

Displaying the redundancy level of a device or enclosure

Use the `vxdmpadm getdmpnode` command to list the devices with less than the required redundancy level.

To list the devices on a specified enclosure with fewer than a given number of enabled paths, use the following command:

```
# vxdmpadm getdmpnode enclosure=encl_name redundancy=value
```

For example, to list the devices with fewer than 3 enabled paths, use the following command:

```
# vxdmpadm getdmpnode enclosure=EMC_CLARiiON0 redundancy=3
```

```
NAME               STATE    ENCLR-TYPE   PATHS ENBL DSBL ENCLR-NAME
=====================================================================
emc_clariion0_162 ENABLED  EMC_CLARiiON 3     2    1    emc_clariion0
emc_clariion0_182 ENABLED  EMC_CLARiiON 2     2    0    emc_clariion0
emc_clariion0_184 ENABLED  EMC_CLARiiON 3     2    1    emc_clariion0
emc_clariion0_186 ENABLED  EMC_CLARiiON 2     2    0    emc_clariion0
```

To display the minimum redundancy level for a particular device, use the `vxdmpadm getattr` command, as follows:

```
# vxdmpadm getattr enclosure|arrayname|arraytype \
 component-name redundancy
```

For example, to show the minimum redundancy level for the enclosure HDS9500-ALUA0:

```
# vxdmpadm getattr enclosure HDS9500-ALUA0 redundancy
```

```
ENCLR_NAME  DEFAULT  CURRENT
=============================================
 HDS9500-ALUA0    0      4
```

Specifying the minimum number of active paths

You can set the minimum redundancy level for a device or an enclosure. The minimum redundancy level is the minimum number of paths that should be active for the device or the enclosure. If the number of paths falls below the minimum redundancy level for the enclosure, a message is sent to the system console and also logged to the Dynamic Multi-Pathing (DMP) log file. Also, notification is sent to `vxnotify` clients.

The value set for minimum redundancy level is persistent across reboots and product upgrades. If no minimum redundancy level is set, the default value is 0.

You can use the `vxdmpadm setattr` command to set the minimum redundancy level.

To specify the minimum number of active paths

- Use the `vxdmpadm setattr` command with the redundancy attribute as follows:

  ```
  # vxdmpadm setattr enclosure|arrayname|arraytype component-name
   redundancy=value
  ```

  where *value* is the number of active paths.

  For example, to set the minimum redundancy level for the enclosure HDS9500-ALUA0:

  ```
  # vxdmpadm setattr enclosure HDS9500-ALUA0 redundancy=2
  ```

Displaying the I/O policy

To display the current and default settings of the I/O policy for an enclosure, array, or array type, use the `vxdmpadm getattr` command.

The following example displays the default and current setting of `iopolicy` for JBOD disks:

```
# vxdmpadm getattr enclosure Disk iopolicy

ENCLR_NAME        DEFAULT         CURRENT
----------------------------------------
Disk              MinimumQ        Balanced
```

The next example displays the setting of `partitionsize` for the enclosure `enc0`, on which the `balanced` I/O policy with a partition size of 2MB has been set:

```
# vxdmpadm getattr enclosure enc0 partitionsize

ENCLR_NAME      DEFAULT         CURRENT
----------------------------------------
enc0            512             4096
```

Specifying the I/O policy

You can use the `vxdmpadm setattr` command to change the Dynamic Multi-Pathing (DMP) I/O policy for distributing I/O load across multiple paths to a disk array or enclosure. You can set policies for an enclosure (for example, `HDS01`), for all enclosures of a particular type (such as `HDS`), or for all enclosures of a particular array type (such as `A/A` for Active/Active, or `A/P` for Active/Passive).

Note: I/O policies are persistent across reboots of the system.

Table 9-6 describes the I/O policies that may be set.

Table 9-6 DMP I/O policies

Policy	Description

`adaptive`	This policy attempts to maximize overall I/O throughput from/to the disks by dynamically scheduling I/O on the paths. It is suggested for use where I/O loads can vary over time. For example, I/O from/to a database may exhibit both long transfers (table scans) and short transfers (random look ups). The policy is also useful for a SAN environment where different paths may have different number of hops. No further configuration is possible as this policy is automatically managed by DMP.
	In this example, the adaptive I/O policy is set for the enclosure `enc1`:
	`# vxdmpadm setattr enclosure enc1 \`
	`  iopolicy=adaptive`

`adaptiveminq`	Similar to the `adaptive` policy, except that I/O is scheduled according to the length of the I/O queue on each path. The path with the shortest queue is assigned the highest priority.

Table 9-6 DMP I/O policies *(continued)*

Policy	Description
`balanced [partitionsize=`*`size`*`]`	This policy is designed to optimize the use of caching in disk drives and RAID controllers. The size of the cache typically ranges from 120KB to 500KB or more, depending on the characteristics of the particular hardware. During normal operation, the disks (or LUNs) are logically divided into a number of regions (or partitions), and I/O from/to a given region is sent on only one of the active paths. Should that path fail, the workload is automatically redistributed across the remaining paths. You can use the partitionsize attribute to specify the size for the partition. The partition size in blocks is adjustable in powers of 2 from 2 up to 231. A value that is not a power of 2 is silently rounded down to the nearest acceptable value. Specifying a partition size of 0 is equivalent to specifying the default partition size. The default value for the partition size is 512 blocks (256k). Specifying a partition size of 0 is equivalent to the default partition size of 512 blocks (256k). The default value can be changed by adjusting the value of the `dmp_pathswitch_blks_shift` tunable parameter. See "DMP tunable parameters" on page 796. **Note:** The benefit of this policy is lost if the value is set larger than the cache size. For example, the suggested partition size for an Hitachi HDS 9960 A/A array is from 32,768 to 131,072 blocks (16MB to 64MB) for an I/O activity pattern that consists mostly of sequential reads or writes. The next example sets the balanced I/O policy with a partition size of 4096 blocks (2MB) on the enclosure enc0: `# `**`vxdmpadm setattr enclosure enc0 \`** **`iopolicy=balanced partitionsize=4096`**
`minimumq`	This policy sends I/O on paths that have the minimum number of outstanding I/O requests in the queue for a LUN. No further configuration is possible as DMP automatically determines the path with the shortest queue. The following example sets the I/O policy to `minimumq` for a JBOD: `# `**`vxdmpadm setattr enclosure Disk \`** **`iopolicy=minimumq`** This is the default I/O policy for all arrays.

Table 9-6 DMP I/O policies *(continued)*

Policy	Description

priority	This policy is useful when the paths in a SAN have unequal performance, and you want to enforce load balancing manually. You can assign priorities to each path based on your knowledge of the configuration and performance characteristics of the available paths, and of other aspects of your system.
	See "Setting the attributes of the paths to an enclosure" on page 237.
	In this example, the I/O policy is set to priority for all SENA arrays:
	`# vxdmpadm setattr arrayname SENA \`
	`iopolicy=priority`

round-robin	This policy shares I/O equally between the paths in a round-robin sequence. For example, if there are three paths, the first I/O request would use one path, the second would use a different path, the third would be sent down the remaining path, the fourth would go down the first path, and so on. No further configuration is possible as this policy is automatically managed by DMP.
	The next example sets the I/O policy to round-robin for all Active/Active arrays:
	`# vxdmpadm setattr arraytype A/A \`
	`iopolicy=round-robin`

singleactive	This policy routes I/O down the single active path. This policy can be configured for A/P arrays with one active path per controller, where the other paths are used in case of failover. If configured for A/A arrays, there is no load balancing across the paths, and the alternate paths are only used to provide high availability (HA). If the current active path fails, I/O is switched to an alternate active path. No further configuration is possible as the single active path is selected by DMP.
	The following example sets the I/O policy to singleactive for JBOD disks:
	`# vxdmpadm setattr arrayname Disk \`
	`iopolicy=singleactive`

Scheduling I/O on the paths of an Asymmetric Active/Active or an ALUA array

You can specify the `use_all_paths` attribute in conjunction with the `adaptive`, `balanced`, `minimumq`, `priority`, and `round-robin` I/O policies to specify whether I/O requests are to be scheduled on the secondary paths in addition to the primary paths of an Asymmetric Active/Active (A/A-A) array or an ALUA array. Depending on the characteristics of the array, the consequent improved load balancing can increase the total I/O throughput. However, this feature should only be enabled if recommended by the array vendor. It has no effect for array types other than A/A-A or ALUA.

For example, the following command sets the `balanced` I/O policy with a partition size of 4096 blocks (2MB) on the enclosure `enc0`, and allows scheduling of I/O requests on the secondary paths:

```
# vxdmpadm setattr enclosure enc0 iopolicy=balanced \
      partitionsize=4096 use_all_paths=yes
```

The default setting for this attribute is `use_all_paths=no`.

You can display the current setting for `use_all_paths` for an enclosure, arrayname, or arraytype. To do this, specify the `use_all_paths` option to the `vxdmpadm gettattr` command.

```
# vxdmpadm getattr enclosure HDS9500-ALUA0 use_all_paths

ENCLR_NAME    ATTR_NAME     DEFAULT CURRENT
===========================================
HDS9500-ALUA0 use_all_paths no      yes
```

The `use_all_paths` attribute only applies to A/A-A arrays and ALUA arrays. For other arrays, the above command displays the message:

```
Attribute is not applicable for this array.
```

Example of applying load balancing in a SAN

This example describes how to use Dynamic Multi-Pathing (DMP) to configure load balancing in a SAN environment where there are multiple primary paths to an Active/Passive device through several SAN switches.

As shown in this sample output from the `vxdisk list` command, the device `sdm` has eight primary paths:

```
# vxdisk list sdq

Device: sdq
    .
    .
    .
numpaths: 8
sdj state=enabled type=primary
sdk state=enabled type=primary
sdl state=enabled type=primary
```

```
sdm state=enabled type=primary
sdn state=enabled type=primary
sdo state=enabled type=primary
sdp state=enabled type=primary
sdq state=enabled type=primary
```

In addition, the device is in the enclosure `ENC0`, belongs to the disk group `mydg`, and contains a simple concatenated volume `myvol1`.

The first step is to enable the gathering of DMP statistics:

```
# vxdmpadm iostat start
```

Next, use the `dd` command to apply an input workload from the volume:

```
# dd if=/dev/vx/rdsk/mydg/myvol1 of=/dev/null &
```

By running the `vxdmpadm iostat` command to display the DMP statistics for the device, it can be seen that all I/O is being directed to one path, `sdq`:

```
# vxdmpadm iostat show dmpnodename=sdq interval=5 count=2
    .
    .
    .
cpu usage = 11294us per cpu memory = 32768b
             OPERATIONS          KBYTES          AVG TIME(ms)
PATHNAME   READS    WRITES   READS    WRITES   READS      WRITES
sdj        0        0        0        0         0.00        0.00
sdk        0        0        0        0         0.00        0.00
sdl        0        0        0        0         0.00        0.00
sdm        0        0        0        0         0.00        0.00
sdn        0        0        0        0         0.00        0.00
sdo        0        0        0        0         0.00        0.00
sdp        0        0        0        0         0.00        0.00
sdq        10986    0        5493     0         0.41        0.00
```

The `vxdmpadm` command is used to display the I/O policy for the enclosure that contains the device:

```
# vxdmpadm getattr enclosure ENC0 iopolicy

ENCLR_NAME     DEFAULT          CURRENT
============================================
ENC0           MinimumQ      Single-Active
```

This shows that the policy for the enclosure is set to `singleactive`, which explains why all the I/O is taking place on one path.

To balance the I/O load across the multiple primary paths, the policy is set to `round-robin` as shown here:

```
# vxdmpadm setattr enclosure ENC0 iopolicy=round-robin
# vxdmpadm getattr enclosure ENC0 iopolicy

ENCLR_NAME     DEFAULT            CURRENT
============================================
ENC0           MinimumQ           Round-Robin
```

The DMP statistics are now reset:

```
# vxdmpadm iostat reset
```

With the workload still running, the effect of changing the I/O policy to balance the load across the primary paths can now be seen.

```
# vxdmpadm iostat show dmpnodename=sdq interval=5 count=2
    .
    .
    .
cpu usage = 14403us per cpu memory = 32768b
             OPERATIONS            KBYTES          AVG TIME(ms)
PATHNAME   READS   WRITES    READS      WRITES   READS       WRITES
sdj        2041    0         1021       0         0.39         0.00
sdk        1894    0         947        0         0.39         0.00
sdl        2008    0         1004       0         0.39         0.00
sdm        2054    0         1027       0         0.40         0.00
sdn        2171    0         1086       0         0.39         0.00
sdo        2095    0         1048       0         0.39         0.00
sdp        2073    0         1036       0         0.39         0.00
sdq        2042    0         1021       0         0.39         0.00
```

The enclosure can be returned to the single active I/O policy by entering the following command:

```
# vxdmpadm setattr enclosure ENC0 iopolicy=singleactive
```

Disabling I/O for paths, controllers, array ports, or DMP nodes

Disabling I/O through a path, HBA controller, array port, or Dynamic Multi-Pathing (DMP) node prevents DMP from issuing I/O requests through the specified path, or the paths that are connected to the specified controller, array port, or DMP node.

If the specified paths have pending I/Os, the `vxdmpadm disable` command waits until the I/Os are completed before disabling the paths.

Note: From release 5.0 of Veritas Volume Manager (VxVM), this operation is supported for controllers that are used to access disk arrays on which cluster-shareable disk groups are configured.

DMP does not support the operation to disable I/O for the controllers that use Third-Party Drivers (TPD) for multi-pathing.

To disable I/O for one or more paths, use the following command:

```
# vxdmpadm [-c|-f] disable path=path_name1[,path_name2,path_nameN]
```

To disable I/O for the paths connected to one or more HBA controllers, use the following command:

```
# vxdmpadm [-c|-f] disable ctlr=ctlr_name1[,ctlr_name2,ctlr_nameN]
```

To disable I/O for the paths connected to an array port, use one of the following commands:

```
# vxdmpadm [-c|-f] disable enclosure=enclr_name portid=array_port_ID
# vxdmpadm [-c|-f] disable pwwn=array_port_WWN
```

where the array port is specified either by the enclosure name and the array port ID, or by the array port’s worldwide name (WWN) identifier.

The following examples show how to disable I/O on an array port:

```
# vxdmpadm disable enclosure=HDS9500V0 portid=1A
# vxdmpadm disable pwwn=20:00:00:E0:8B:06:5F:19
```

To disable I/O for a particular path, specify both the controller and the portID, which represent the two ends of the fabric:

```
# vxdmpadm [-c|-f] disable ctlr=ctlr_name enclosure=enclr_name \
portid=array_port_ID
```

To disable I/O for a particular DMP node, specify the DMP node name.

```
# vxdmpadm [-c|-f] disable dmpnodename=dmpnode
```

You can use the `-c` option to check if there is only a single active path to the disk.

Use the **-f** option to disable the last path, irrespective of whether the device is in use or not.

The `disable` operation fails if it is issued to a controller that is connected to the root disk through a single path, and there are no root disk mirrors configured on alternate paths. If such mirrors exist, the command succeeds. The `disable` operation fails if it is issued to a controller that is connected to the swap device through a single path.

Enabling I/O for paths, controllers, array ports, or DMP nodes

Enabling a controller allows a previously disabled path, HBA controller, array port, or Dynamic Multi-Pathing (DMP) node to accept I/O again. This operation succeeds only if the path, controller, array port, or DMP node is accessible to the host, and I/O can be performed on it. When connecting Active/Passive disk arrays, the `enable` operation results in failback of I/O to the primary path. The `enable` operation can also be used to allow I/O to the controllers on a system board that was previously detached.

Note: This operation is supported for controllers that are used to access disk arrays on which cluster-shareable disk groups are configured.

DMP does not support the operation to enable I/O for the controllers that use Third-Party Drivers (TPD) for multi-pathing.

To enable I/O for one or more paths, use the following command:

```
# vxdmpadm enable path=path_name1[,path_name2,path_nameN]
```

To enable I/O for the paths connected to one or more HBA controllers, use the following command:

```
# vxdmpadm enable ctlr=ctlr_name1[,ctlr_name2,ctlr_nameN]
```

To enable I/O for the paths connected to an array port, use one of the following commands:

```
# vxdmpadm enable enclosure=enclr_name portid=array_port_ID
# vxdmpadm enable pwwn=array_port_WWN
```

where the array port is specified either by the enclosure name and the array port ID, or by the array port’s worldwide name (WWN) identifier.

The following are examples of using the command to enable I/O on an array port:

```
# vxdmpadm enable enclosure=HDS9500V0 portid=1A
# vxdmpadm enable pwwn=20:00:00:E0:8B:06:5F:19
```

To enable I/O for a particular path, specify both the controller and the portID, which represent the two ends of the fabric:

```
# vxdmpadm enable ctlr=ctlr_name enclosure=enclr_name \
  portid=array_port_ID
```

To enable I/O for a particular DMP node, specify the DMP node name.

```
# vxdmpadm enable dmpnodename=dmpnode
```

Renaming an enclosure

The `vxdmpadm setattr` command can be used to assign a meaningful name to an existing enclosure, for example:

```
# vxdmpadm setattr enclosure emc0 name=GRP1
```

This example changes the name of an enclosure from `emc0` to `GRP1`.

Note: The maximum length of the enclosure name prefix is 23 characters.

The following command shows the changed name:

```
# vxdmpadm listenclosure all
```

```
ENCLR_NAME      ENCLR_TYPE     ENCLR_SNO     STATUS    ARRAY_TYPE LUN_COUNT F
=============================================================================
Disk            Disk           DISKS         CONNECTED Disk       6         -
GRP1            EMC            000292601383  CONNECTED A/A        1         5
hitachi_usp-vm0 Hitachi_USP-VM 25847         CONNECTED A/A        1         6
emc_clariion0   EMC_CLARiiON   CK20007040035 CONNECTED CLR-A/PF   2         0
```

Configuring the response to I/O failures

You can configure how Dynamic Multi-Pathing (DMP) responds to failed I/O requests on the paths to a specified enclosure, disk array name, or type of array. By default, DMP is configured to retry a failed I/O request up to five minutes on various active paths.

To display the current settings for handling I/O request failures that are applied to the paths to an enclosure, array name, or array type, use the `vxdmpadm getattr` command.

See “Displaying recovery option values” on page 253.

To set a limit for the number of times that DMP attempts to retry sending an I/O request on a path, use the following command:

```
# vxdmpadm setattr \
  {enclosure enc-name|arrayname name|arraytype type} \
  recoveryoption=fixedretry retrycount=n
```

The value of the argument to `retrycount` specifies the number of retries to be attempted before DMP reschedules the I/O request on another available path, or fails the request altogether.

As an alternative to specifying a fixed number of retries, you can specify the amount of time DMP allows for handling an I/O request. If the I/O request does not succeed within that time, DMP fails the I/O request. To specify an `iotimeout` value, use the following command:

```
# vxdmpadm setattr \
  {enclosure enc-name|arrayname name|arraytype type} \
  recoveryoption=timebound iotimeout=seconds
```

The default value of `iotimeout` is 300 seconds. For some applications such as Oracle, it may be desirable to set `iotimeout` to a larger value. The `iotimeout` value for DMP should be greater than the I/O service time of the underlying operating system layers.

Note: The `fixedretry` and `timebound` settings are mutually exclusive.

The following example configures time-bound recovery for the enclosure `enc0`, and sets the value of `iotimeout` to 360 seconds:

```
# vxdmpadm setattr enclosure enc0 recoveryoption=timebound \
  iotimeout=360
```

The next example sets a fixed-retry limit of 10 for the paths to all Active/Active arrays:

```
# vxdmpadm setattr arraytype A/A recoveryoption=fixedretry \
  retrycount=10
```

Specifying `recoveryoption=default` resets DMP to the default settings for recovery.

For example, the following command sets the default settings:

```
# vxdmpadm setattr arraytype A/A recoveryoption=default
```

For PCI devices, the default settings are `recoveryoption=fixedretry retrycount=5`.

For all other devices, the default settings are `recoveryoption=timebound iotimeout=300`

Specifying `recoveryoption=default`also has the effect of configuring I/O throttling with the default settings.

See "Configuring the I/O throttling mechanism" on page 251.

Note: The response to I/O failure settings is persistent across reboots of the system.

Configuring the I/O throttling mechanism

By default, Dynamic Multi-Pathing (DMP) is configured with I/O throttling turned off for all paths. To display the current settings for I/O throttling that are applied to the paths to an enclosure, array name, or array type, use the `vxdmpadm getattr` command.

See "Displaying recovery option values" on page 253.

If enabled, I/O throttling imposes a small overhead on CPU and memory usage because of the activity of the statistics-gathering daemon. If I/O throttling is disabled, the daemon no longer collects statistics, and remains inactive until I/O throttling is re-enabled.

To turn off I/O throttling, use the following form of the `vxdmpadm setattr` command:

```
# vxdmpadm setattr \
  {enclosure enc-name|arrayname name|arraytype type} \
  recoveryoption=nothrottle
```

The following example shows how to disable I/O throttling for the paths to the enclosure `enc0`:

```
# vxdmpadm setattr enclosure enc0 recoveryoption=nothrottle
```

The `vxdmpadm setattr` command can be used to enable I/O throttling on the paths to a specified enclosure, disk array name, or type of array:

```
# vxdmpadm setattr \
  {enclosure enc-name|arrayname name|arraytype type}\
  recoveryoption=throttle [iotimeout=seconds]
```

If the `iotimeout` attribute is specified, its argument specifies the time in seconds that DMP waits for an outstanding I/O request to succeed before invoking I/O throttling on the path. The default value of iotimeout is 10 seconds. Setting iotimeout to a larger value potentially causes more I/O requests to become queued up in the SCSI driver before I/O throttling is invoked.

The following example sets the value of iotimeout to 60 seconds for the enclosure enc0:

```
# vxdmpadm setattr enclosure enc0 recoveryoption=throttle \
  iotimeout=60
```

Specify recoveryoption=default to reset I/O throttling to the default settings, as follows:

```
# vxdmpadm setattr arraytype A/A recoveryoption=default
```

The above command configures the default behavior, corresponding to recoveryoption=nothrottle. The above command also configures the default behavior for the response to I/O failures.

See "Configuring the response to I/O failures" on page 249.

Note: The I/O throttling settings are persistent across reboots of the system.

Configuring Low Impact Path Probing (LIPP)

The Low Impact Path Probing (LIPP) feature can be turned on or off using the vxdmpadm settune command:

```
# vxdmpadm settune dmp_low_impact_probe=[on|off]
```

Path probing will be optimized by probing a subset of paths connected to the same HBA and array port. The size of the subset of paths can be controlled by the dmp_probe_threshold tunable. The default value is set to 5.

```
# vxdmpadm settune dmp_probe_threshold=N
```

Configuring Subpaths Failover Groups (SFG)

The Subpaths Failover Groups (SFG) feature can be turned on or off using the tunable dmp_sfg_threshold. The default value of the tunable is 1, which represents that the feature is on.

To turn off the feature, set the tunable dmp_sfg_threshold value to 0:

```
# vxdmpadm settune dmp_sfg_threshold=0
```

To turn on the feature, set the `dmp_sfg_threshold` value to the required number of path failures that triggers SFG.

```
# vxdmpadm settune dmp_sfg_threshold=N
```

To see the Subpaths Failover Groups ID, use the following command:

```
# vxdmpadm getportids {ctlr=ctlr_name | dmpnodename=dmp_device_name \
    | enclosure=enclr_name | path=path_name}
```

Displaying recovery option values

To display the current settings for handling I/O request failures that are applied to the paths to an enclosure, array name, or array type, use the following Dynamic Multi-Pathing (DMP) command:

```
# vxdmpadm getattr \
  {enclosure enc-name|arrayname name|arraytype type} \
  recoveryoption
```

The following example shows the `vxdmpadm getattr` command being used to display the `recoveryoption` option values that are set on an enclosure.

```
# vxdmpadm getattr enclosure HDS9500-ALUA0 recoveryoption
ENCLR-NAME     RECOVERY-OPTION  DEFAULT[VAL]    CURRENT[VAL]
===============================================================
HDS9500-ALUA0  Throttle         Nothrottle[0]   Nothrottle[0]
HDS9500-ALUA0  Error-Retry      Timebound[300]  Timebound[300]
```

The command output shows the default and current policy options and their values.

Table 9-7 summarizes the possible recovery option settings for retrying I/O after an error.

Table 9-7 Recovery options for retrying I/O after an error

Recovery option	Possible settings	Description
recoveryoption=fixedretry	Fixed-Retry (retrycount)	DMP retries a failed I/O request for the specified number of times if I/O fails.
recoveryoption=timebound	Timebound (iotimeout)	DMP retries a failed I/O request for the specified time in seconds if I/O fails.

Table 9-8 summarizes the possible recovery option settings for throttling I/O.

Table 9-8 Recovery options for I/O throttling

Recovery option	Possible settings	Description
recoveryoption=nothrottle	None	I/O throttling is not used.
recoveryoption=throttle	Timebound (iotimeout)	DMP throttles the path if an I/O request does not return within the specified time in seconds.

Configuring DMP path restoration policies

Dynamic Multi-Pathing (DMP) maintains a kernel task that re-examines the condition of paths at a specified interval. The type of analysis that is performed on the paths depends on the checking policy that is configured.

Note: The DMP path restoration task does not change the disabled state of the path through a controller that you have disabled using `vxdmpadm disable`.

When configuring DMP path restoration policies, you must stop the path restoration thread, and then restart it with new attributes.

See "Stopping the DMP path restoration thread" on page 255.

Use the `vxdmpadm settune dmp_restore_policy` command to configure one of the following restore policies. The policy remains in effect until the restore thread is stopped or the values are changed using the `vxdmpadm settune` command.

- `check_all`

  The path restoration thread analyzes all paths in the system and revives the paths that are back online, as well as disabling the paths that are inaccessible. The command to configure this policy is:

  ```
  # vxdmpadm settune dmp_restore_policy=check_all
  ```

- `check_alternate`

  The path restoration thread checks that at least one alternate path is healthy. It generates a notification if this condition is not met. This policy avoids inquiry commands on all healthy paths, and is less costly than `check_all` in cases where a large number of paths are available. This policy is the same as `check_all` if there are only two paths per DMP node. The command to configure this policy is:

  ```
  # vxdmpadm settune dmp_restore_policy=check_alternate
  ```

- check_disabled

  This is the default path restoration policy. The path restoration thread checks the condition of paths that were previously disabled due to hardware failures, and revives them if they are back online. The command to configure this policy is:

  ```
  # vxdmpadm settune dmp_restore_policy=check_disabled
  ```

- check_periodic

  The path restoration thread performs check_all once in a given number of cycles, and check_disabled in the remainder of the cycles. This policy may lead to periodic slowing down (due to check_all) if a large number of paths are available. The command to configure this policy is:

  ```
  # vxdmpadm settune dmp_restore_policy=check_periodic
  ```

  The default number of cycles between running the check_all policy is 10.

The dmp_restore_interval tunable parameter specifies how often the path restoration thread examines the paths. For example, the following command sets the polling interval to 400 seconds:

```
# vxdmpadm settune dmp_restore_interval=400
```

The settings are immediately applied and are persistent across reboots. Use the vxdmpadm gettune command to view the current settings.

See "DMP tunable parameters" on page 796.

If the vxdmpadm start restore command is given without specifying a policy or interval, the path restoration thread is started with the persistent policy and interval settings previously set by the administrator with the vxdmpadm settune command. If the administrator has not set a policy or interval, the system defaults are used. The system default restore policy is check_disabled. The system default interval is 300 seconds.

Warning: Decreasing the interval below the system default can adversely affect system performance.

Stopping the DMP path restoration thread

Use the following command to stop the Dynamic Multi-Pathing (DMP) path restoration thread:

```
# vxdmpadm stop restore
```

Warning: Automatic path failback stops if the path restoration thread is stopped.

Displaying the status of the DMP path restoration thread

Use the `vxdmpadm gettune` command to display the tunable parameter values that show the status of the Dynamic Multi-Pathing (DMP) path restoration thread. These tunables include:

dmp_restore_state	the status of the automatic path restoration kernel thread.
dmp_restore_interval	the polling interval for the DMP path restoration thread.
dmp_restore_policy	the policy that DMP uses to check the condition of paths.

To display the status of the DMP path restoration thread

- ◆ Use the following commands:

```
# vxdmpadm gettune dmp_restore_state

# vxdmpadm gettune dmp_restore_interval

# vxdmpadm gettune dmp_restore_policy
```

Configuring Array Policy Modules

Dynamic Multi-Pathing (DMP) provides Array Policy Modules (APMs) for use with an array. An APM is a dynamically loadable kernel module (or plug-in) that defines array-specific procedures and commands to:

- Select an I/O path when multiple paths to a disk within the array are available.
- Select the path failover mechanism.
- Select the alternate path in the case of a path failure.
- Put a path change into effect.
- Respond to SCSI reservation or release requests.

DMP supplies default procedures for these functions when an array is registered. An APM may modify some or all of the existing procedures that DMP provides, or that another version of the APM provides.

You can use the following command to display all the APMs that are configured for a system:

```
# vxdmpadm listapm all
```

The output from this command includes the file name of each module, the supported array type, the APM name, the APM version, and whether the module is currently loaded and in use.

To see detailed information for an individual module, specify the module name as the argument to the command:

```
# vxdmpadm listapm module_name
```

To add and configure an APM, use the following command:

```
# vxdmpadm -a cfgapm module_name [attr1=value1 \
      [attr2=value2 ...]]
```

The optional configuration attributes and their values are specific to the APM for an array. Consult the documentation from the array vendor for details.

Note: By default, DMP uses the most recent APM that is available. Specify the `-u` option instead of the `-a` option if you want to force DMP to use an earlier version of the APM. The current version of an APM is replaced only if it is not in use.

Specify the `-r` option to remove an APM that is not currently loaded:

```
# vxdmpadm -r cfgapm module_name
```

See the `vxdmpadm`(1M) manual page.

Configuring latency threshold tunable for metro/geo array

The Metro/Geo cluster system has visibility for both paths, local and remote. Dynamic Multi-Pathing (DMP) A/A array IO policy distributes IO evenly across both of these paths, local and remote. Also, Metro/Geo support serves IO through the local path if the remote paths have higher latency than latency_threshold_difference.

Tunable lets you manage the parameter values of latency_threshold_difference. The default value of the latency_threshold_difference of the tunable parameter is 200 µs. The latency_threshold_difference tunable is valid only for the METRO type of array.

You can assign a new latency threshold difference value to a specified enclosure. Using tunable option you can increase or decrease the latency_threshold_difference value.

The following is the syntax for the tuning latency_threshold_difference:

```
# vxdmpadm getattr {enclosure enc-name}
\  latency_threshold_difference
```

Use the following command `vxdmpadm getattr` that displays the latency_threshold_difference value that is set for an enclosure **emc0**:

```
# vxdmpadm getattr enclosure emc0 latency_threshold_difference
ENCLR_NAME  default  CURRENT
============================================
emc0        200      200
```

To set new value, you can use the following syntax for the specified enclosure:

```
#  vxdmpadm setattr enclosure <enclr-name>
{name=<value> | \  latency_threshold_difference=<value>}
```

The following example shows the `vxdmpadm setattr` command to set the latency_threshold_difference value that is set for an enclosure **emc0:**

```
#  vxdmpadm setattr enclosure emc0 latency_threshold_difference=300
```

Note: The latency_threshold_difference is applicable for enclosure only.

DMP serves IO with local paths only if remote path has higher IO latency, then marks it as STANDBY.

```
#vxdmpadm getsubpaths dmpnodename=emc0_00d2
```

Typical output from the`vxdmpadm getsubpaths`command is as follows:

```
NAME STATE[A]  PATH-TYPE[M] CLTR-NAME ENCLR-TYPE ENCLR-NAME ATTRS  PRIORITY
===========================================================================
Sdar ENABLED       -           c9       EMC         emc0     STANDBY    -
Sdbj ENABLED(A)    -           c9       EMC         emc0        -       -
Sdh  ENABLED(A)    -           c9       EMC         emc0        -       -
Sdz  ENABLED       -           c9       EMC         emc0     STANDBY    -
```

See the `vxdmpadm` (1M) manual page.

Chapter 10

Dynamic Reconfiguration of devices

This chapter includes the following topics:

About online dynamic reconfiguration

You can perform the following kinds of online dynamic reconfigurations:

- Reconfiguring a LUN online that is under DMP control
  See "Manually reconfiguring a LUN online that is under DMP control" on page 268.
- Updating the array controller firmware, also known as a nondisruptive upgrade
  See "Upgrading the array controller firmware online" on page 279.

Reconfiguring a LUN online that is under DMP control using the Dynamic Reconfiguration tool

Perform the following tasks to reconfigure a LUN online that is under DMP control using the Dynamic Reconfiguration tool:

Table 10-1

Task	Topic
Removing LUNs dynamically from an existing target ID	See "Removing LUNs dynamically from an existing target ID" on page 260.
Adding LUNs dynamically to a new target ID	See "Adding new LUNs dynamically to a target ID" on page 263.
Replacing a LUN on an existing target ID	See "Replacing LUNs dynamically from an existing target ID" on page 266.
Changing the LUN characteristics	See "Changing the characteristics of a LUN from the array side" on page 277.

Removing LUNs dynamically from an existing target ID

Dynamic Multi-Pathing (DMP) provides a Dynamic Reconfiguration tool to simplify the removal of LUNs from an existing target ID. Each LUN is unmapped from the host. DMP issues an operating system device scan and cleans up the operating system device tree.

Warning: Do not run any device discovery operations outside of the Dynamic Reconfiguration tool until the device operation is completed.

In a cluster, perform the steps on all nodes in the cluster.

To remove LUNs dynamically from an existing target ID

1 Stop all applications and volumes that are hosted on the LUNs that are to be removed.

   If the device is in use by Veritas Volume Manager (VxVM), perform the following steps:

   - If the device is part of a disk group, move the disk out of the disk group.

     ```
     # vxdg -g dgname rmdisk daname
     ```

   - Remove the disk from the `vxdisk` list.
     In a cluster, perform this step from all of the nodes.

     ```
     # vxdisk rm da-name
     ```

     For example:

     ```
     # vxdisk rm eva4k6k0_0
     ```

For LUNs using Linux LVM over DMP devices, remove the device from the LVM volume group.

```
# vgreduce vgname
      devicepath
```

2. Start the `vxdiskadm` utility:

```
# vxdiskadm
```

3. Select the **Dynamic Reconfiguration operations** option from the `vxdiskadm` menu.

4. Select the **Remove LUNs** option.

5. Type **list** or press **Return** to display a list of LUNs that are available for removal. A LUN is available for removal if it is not in use.

   The following shows an example output:

```
Select disk devices to remove: [<pattern-list>,all,list]: list
LUN(s) available for removal:
eva4k6k0_0
eva4k6k0_1
eva4k6k0_2
eva4k6k0_3
eva4k6k0_4
emc0_0119
```

6. Enter the name of a LUN, a comma-separated list of LUNs, or a regular expression to specify the LUNs to remove.

   For example, enter `emc0_0119`.

```
Select disk devices to Remove: [<pattern-list>,all,list,
file=<filename>,q] (default:list): emc0_0119
```

7. At the prompt, confirm the LUN selection.

   DMP removes the LUN from VxVM usage.

8 At the following prompt, remove the LUN from the array/target.

```
Remove Luns
Menu: VolumeManager/Disk/DynamicReconfigurationOperations/RemoveLuns

INFO: Removing Lun [emc0_0119] from VxVM
INFO: LUN [emc0_0119] removed successfully from VxVM.
---------------------------------------------------
Enclosure=emc0 AVID=0119
Device=emc0_0119 Serial=2200119000
PATH=sdad ctlr=c11 port=16c-0 [-]
PATH=sdah ctlr=c12 port=16c-0 [-]
PATH=sdaj ctlr=c12 port=16c-1 [-]
PATH=sdaf ctlr=c11 port=16c-1 [-]
---------------------------------------------------
Please remove LUNs with Above details from array and press 'y' to
continue removal or 'q' to quit :
```

9 The following are sample EMC Symmetrix commands:

```
# symmask -sid 822 -wwn 2001000e1ec307de -dir 16c -p 0 remove devs 0119
# symmask -sid 822 -wwn 2001000e1ec307de -dir 16c -p 1 remove devs 0119
# symmask -sid 822 -wwn 2001000e1ec307df -dir 16c -p 0 remove devs 0119
# symmask -sid 822 -wwn 2001000e1ec307df -dir 16c -p 1 remove dev 0119

# symmask -sid 822 refresh -nopr

Symmetrix FA/SE directors updated with contents of SymMask
Database 000290300822

When complete, respond to previous array prompt.

Please remove LUNs with Above details from array and
press 'y' to continue removal or 'q' to quit : y
```

10 DMP completes the removal of the device from VxVM usage. Output similar to the following is displayed:

```
Remove Luns
Menu: VolumeManager/Disk/DynamicReconfigurationOperations/RemoveLuns

INFO: Checking/Removing stale device entries (if any).
INFO: Refreshing OS device Tree
INFO: Updating VxVM device tree
---------------------------------------------------------
Luns Removed
---------------------------------------------------------
emc0_0119
---------------------------------------------------------

Press <Enter> or <Return> to continue:
```

11 Specify the dynamic reconfiguration operation to be done:

```
Specify Dynamic Reconfiguration Operation to be done:
Menu: VolumeManager/Disk/DynamicReconfigurationOperations

 1 Add Luns
 2 Remove Luns
 3 Replace Luns
 4 Replace HBA


 ? Display help about menu
 ?? Display help about the menuing system
 q Exit
```

To exit the Dynamic Reconfiguration tool, enter: `q`

Adding new LUNs dynamically to a target ID

Dynamic Multi-Pathing (DMP) provides a Dynamic Reconfiguration tool to simplify the addition of new LUNs to a new or existing target ID. One or more new LUNs are mapped to the host by way of multiple HBA ports. An operating system device scan is issued for the LUNs to be recognized and added to DMP control.

Warning: Do not run any device discovery operations outside of the Dynamic Reconfiguration tool until the device operation is completed.

In a cluster, perform the steps on all the nodes in the cluster.

To add new LUNs dynamically to a target ID

1 Start the `vxdiskadm` utility:

```
# vxdiskadm
```

2 Select the **Dynamic Reconfiguration operations** option from the `vxdiskadm` menu.

3 Select the **Add LUNs** option.

Output similar to the following is displayed:

```
Add Luns
Menu: VolumeManager/Disk/DynamicReconfigurationOperations/AddLuns

INFO: Refreshing OS device Tree
INFO: Updating VxVM device tree
Add LUNs from array, once done then press 'y' to continue
or 'q' to quit. :
```

4 The following are sample EMC Symmetrix commands:

```
# symmask -sid 822 -wwn 2001000e1ec307de -dir 16c -p 0 add devs
0119 -nopr
# symmask -sid 822 -wwn 2001000e1ec307de -dir 16c -p 1 add devs
0119 -nopr
# symmask -sid 822 -wwn 2001000e1ec307df -dir 16c -p 0 add devs
0119 -nopr
# symmask -sid 822 -wwn 2001000e1ec307df -dir 16c -p 1 add devs
0119 -nopr
# symmask -sid 822 refresh -nopr

Symmetrix FA/SE directors updated with contents of SymMask
Database 000290300822
```

5. When the prompt displays, add the LUNs from the array.

   Output similar to the following is displayed:

   ```
   Add LUNs from array, once done then press 'y' to continue
   or 'q' to quit. : y

   Add Luns
   Menu: VolumeManager/Disk/DynamicReconfigurationOperations/AddLuns

   INFO: Refreshing OS device Tree
   INFO: Updating VxVM device tree
   INFO: Updating partition table information and disk size
   INFO: Number of Paths for Lun [emc0_0119] presented=4
   INFO: Updating VxVM device tree
   ```

6. Select **y** to continue to add the LUNs to DMP.

   DMP updates the operating system device tree and the VxVM device tree. The newly-discovered devices are now visible.

   ```
   ---------------------------------------------------------
   Luns Added
   ---------------------------------------------------------
   Enclosure=emc0 AVID=0119
   Device=emc0_0119 Serial=2200119000
   PATH=sdaf ctlr=c11 port=16c-1 [-]
   PATH=sdah ctlr=c12 port=16c-0 [-]
   PATH=sdaj ctlr=c12 port=16c-1 [-]
   PATH=sdad ctlr=c11 port=16c-0 [-]
   ---------------------------------------------------------

   Press <Enter> or <Return> to continue:
   ```

7. Specify the dynamic reconfiguration operation to be done:

   ```
   Specify Dynamic Reconfiguration Operation to be done:
   Menu: VolumeManager/Disk/DynamicReconfigurationOperations

    1 Add Luns
    2 Remove Luns
    3 Replace Luns
    4 Replace HBA


    ? Display help about menu
    ?? Display help about the menuing system
    q Exit

   Select an operation to perform : q
   ```

   To exit the Dynamic Reconfiguration tool, enter: `q`

Replacing LUNs dynamically from an existing target ID

Dynamic Multi-Pathing (DMP) provides a Dynamic Reconfiguration tool to simplify the replacement of new LUNs from an existing target ID. Each LUN is unmapped from the host. DMP issues an operating system device scan and cleans up the operating system device tree.

Warning: Do not run any device discovery operations outside of the Dynamic Reconfiguration tool until the device operation is completed.

In a cluster, perform the steps on all the nodes in the cluster.

To replace LUNs dynamically from an existing target ID

1. Stop all applications and volumes that are hosted on the LUNs that are to be removed.

   If the device is in use by Veritas Volume Manager (VxVM), perform the following steps:

   - If the device is part of a disk group, move the disk out of the disk group.

     ```
     # vxdg -g dgname rmdisk daname
     ```

   - Remove the disk from the `vxdisk` list.
     In a cluster, perform this step from all of the nodes.

     ```
     # vxdisk rm da-name
     ```

     For example:

     ```
     # vxdisk rm eva4k6k0_0
     ```

   For LUNs using Linux LVM over DMP devices, remove the device from the LVM volume group

   ```
   # vgreduce vgname
         devicepath
   ```

2. Start the `vxdiskadm` utility:

   ```
   # vxdiskadm
   ```

3. Select the **Dynamic Reconfiguration operations** option from the `vxdiskadm` menu.

4. Select the **Replace LUNs** option.

   The output displays a list of LUNs that are available for replacement. A LUN is available for replacement if there is no open on the LUN, and the state is online or nolabel.

5. Select one or more LUNs to replace.

6. At the prompt, confirm the LUN selection.

7 Remove the LUN from the array/target.

8 Return to the Dynamic Reconfiguration tool and select `y` to continue the removal.

After the removal completes successfully, the Dynamic Reconfiguration tool prompts you to add a LUN.

9 When the prompt displays, add the LUNs from the array/target.

10 Select **y** to continue to add the LUNs.

DMP updates the operating system device tree and the VxVM device tree. The newly-discovered devices are now visible.

Replacing a host bus adapter online

Dynamic Multi-Pathing (DMP) provides a Dynamic Reconfiguration tool to simplify the removal of host bus adapters from an existing system.

To replace a host bus adapter online

1 Start the `vxdiskadm` utility:

```
# vxdiskadm
```

2 Select the **Dynamic Reconfiguration operations** option from the `vxdiskadm` menu.

3 Select the **Replace HBAs** option.

The output displays a list of HBAs that are available to DMP.

4 Select one or more HBAs to replace.

5 At the prompt, confirm the HBA selection.

6 Replace the host bus adapter.

7 Return to the Dynamic Reconfiguration tool and select `y` to continue the replacement process.

DMP updates the operating system device tree.

Manually reconfiguring a LUN online that is under DMP control

Dynamic LUN reconfigurations require array configuration commands, operating system commands, and Veritas Volume manager commands. To complete the operations correctly, you must issue the commands in the proper sequence on the host.

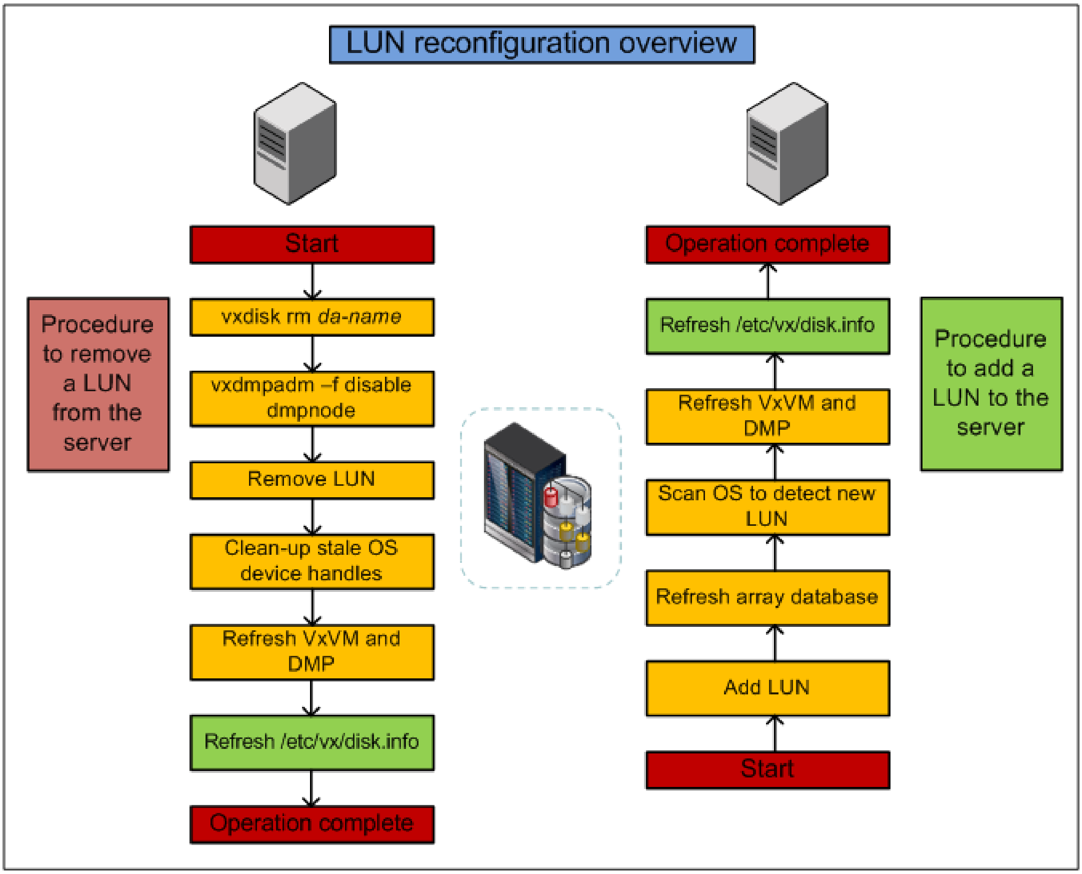
Overview of manually reconfiguring a LUN

This section only provides an overview of the prechecks and the procedure to manually add or remove a LUN. The procedures have been elaborately documented in the topics listed in the following table:

Table 10-2

Task	Topic
Removing LUN dynamically from an existing target ID	See “Manually removing LUNs dynamically from an existing target ID” on page 272.
Cleaning up the operating system device tree after removing LUNs	See “Manually cleaning up the operating system device tree after removing LUNs” on page 277.
Scanning an operating system device tree after adding or removing LUNs	See “Scanning an operating system device tree after adding or removing LUNs” on page 276.
Adding LUN dynamically to a new target ID	See “Manually adding new LUNs dynamically to a new target ID” on page 274.
Changing the LUN characteristics	See “Changing the characteristics of a LUN from the array side” on page 277.

Figure 10-1 LUN reconfiguration overview


LUN reconfiguration overview
Procedure to remove a LUN from the server
Start
vxdisk rm *da-name*
vxdmpadm –f disable dmpnode
Remove LUN
Clean-up stale OS device handles
Refresh VxVM and DMP
Refresh /etc/vx/disk.info
Operation complete
Procedure to add a LUN to the server
Start
Add LUN
Refresh array database
Scan OS to detect new LUN
Refresh VxVM and DMP
Refresh /etc/vx/disk.info
Operation complete

Prechecks

Perform the following prechecks before manually reconfiguring a LUN:

Table 10-3 Prechecks

Task	Command
Check the `/etc/vx/disk.info` file	`# grep “0xffff” /etc/vx/disk.info`

Refresh the OS layer	`# echo '- - -' >`
	`/sys/class/scsi_host/host$i/scan`

List OS device handles	`# lsscsi`
Refresh VxVM and DMP	`# vxdisk scandisks`

Table 10-3 Prechecks *(continued)*

Task	Command
Refresh DDL layer/dev_t (device number) list	# vxddladm assign names

Note: Ensure that the OS and VxVM are both clean prior to provisioning any new LUNs.

Manually removing a LUN

Perform the following steps to manually remove a LUN:

Table 10-4 LUN removal steps

Task	Validation
Unmount file system (s)	Confirm whether the disk has been removed from the disk group.
Close the VxVM device: # vxdisk rm *da-name*	Confirm whether the VxVM device has been closed: # vxdisk list
Disable DMP paths: # vxdmpadm -f disable dmpnodename=*da-name*	Confirm whether the DMP paths have been disabled: # vxdmpadm getsubpaths dmpnodename=*da-name*
Mask LUN from the server	Confirm whether the LUN has been removed at the array level.
Clean-up OS device handles: # echo 1 > /sys/block/*device_name*/device/delete	Confirm whether OS device handles are clean: # lsscsi
Refresh VxVM and DMP: # vxdisk scandisks	
Refresh DDL layer/dev_t (device number) list: # vxddladm assign names	

Manually adding a LUN

To manually add a LUN

1. Mask LUN to HBA worldwide name (WWN) in the server.
2. Refresh the array database.
3. Refresh OS device handles.
4. Refresh VxVM and DMP.
5. Refresh the `/etc/vx/disk.info` file.

Manually removing LUNs dynamically from an existing target ID

In this case, a group of LUNs is unmapped from the host HBA ports and an operating system device scan is issued. To add subsequent LUNs seamlessly, perform additional steps to clean up the operating system device tree.

The high-level procedure and the SF commands are generic. However, the operating system commands may vary depending on the Linux version. For example, the following procedure uses Linux Suse10.

To remove LUNs dynamically from an existing target ID

1. Prior to any dynamic reconfiguration, ensure that the `dmp_cache_open` tunable is set to `on`. This setting is the default.

   ```
   # vxdmpadm gettune dmp_cache_open
   ```

   If the tunable is set to `off`, set the `dmp_cache_open` tunable to `on`.

   ```
   # vxdmpadm settune dmp_cache_open=on
   ```

2. Identify which LUNs to remove from the host. Do one of the following:
   - Use Storage Array Management to identify the Array Volume ID (AVID) for the LUNs.
   - If the array does not report the AVID, use the LUN index.
3. For LUNs under VxVM, perform the following steps:
   - Evacuate the data from the LUNs using the `vxevac` command.
     See the `vxevac`(1M) online manual page.
     After the data has been evacuated, enter the following command to remove the LUNs from the disk group:

     ```
     # vxdg -g diskgroup rmdisk da-name
     ```

- If the data has not been evacuated and the LUN is part of a subdisk or disk group, enter the following command to remove the LUNs from the disk group. If the disk is part of a shared disk group, you must use the `-k` option to force the removal.

  ```
  # vxdg -g diskgroup -k rmdisk da-name
  ```

4 For LUNs using Linux LVM over DMP devices, remove the device from the LVM volume group

```
# vgreduce vgname
      devicepath
```

5 Using the AVID or LUN index, use Storage Array Management to unmap or unmask the LUNs you identified in step 2.

6 Remove the LUNs from the `vxdisk` list. Enter the following command on all nodes in a cluster:

```
# vxdisk rm da-name
```

This is a required step. If you do not perform this step, the DMP device tree shows ghost paths.

7 Clean up the Linux SCSI device tree for the devices that you removed in step 6.

See "Manually cleaning up the operating system device tree after removing LUNs" on page 277.

This step is required. You must clean up the operating system SCSI device tree to release the SCSI target ID for reuse if a new LUN is added to the host later.

8 Scan the operating system device tree.

See "Scanning an operating system device tree after adding or removing LUNs" on page 276.

9 Use SF to perform a device scan. You must perform this operation on all nodes in a cluster. Enter one of the following commands:

- `# vxdctl enable`

- `# vxdisk scandisks`

10 Refresh the DMP device name database using the following command:

```
# vxddladm assign names
```

11 Verify that the LUNs were removed cleanly by answering the following questions:

- Is the device tree clean?
  Verify that the operating system metanodes are removed from the `/sys/block` directory.
- Were all the appropriate LUNs removed?
  Use the DMP disk reporting tools such as the `vxdisk list` command output to determine if the LUNs have been cleaned up successfully.
- Is the `vxdisk list` output correct?
  Verify that the `vxdisk list` output shows the correct number of paths and does not include any ghost disks.

If the answer to any of these questions is "No," return to step 5 and perform the required steps.

If the answer to all of the questions is "Yes," the LUN remove operation is successful.

Manually adding new LUNs dynamically to a new target ID

In this case, a new group of LUNs is mapped to the host via multiple HBA ports. An operating system device scan is issued for the LUNs to be recognized and added to DMP control.

The high-level procedure and the SF commands are generic. However, the operating system commands may vary depending on the Linux version. For example, the following procedure uses Linux Suse10.

To add new LUNs dynamically to a new target ID

1 Prior to any dynamic reconfiguration, ensure that the `dmp_cache_open` tunable is set to `on`. This setting is the default.

```
# vxdmpadm gettune dmp_cache_open
```

If the tunable is set to `off`, set the `dmp_cache_open` tunable to `on`.

```
# vxdmpadm settune dmp_cache_open=on
```

2 Identify which LUNs to add to the host. Do one of the following:

- Use Storage Array Management to identify the Array Volume ID (AVID) for the LUNs.
- If the array does not report the AVID, use the LUN index.

3 Map/mask the LUNs to the new target IDs on multiple hosts.

4 Scan the operating system device.

See "Scanning an operating system device tree after adding or removing LUNs" on page 276.

Repeat step 2 and step 3 until you see that all the LUNs have been added.

5 Use SF to perform a device scan. You must perform this operation on all nodes in a cluster. Enter one of the following commands:

-
  ```
  # vxdctl enable
  ```

-
  ```
  # vxdisk scandisks
  ```

6 Refresh the DMP device name database using the following command:

```
# vxddladm assign names
```

7 Verify that the LUNs were added correctly by answering the following questions:

- Do the newly provisioned LUNs appear in the `vxdisk list` output?
- Are the configured paths present for each LUN?

If the answer to any of these questions is "No," return to step 2 and begin the procedure again.

If the answer to all of the questions is "Yes," the LUNs have been successfully added. You can now add the LUNs to a disk group, create new volumes, or grow existing volumes.

If the dmp_native_support tunable is set to ON and the new LUN does not have a VxVM label or is not claimed by a TPD driver then the LUN is available for use by LVM.

About detecting target ID reuse if the operating system device tree is not cleaned up

If you try to reprovision a LUN or set of LUNs whose previously-valid operating system device entries are not cleaned up, the following messages are displayed. Also, DMP reconfiguration during the DMP device scan and DMP reconfiguration are temporarily inhibited.

See “Manually cleaning up the operating system device tree after removing LUNs” on page 277.

```
VxVM vxdisk ERROR V-5-1-14519 Data Corruption Protection Activated
- User Corrective Action Needed

VxVM vxdisk INFO V-5-1-14521 To recover, first ensure that the OS
device tree is up to date (requires OS specific commands).

VxVM vxdisk INFO V-5-1-14520 Then, execute 'vxdisk rm' on the
following devices before reinitiating device discovery. <DA names>
```

The message above indicates that a new LUN is trying to reuse the target ID of an older LUN. The device entries have not been cleaned, so the new LUN cannot use the target ID. Until the operating system device tree is cleaned up, DMP prevents this operation.

Scanning an operating system device tree after adding or removing LUNs

After you add or remove LUNs, scan the operating system device tree to verify that the operation completed successfully.

Linux provides several methods for rescanning the SCSI bus and identifying the devices mapped to it. These methods include the following:

- The SCSI scan function in the `/sys` directory
- HBA vendor utilities

To scan using the SCSI scan function

◆ Enter the following command:

```
# echo '- - -' > /sys/class/scsi_host/host$i/scan
```

where the three dashes refer to the channel, target, and LUN numbers, and *host$i* is the host bus adapter instance. This example scans every channel, target, and LUN visible via this host bus adapter instance.

To scan using HBA vendor utilities

◆ Follow the vendor's instructions for the HBA utility. Examples include the following:

- QLogic provides a script that dynamically scans for newly-added LUNs. You can download it from the QLogic Web site. To run the script, enter the following command:

```
# ./ql-dynamic-tgt-lun-disc.sh
```

- Emulex provides an HBAnywhere script. You can download it from the Emulex web site. The script has a LUN Scan Utility that dynamically scans for newly-added LUNs. To run the utility, enter the following command:

  ```
  # lun_scan all
  ```

Manually cleaning up the operating system device tree after removing LUNs

After you remove LUNs, you must clean up the operating system device tree.

The operating system commands may vary, depending on the Linux version. The following procedure uses SUSE 10. If any of these steps do not produce the desired result, contact Novell support.

To clean up the operating system device tree after removing LUNs

1. Remove the device from the operating system database. Enter the following command:

   ```
   # echo 1 > /sys/block/$PATH_SYS/device/delete
   ```

   where *PATH_SYS* is the name of the device you want to remove.

2. When you enter the following command, no devices should be displayed. This step verifies that the LUNs have been removed.

   ```
   # lsscsi | grep PATH_SYS
   ```

3. After you remove the LUNS, clean up the device. Enter the following command:

   ```
   # echo "- - -" > /sys/class/scsi_host/host$I/scan
   ```

   where the three dashes refer to the channel, target, and LUN numbers, and *host$i* is the host bus adapter instance. This example cleans up every channel, target, and LUN visible via this host bus adapter instance.

Changing the characteristics of a LUN from the array side

Some arrays provide a way to change the properties of LUNs. In most cases, you must completely stop usage of the device before the device shows the changed characteristics. We recommend taking the device offline before changing the LUN properties, and bringing the device back online again afterwards.

In certain cases, such as EMC BCV and SRDF operations, the device can remain online during this procedure.

In a cluster, perform the steps on all the nodes in the cluster.

To change the properties of a LUN

1. Stop all applications and volumes that are hosted on the device.

   If the device is in use by Veritas Volume Manager (VxVM), perform the following steps:

   - If the device is part of a disk group, move the disk out of the disk group.

     ```
     # vxdg -g dgname rmdisk da_name
     ```

   - Bring the disk offline.
     In a cluster, perform this step from all of the nodes.

     ```
     # vxdisk offline da_name
     ```

     For example:

     ```
     # vxdisk offline eva4k6k0_0
     ```

   For LUNs using Linux LVM over DMP devices, remove the device from the LVM volume group

   ```
   # vgreduce vgname
         devicepath
   ```

2. Change the LUN characteristics.
3. Bring the device online.

   For a Veritas Volume Manager disk:

   - Bring the device online.

     ```
     # vxdisk online da_name
     ```

   - Add the disk back into the disk group.

     ```
     # vxdg -g dgname adddisk da_name
     ```

   For LUNs using Linux LVM over DMP devices, add the device back into the LVM volume group

```
# vgreduce vgname
     devicepath
```

4 Use DMP to perform a device scan.

In a cluster, perform this command on all the nodes.

```
# vxdisk scandisks
```

Upgrading the array controller firmware online

Storage array subsystems need code upgrades as fixes, patches, or feature upgrades. You can perform these upgrades online when the file system is mounted and I/Os are being served to the storage.

Storage subsystems contain multiple controllers for redundancy. An online upgrade is done one controller at a time. Dynamic Multi-Pathing (DMP) fails over all I/O to an alternate controller while one of the controllers is undergoing an Online Controller Upgrade. After the controller has completely staged the code, it reboots, resets, and comes online with the new version of the code. The other controller goes through the same process, and I/O fails over to the alternate controller.

Note: Throughout this process, application I/O is not affected.

Array vendors have different names for this process. For example, EMC calls it a nondisruptive upgrade (NDU) for CLARiiON arrays.

A/A type arrays require no special handling during this online upgrade process. For A/P, A/PF, and ALUA type arrays, DMP performs array-specific handling through vendor-specific array policy modules (APMs) during an online controller code upgrade.

When a controller resets and reboots during a code upgrade, DMP detects this state through the SCSI status. DMP immediately fails over all I/O to the next controller.

If the array does not fully support NDU, all paths to the controllers may be unavailable for I/O for a short period of time. Before beginning the upgrade, set the `dmp_lun_retry_timeout` tunable to a period greater than the time that you expect the controllers to be unavailable for I/O. DMP does not fail the I/Os until the end of the `dmp_lun_retry_timeout` period, or until the I/O succeeds, whichever happens first. Therefore, you can perform the firmware upgrade without interrupting the application I/Os.

For example, if you expect the paths to be unavailable for I/O for 300 seconds, use the following command:

```
# vxdmpadm settune dmp_lun_retry_timeout=300
```

DMP does not fail the I/Os for 300 seconds, or until the I/O succeeds.

To verify which arrays support Online Controller Upgrade or NDU, refer to the hardware compatibility list (HCL).

Reformatting NVMe devices manually

You can modify the sector size of NVMe devices by removing the device from VxVM and reformatting it.

To reformat NVMe devices manually

1 Take the disk offline.

```
# vxdisk offline r720xd-114217_intel_nvme0_0
```

2 Remove the device from VxVM.

```
# vxdisk rm r720xd-114217_intel_nvme0_0
```

3 Removing the NVMe device from the operating system.

```
# echo 1 > /sys/block/nvme0n1/device/device/remove
```

4 Refresh the VxVM device tree.

```
# vxdisk scandisks
```

5 Verify that device is not present.

```
# vxdisk list | grep nvme
```

6 Rescan the NVMe PCI device to add it to the operating system device tree.

```
# echo 1 > /sys/bus/pci/rescan
# echo 1 > /sys/bus/pci/drivers/nvme/0000\:05\:00.0/rescan
```

7 Format the NVMe device to the required sector size using the Intel® SSD Data Center Tool (ISDCT).

```
# isdct start -intelssd 0 -nvmeformat LBAFormat=3 SecureEraseSetting=0 \
ProtectionInformation=0 MetadataSettings=0
WARNING! You have selected to format the drive!
Proceed with the format? (Y|N): y
Formatting...

- Intel SSD DC P3700 Series CVFT5456000V2P0EGN -

Status : NVMeFormat successful.
```

8 Refresh the VxVM device tree.

```
# vxdisk scandisks
```

9 Verify the device.

```
# vxdisk list | grep nvme
r720xd-114217_intel_nvme0_0 auto:none  -   - online invalid
```

Chapter 11

Managing devices

This chapter includes the following topics:

Displaying disk information

Before you use a disk, you need to know if it has been initialized and placed under Veritas Volume Manager (VxVM) control. You also need to know if the disk is part of a disk group, because you cannot create volumes on a disk that is not part of a disk group. The `vxdisk list` command displays device names for all recognized disks, the disk names, the disk group names associated with each disk, and the status of each disk.

To display information on all disks that are known to VxVM

- Use the following command:

  ```
  # vxdisk list
  ```

  VxVM displays output similar to the following:

  ```
  DEVICE        TYPE            DISK    GROUP   STATUS
  emc_clariion0_26    auto:cdsdisk      -       -     online
  emc_clariion0_27    auto:cdsdisk      -       -     online
  emc_clariion0_107   auto:cdsdisk    dsk0    tcdg    online
  emc_clariion0_108   auto:cdsdisk    dsk1    tcdg    online
  emc_clariion0_110   auto:cdsdisk    dsk2    tcdg    online
  emc_clariion0_111   auto:cdsdisk    dsk3    tcdg    online
  emc_clariion0_144   auto:none        -        -     online invalid
  ```

  The phrase `online invalid` in the `STATUS` line indicates that a disk has not yet been added to VxVM control. These disks may or may not have been initialized by VxVM previously. Disks that are listed as `online` are already under VxVM control.

To display information about an individual disk

- Use the following command:

  ```
  # vxdisk [-v] list diskname
  ```

  The `-v` option causes the command to additionally list all tags and tag values that are defined for the disk. By default, tags are not displayed.

VxVM leverages 4K sector device support by using Media Format Discovery, `-o mfd` option along with the existing `vxdisk list` command.

About Media Format Discovery

The media format discovery or the `-o mfd` option helps to identify the correct format of disk and also prevents an accidental erasure of disk when the disk is moved within different operating system environments.

Viewing information about the native layouts of operating system

To view information about the native layouts across operating systems, use the following command:

```
# vxdisk -o mfd list
```

Following examples illustrates the importance of `-o mfd` option with `vxdisk list` for different operating system environments:

Example 1: In Linux environment

With `-o mfd` option:

```
# vxdisk -o mfd list sda | egrep "^info:|flags:"

info:      format=linux:ext4

flags:     unusable online ready private autoconfig invalid
```

Without `-o mfd` option:

```
# vxdisk list sda | egrep "^info:|flags:"

info:      format=none

flags:     online ready private autoconfig invalid
```

Example 2: In Solaris environment

With `-o mfd` option:

```
# vxdisk -o mfd list disk_0 | egrep "^info:|flags:"

info:      format=solaris:ZFS

flags:     unusable online ready private autoconfig invalid
```

Without `-o mfd` option:

```
# vxdisk list disk_0 | egrep "^info:|flags:"

info:      format=ZFS

flags:     ZFS online ready private autoconfig invalid
```

Note: Veritas Volume Manager cannot directly initialize the disks which are flagged as **unusable**. In this case, the `vxdisk init` command fails with an error message and requests you to remove the foreign format.

For example:

```
#vxdisk init disk_0

VxVM vxdisk ERROR V-5-1-0 Device disk_0 is in use.
If it is still desired to initialize this device for VxVM use, please remove
```

```
the foreign format signatures from each of the following partition(s)
using 'dd' command or some other tool.
PARTITION  TYPE
0          solaris:zfs
```

Viewing information about the disk sector size

To view information about the disk sector size, you can use the `vxdisk list` and `vxmediadisc` commands.

See the following examples for more details:

Example 1: Using the `vxdisk list` command

```
# vxdisk list sdz | grep "^iosize"

iosize:    min=4096 (bytes) max=1024 (blocks)
```

Example 2: Using the `vxmediadisc` command

```
# /etc/vx/diag.d/vxmediadisc -p /dev/sdz | grep "Sector size"

vxmediadisc: Sector size:   4096
```

Supported operating systems

Viewing information about the native layouts of operating system and disk sector size is supported on the following operating systems:

1. Linux (RHEL, SLES, and supported RHEL compatible distributions)

Displaying disk information with vxdiskadm

Veritas Volume Manager (VxVM) enables you to see disk information using the `vxdiskadm` program. Disk information shows you which disks are initialized, to which disk groups they belong, and the disk status. The `list` option displays device names for all recognized disks, the disk names, the disk group names associated with each disk, and the status of each disk.

To display disk information

1. Start the `vxdiskadm` program, and select `list (List disk information)` from the main menu.
2. At the following prompt, enter the name of the device you want to see, or enter `all` for a list of all devices:

```
List disk information
Menu: VolumeManager/Disk/ListDisk

VxVM INFO V-5-2-475 Use this menu operation to display a list of
disks. You can also choose to list detailed information about
the disk at a specific disk device address.

Enter disk device or "all" [<address>,all,q,?] (default: all)
```

- If you enter `all`, VxVM displays the device name, disk name, group, and status of all the devices.
- If you enter the name of a device, VxVM displays complete disk information (including the device name, the type of disk, and information about the public and private areas of the disk) of that device.

Once you have examined this information, press **Return** to return to the main menu.

Changing the disk device naming scheme

You can either use enclosure-based naming for disks or the operating system’s naming scheme. DMP commands display device names according to the current naming scheme.

The default naming scheme is enclosure-based naming (EBN).

When you use Dynamic Multi-Pathing (DMP) with native volumes, the disk naming scheme must be EBN, the `use_avid` attribute must be `yes`, and the persistence attribute must be set to `yes`.

To change the disk-naming scheme

- Select `Change the disk naming scheme` from the `vxdiskadm` main menu to change the disk-naming scheme that you want SF to use. When prompted, enter `y` to change the naming scheme.

  OR

  Change the naming scheme from the command line. Use the following command to select enclosure-based naming:

  ```
  # vxddladm set namingscheme=ebn [persistence={yes|no}] \
  [use_avid={yes|no}] [lowercase={yes|no}]
  ```

  Use the following command to select operating system-based naming:

  ```
  # vxddladm set namingscheme=osn [persistence={yes|no}] \
  [lowercase=yes|no]
  ```

  The optional `persistence` argument allows you to select whether the names of disk devices that are displayed by SF remain unchanged after disk hardware has been reconfigured and the system rebooted. By default, enclosure-based naming is persistent. Operating system-based naming is not persistent by default.

  To change only the naming persistence without changing the naming scheme, run the `vxddladm set namingscheme` command for the current naming scheme, and specify the persistence attribute.

  By default, the names of the enclosure are converted to lowercase, regardless of the case of the name specified by the ASL. The enclosure-based device names are therefore in lowercase. Set the `lowercase=no` option to suppress the conversion to lowercase.

  For enclosure-based naming, the `use_avid` option specifies whether the Array Volume ID is used for the index number in the device name. By default, `use_avid=yes`, indicating the devices are named as *enclosure_avid.* If `use_avid` is set to `no`, DMP devices are named as *enclosure_index.* The index number is assigned after the devices are sorted by LUN serial number.

  The change is immediate whichever method you use.

  See “Regenerating persistent device names” on page 289.

Displaying the disk-naming scheme

In Dynamic Multi-Pathing (DMP), disk naming can be operating system-based naming or enclosure-based naming.

The following command displays whether the SF disk-naming scheme is currently set. It also displays the attributes for the disk naming scheme, such as whether persistence is enabled.

To display the current disk-naming scheme and its mode of operations, use the following command:

```
# vxddladm get namingscheme
NAMING_SCHEME     PERSISTENCE LOWERCASE USE_AVID
===============================================
Enclosure Based   Yes         Yes       Yes
```

Setting customized names for DMP nodes

The Dynamic Multi-Pathing (DMP) node name is the metadevice name that represents the multiple paths to a disk. The Device Discovery Layer (DDL) generates the DMP node name from the device name according to the Storage Foundation (SF) naming scheme.

You can specify a customized name for a DMP node. User-specified names are persistent even if names persistence is turned off.

You cannot assign a customized name that is already in use by a device. However, if you assign names that follow the same naming conventions as the names that the DDL generates, a name collision can potentially occur when a device is added. If the user-defined name for a DMP device is the same as the DDL-generated name for another DMP device, the `vxdisk list` command output displays one of the devices as 'error'.

To specify a custom name for a DMP node

◆ Use the following command:

```
# vxdmpadm setattr dmpnode dmpnodename name=name
```

You can also assign names from an input file. This enables you to customize the DMP nodes on the system with meaningful names.

To specify a custom name for an enclosure

◆ Use the following command:

```
# vxdmpadm setattr enclosure enc_name name=custom_name
```

To assign DMP nodes from a file

1. To obtain a file populated with the names of the devices in your configuration, use the following command:

   ```
   # vxddladm -l assign names  > filename
   ```

   The sample file shows the format required and serves as a template to specify your customized names.

   You can also use the script `vxgetdmpnames` to get a sample file populated from the devices in your configuration.

2. Modify the file as required. Be sure to maintain the correct format in the file.

3. To assign the names, specify the name and path of the file to the following command:

   ```
   # vxddladm assign names file=pathname
   ```

To clear custom names

◆ To clear the names, and use the default operating system-based naming or enclosure-based naming, use the following command:

```
# vxddladm -c assign names
```

Regenerating persistent device names

The persistent device naming feature makes the names of disk devices persistent across system reboots. The Device Discovery Layer (DDL) assigns device names according to the persistent device name database.

If operating system-based naming is selected, each disk name is usually set to the name of one of the paths to the disk. After hardware reconfiguration and a subsequent reboot, the operating system may generate different names for the paths to the disks. Therefore, the persistent device names may no longer correspond to the actual paths. This does not prevent the disks from being used, but the association between the disk name and one of its paths is lost.

Similarly, if enclosure-based naming is selected, the device name depends on the name of the enclosure and an index number. If a hardware configuration changes the order of the LUNs exposed by the array, the persistent device name may not reflect the current index.

To regenerate persistent device names

- To regenerate the persistent names repository, use the following command:

  ```
  # vxddladm [-c] assign names
  ```

  The `-c` option clears all user-specified names and replaces them with autogenerated names.

  If the `-c` option is not specified, existing user-specified names are maintained, but operating system-based and enclosure-based names are regenerated.

Changing device naming for enclosures controlled by third-party drivers

By default, enclosures controlled by third-party drivers (TPD) use pseudo device names based on the TPD-assigned node names. If you change the device naming to native, the devices are named in the same format as other Storage Foundation (SF) devices. The devices use either operating system names (OSN) or enclosure-based names (EBN), depending on which naming scheme is set.

See "Displaying the disk-naming scheme" on page 287.

To change device naming for TPD-controlled enclosures

- For disk enclosures that are controlled by third-party drivers (TPD) whose coexistence is supported by an appropriate Array Support Library (ASL), the default behavior is to assign device names that are based on the TPD-assigned node names. You can use the `vxdmpadm` command to switch between these names and the device names that are known to the operating system:

  ```
  # vxdmpadm setattr enclosure enclosure_name tpdmode=native|pseudo
  ```

  The argument to the `tpdmode` attribute selects names that are based on those used by the operating system (`native`), or TPD-assigned node names (`pseudo`).

  The use of this command to change between TPD and operating system-based naming is illustrated in the following example for the enclosure named `pp_emc_clariion0`. In this example, the device-naming scheme is set to OSN.

  ```
  # vxdisk list

  DEVICE        TYPE            DISK         GROUP        STATUS
  emcpowerp     auto:cdsdisk    -            -            online
  emcpowerq     auto:cdsdisk    -            -            online
  emcpowerr     auto:cdsdisk    -            -            online
  emcpowers     auto:cdsdisk    -            -            online
  emcpowert     auto:cdsdisk    -            -            online
  ```

```
# vxdmpadm setattr enclosure pp_emc_clariion0 tpdmode=native

# vxdisk list

DEVICE       TYPE            DISK         GROUP        STATUS
sde          auto:cdsdisk    -            -            online
sdf          auto:cdsdisk    -            -            online
sdg          auto:cdsdisk    -            -            online
sdh          auto:cdsdisk    -            -            online
sdi          auto:cdsdisk    -            -            online
```

If `tpdmode` is set to `native`, the path with the smallest device number is displayed.

About the Array Volume Identifier (AVID) attribute

DMP assigns enclosure-based names to DMP metadevices using an array-specific attribute called the Array Volume ID (AVID). The AVID is a unique identifier for the LUN that is provided by the array. The Array Support Library (ASL) corresponding to the array provides the AVID property. Within an array enclosure, DMP uses the Array Volume Identifier (AVID) as an index in the DMP metanode name. The DMP metanode name is in the format *enclosureID_AVID*.

The SF utilities such as `vxdmpadm getdmpnode` display the DMP metanode name, which includes the AVID property. Use the AVID to correlate the DMP metanode name to the LUN displayed in the array management interface (GUI or CLI) .

If the ASL does not provide the array volume ID property, then DMP generates an index number. DMP sorts the devices seen from an array by the LUN serial number and then assigns the index number. In this case, the DMP metanode name is in the format *enclosureID_index*.

Enclosure based naming with the Array Volume Identifier (AVID) attribute

By default, Dynamic Multi-Pathing (DMP) assigns enclosure-based names to DMP metadevices using an array-specific attribute called the Array Volume ID (AVID). The AVID provides a unique identifier for the LUN that is provided by the array. The ASL corresponding to the array provides the AVID property. Within an array enclosure, DMP uses the Array Volume Identifier (AVID) as an index in the DMP metanode name. The DMP metanode name is in the format *enclosureID_AVID*.

With the introduction of AVID to the enclosure-based naming (EBN) naming scheme, identifying storage devices becomes much easier. The array volume identifier (AVID) enables you to have consistent device naming across multiple nodes connected to the same storage. The disk access name never changes, because it is based on the name defined by the array itself.

Note: DMP does not support AVID with third party drivers.

If DMP does not have access to a device's AVID, it retrieves another unique LUN identifier called the LUN serial number. DMP sorts the devices based on the LUN Serial Number (LSN), and then assigns the index number. All hosts see the same set of devices, so all hosts will have the same sorted list, leading to consistent device indices across the cluster. In this case, the DMP metanode name is in the format *`enclosureID_index`*.

DMP also supports a scalable framework, that allows you to fully customize the device names on a host by applying a device naming file that associates custom names with cabinet and LUN serial numbers.

If a Cluster Volume Manager (CVM) cluster is symmetric, each node in the cluster accesses the same set of disks. Enclosure-based names provide a consistent naming system so that the device names are the same on each node.

The Storage Foundation (SF) utilities such as `vxdisk list` display the DMP metanode name, which includes the AVID property. Use the AVID to correlate the DMP metanode name to the LUN displayed in the array management interface (GUI or CLI) .

For example, on an EMC CX array where the enclosure is emc_clariion0 and the array volume ID provided by the ASL is 91, the DMP metanode name is emc_clariion0_91. The following sample output shows the DMP metanode names:

```
$ vxdisk list
emc_clariion0_91  auto:cdsdisk  emc_clariion0_91   dg1  online shared
emc_clariion0_92  auto:cdsdisk  emc_clariion0_92   dg1  online shared
emc_clariion0_93  auto:cdsdisk  emc_clariion0_93   dg1  online shared
emc_clariion0_282 auto:cdsdisk  emc_clariion0_282  dg1  online shared
emc_clariion0_283 auto:cdsdisk  emc_clariion0_283  dg1  online shared
emc_clariion0_284 auto:cdsdisk  emc_clariion0_284  dg1  online shared
```

```
# vxddladm get namingscheme
NAMING_SCHEME       PERSISTENCE    LOWERCASE      USE_AVID
============================================================
Enclosure Based     Yes            Yes            Yes
```

About disk installation and formatting

Depending on the hardware capabilities of your disks and of your system, you may either need to shut down and power off your system before installing the disks, or you may be able to hot-insert the disks into the live system. Many operating systems can detect the presence of the new disks on being rebooted. If the disks are inserted while the system is live, you may need to enter an operating system-specific command to notify the system.

If the disks require low or intermediate-level formatting before use, use the operating system-specific formatting command to do this.

Note: SCSI disks are usually preformatted. Reformatting is needed only if the existing formatting has become damaged.

See “Adding a disk to VxVM” on page 293.

Adding and removing disks

This section describes managing devices.

Adding a disk to VxVM

Note: Root Disk Encapsulation (RDE) is not supported on Linux from 7.3.1 onwards.

Formatted disks being placed under Veritas Volume Manager (VxVM) control may be new or previously used outside VxVM.

The set of disks can consist of all disks on a controller, selected disks, or a combination of these.

Depending on the circumstances, all of the disks may not be processed in the same way.

For example, some disks may be initialized, while others may be encapsulated to preserve existing data on the disks.

When initializing multiple disks at one time, it is possible to exclude certain disks or certain controllers.

You can also exclude certain disks or certain controllers when encapsulating multiple disks at one time.

To exclude a device from the view of VxVM, select `Prevent multipathing/Suppress devices from VxVM's view` from the `vxdiskadm` main menu.

Warning: Initialization does not preserve the existing data on the disks.

A disk cannot be initialized if it does not have a valid useable partition table. You can use the `fdisk` command to create an empty partition table on a disk as shown here:

```
# fdisk /dev/sdX

   Command (m for help): o
   Command (m for help): w
```

where */dev/sdX* is the name of the disk device, for example, `/dev/sdi`.

Warning: The `fdisk` command can destroy data on the disk. Do not use this command if the disk contains data that you want to preserve.

See "Making devices invisible to VxVM" on page 211.

To initialize disks for VxVM use

1. Select `Add or initialize one or more disks` from the `vxdiskadm` main menu.

2. At the following prompt, enter the disk device name of the disk to be added to VxVM control (or enter `list` for a list of disks):

   ```
   Select disk devices to add:
   [<pattern-list>,all,list,q,?]
   ```

   The *pattern-list* can be a single disk, or a series of disks. If *pattern-list* consists of multiple items, separate them using white space. For example, specify four disks as follows:

   ```
   sde sdf sdg sdh
   ```

   If you enter `list` at the prompt, the `vxdiskadm` program displays a list of the disks available to the system:

   ```
   DEVICE       DISK         GROUP        STATUS
   sdb          mydg01       mydg         online
   sdc          mydg02       mydg         online
   sdd          mydg03       mydg         online
   sde          -            -            online
   sdf          mydg04       mydg         online
   sdg          -            -            online invalid
   ```

   The phrase `online invalid` in the `STATUS` line indicates that a disk has yet to be added or initialized for VxVM control. Disks that are listed as `online` with a disk name and disk group are already under VxVM control.

   Enter the device name or pattern of the disks that you want to initialize at the prompt and press **Return**.

3. To continue with the operation, enter `y` (or press **Return**) at the following prompt:

   ```
   Here are the disks selected. Output format: [Device]
   list of device names

   Continue operation? [y,n,q,?] (default: y) y
   ```

4. At the following prompt, specify the disk group to which the disk should be added, or `none` to reserve the disks for future use:

   ```
   You can choose to add these disks to an existing disk group,
   a new disk group, or you can leave these disks available for use
   by future add or replacement operations. To create a new disk
   group, select a disk group name that does not yet exist. To
   leave the disks available for future use, specify a disk group
   name of none.

   Which disk group [<group>,none,list,q,?]
   ```

5. If you specified the name of a disk group that does not already exist, `vxdiskadm` prompts for confirmation that you really want to create this new disk group:

   ```
   There is no active disk group named disk group name.

   Create a new group named disk group name? [y,n,q,?]
   (default: y)y
   ```

   You are then prompted to confirm whether the disk group should support the Cross-platform Data Sharing (CDS) feature:

   ```
   Create the disk group as a CDS disk group? [y,n,q,?]
   (default: y)
   ```

   If the new disk group may be moved between different operating system platforms, enter `y`. Otherwise, enter `n`.

6. At the following prompt, either press **Return** to accept the default disk name or enter `n` to allow you to define your own disk names:

   ```
   Use default disk names for the disks? [y,n,q,?] (default: y) n
   ```

7. When prompted whether the disks should become hot-relocation spares, enter `n` (or press **Return**):

   ```
   Add disks as spare disks for disk group name? [y,n,q,?]
   (default: n) n
   ```

8. When prompted whether to exclude the disks from hot-relocation use, enter `n` (or press **Return**).

   ```
   Exclude disks from hot-relocation use? [y,n,q,?}
   (default: n) n
   ```

9. You are next prompted to choose whether you want to add a site tag to the disks:

   ```
   Add site tag to disks? [y,n,q,?] (default: n)
   ```

   A site tag is usually applied to disk arrays or enclosures, and is not required unless you want to use the Remote Mirror feature.

   If you enter `y` to choose to add a site tag, you are prompted to the site name at step 11.

10. To continue with the operation, enter `y` (or press **Return**) at the following prompt:

    ```
    The selected disks will be added to the disk group
    disk group name with default disk names.
    list of device names
    Continue with operation? [y,n,q,?] (default: y) y
    ```

11. If you chose to tag the disks with a site in step 9, you are now prompted to enter the site name that should be applied to the disks in each enclosure:

    ```
    The following disk(s):
    list of device names

    belong to enclosure(s):
    list of enclosure names

    Enter site tag for disks on enclosure enclosure name
    [<name>,q,?] site_name
    ```

12 If you see the following prompt, it lists any disks that have already been initialized for use by VxVM:

```
The following disk devices appear to have been initialized
already.
The disks are currently available as replacement disks.
Output format: [Device]

list of device names

Use these devices? [Y,N,S(elect),q,?] (default: Y) Y
```

This prompt allows you to indicate “yes” or “no” for all of these disks (`Y` or `N`) or to select how to process each of these disks on an individual basis (`S`).

If you are sure that you want to reinitialize all of these disks, enter `Y` at the following prompt:

```
VxVM NOTICE V-5-2-366 The following disks you selected for use
appear to already have been initialized for the Volume
Manager. If you are certain the disks already have been
initialized for the Volume Manager, then you do not need to
reinitialize these disk devices.
Output format: [Device]

list of device names

Reinitialize these devices? [Y,N,S(elect),q,?] (default: Y) Y
```

13 `vxdiskadm` may now indicate that one or more disks is a candidate for encapsulation. Encapsulation allows you to add an active disk to VxVM control and preserve the data on that disk.If you want to preserve the data on the disk, enter `y`. If you are sure that there is no data on the disk that you want to preserve, enter `n` to avoid encapsulation.

```
VxVM NOTICE V-5-2-355 The following disk device has a valid
partition table, but does not appear to have been initialized
for the Volume Manager. If there is data on the disk that
should NOT be destroyed you should encapsulate the existing
disk partitions as volumes instead of adding the disk as a new
disk.
Output format: [Device]

device name

Encapsulate this device? [y,n,q,?] (default: y)
```

14. If you choose to encapsulate the disk, `vxdiskadm` confirms its device name and prompts you for permission to proceed. Enter `y` (or press **Return**) to continue encapsulation:

```
VxVM NOTICE V-5-2-311 The following disk device has been
selected for encapsulation.
Output format: [Device]

device name

Continue with encapsulation? [y,n,q,?] (default: y) y
vxdiskadm now displays an encapsulation status and informs you
that you must perform a shutdown and reboot as soon as
possible:
VxVM INFO V-5-2-333 The disk device device name will be
encapsulated and added to the disk group disk group name with the
disk name disk name.
```

You can now choose whether the disk is to be formatted as a CDS disk that is portable between different operating systems, or as a non-portable sliced or simple disk:

```
Enter the desired format [cdsdisk,sliced,simple,q,?]
(default: cdsdisk)
```

Enter the format that is appropriate for your needs. In most cases, this is the default format, `cdsdisk`.

At the following prompt, `vxdiskadm` asks if you want to use the default private region size of 65536 blocks (32MB). Press **Return** to confirm that you want to use the default value, or enter a different value. (The maximum value that you can specify is 524288 blocks.)

```
Enter desired private region length [<privlen>,q,?]
(default: 65536)
```

If you entered `cdsdisk` as the format, you are prompted for the action to be taken if the disk cannot be converted to this format:

```
 Do you want to use sliced as the format should cdsdisk fail?
[y,n,q,?] (default: y)
```

If you enter `y`, and it is not possible to encapsulate the disk as a CDS disk, it is encapsulated as a sliced disk. Otherwise, the encapsulation fails.

`vxdiskadm` then proceeds to encapsulate the disks. You should now reboot your system at the earliest possible opportunity, for example by running this command:

```
# shutdown -r now
```

The `/etc/fstab` file is updated to include the volume devices that are used to mount any encapsulated file systems. You may need to update any other references in backup scripts, databases, or manually created swap devices. The original `/etc/fstab` file is saved as `/etc/fstab.b4vxvm`.

15 If you choose not to encapsulate the disk, `vxdiskadm` asks if you want to initialize the disk instead. Enter `y` to confirm this:

Instead of encapsulating, initialize? [y,n,q,?] (default: n) **y**`vxdiskadm` now confirms those disks that are being initialized and added to VxVM control with messages similar to the following. In addition, you may be prompted to perform surface analysis.

```
VxVM INFO V-5-2-205 Initializing device device name.
```

16 You can now choose whether the disk is to be formatted as a CDS disk that is portable between different operating systems, or as a non-portable sliced or simple disk:

```
Enter the desired format [cdsdisk,sliced,simple,q,?]
(default: cdsdisk)
```

Enter the format that is appropriate for your needs. In most cases, this is the default format, `cdsdisk`.

17 At the following prompt, `vxdiskadm` asks if you want to use the default private region size of 65536 blocks (32MB). Press **Return** to confirm that you want to use the default value, or enter a different value. (The maximum value that you can specify is 524288 blocks.)

```
Enter desired private region length [<privlen>,q,?]
(default: 65536)
```

`vxdiskadm` then proceeds to add the disks.

```
VxVM INFO V-5-2-88 Adding disk device device name to disk group
disk group name with disk name disk name.
 .
 .
 .
```

18 If you choose not to use the default disk names, `vxdiskadm` prompts you to enter the disk name.

19 At the following prompt, indicate whether you want to continue to initialize more disks (`y`) or return to the `vxdiskadm` main menu (`n`):

```
Add or initialize other disks? [y,n,q,?] (default: n)
```

You can change the default layout for disks using the `vxdisk` command or the `vxdiskadm` utility.

See the `vxdisk`(1M) manual page.

See the `vxdiskadm`(1M) manual page.

Disk reinitialization

You can reinitialize a disk that has previously been initialized for use by Veritas Volume Manager (VxVM) by putting it under VxVM control as you would a new disk.

See "Adding a disk to VxVM" on page 293.

Warning: Reinitialization does not preserve data on the disk. If you want to reinitialize the disk, make sure that it does not contain data that should be preserved.

If the disk you want to add has been used before, but not with a volume manager, you can encapsulate the disk to preserve its information. If the disk you want to add has previously been under LVM control, you can preserve the data it contains on a VxVM disk by the process of conversion.

For detailed information about migrating volumes, see the *Veritas InfoScale Solutions Guide*.

Using vxdiskadd to put a disk under VxVM control

You can add a disk to Veritas Volume Manager (VxVM) control with the `vxdiskadd` command.

To use the vxdiskadd command to put a disk under VxVM control.

- Type the following command:

  ```
  # vxdiskadd disk
  ```

  For example, to initialize the disk `sdb`:

  ```
  # vxdiskadd sdb
  ```

  The `vxdiskadd` command examines your disk to determine whether it has been initialized and also checks for disks that have been added to VxVM, and for other conditions.

  The `vxdiskadd` command also checks for disks that can be encapsulated.

  See "Encapsulating a disk" on page 730.

  If you are adding an uninitialized disk, the `vxdiskadd` command displays warning and error messages on the console. Ignore these messages. These messages should not appear after the disk has been fully initialized; the `vxdiskadd` command displays a success message when the initialization completes.

  The interactive dialog for adding a disk using `vxdiskadd` is similar to that for `vxdiskadm`.

  See "Adding a disk to VxVM" on page 293.

Removing disks

This section describes how to remove a Veritas Volume Manager (VxVM) disk.

You must disable a disk group before you can remove the last disk in that group.

See "Disabling a disk group" on page 710.

As an alternative to disabling the disk group, you can destroy the disk group.

See "Destroying a disk group" on page 710.

You can remove a disk from a system and move it to another system if the disk is failing or has failed.

To remove a disk

1. Stop all activity by applications to volumes that are configured on the disk that is to be removed. Unmount file systems and shut down databases that are configured on the volumes.
2. Use the following command to stop the volumes:

   ```
   # vxvol [-g diskgroup] stop vol1 vol2 ...
   ```

3. Move the volumes to other disks or back up the volumes. To move a volume, use `vxdiskadm` to mirror the volume on one or more disks, then remove the original copy of the volume. If the volumes are no longer needed, they can be removed instead of moved.
4. Check that any data on the disk has either been moved to other disks or is no longer needed.
5. Select `Remove a disk` from the `vxdiskadm` main menu.
6. At the following prompt, enter the disk name of the disk to be removed:

   ```
   Enter disk name [<disk>,list,q,?] mydg01
   ```

7. If there are any volumes on the disk, VxVM asks you whether they should be evacuated from the disk. If you wish to keep the volumes, answer `y`. Otherwise, answer `n`.
8. At the following verification prompt, press **Return** to continue:

   ```
   VxVM NOTICE V-5-2-284 Requested operation is to remove disk
   mydg01 from group mydg.

   Continue with operation? [y,n,q,?] (default: y)
   ```

   The `vxdiskadm` utility removes the disk from the disk group and displays the following success message:

   ```
   VxVM INFO V-5-2-268 Removal of disk mydg01 is complete.
   ```

   You can now remove the disk or leave it on your system as a replacement.
9. At the following prompt, indicate whether you want to remove other disks (`y`) or return to the `vxdiskadm` main menu (`n`):

   ```
   Remove another disk? [y,n,q,?] (default: n)
   ```

Removing a disk with subdisks

You can remove a Veritas Volume Manager (VxVM) disk on which some subdisks are defined. For example, you can consolidate all the volumes onto one disk. If you use the `vxdiskadm` program to remove a disk, you can choose to move volumes off that disk.

Some subdisks are not movable. A subdisk may not be movable for one of the following reasons:

- There is not enough space on the remaining disks in the subdisks disk group.
- Plexes or striped subdisks cannot be allocated on different disks from existing plexes or striped subdisks in the volume.

If the `vxdiskadm` program cannot move some subdisks, remove some plexes from some disks to free more space before proceeding with the disk removal operation.

See “Removing a volume” on page 719.

To remove a disk with subdisks

1. Run the `vxdiskadm` program and select `Remove a disk` from the main menu.

   If the disk is used by some subdisks, the following message is displayed:

   ```
   VxVM ERROR V-5-2-369 The following volumes currently use part of
   disk mydg02:

   home usrvol

   Volumes must be moved from mydg02 before it can be removed.

   Move volumes to other disks? [y,n,q,?] (default: n)
   ```

2. Choose `y` to move all subdisks off the disk, if possible.

Removing a disk with no subdisks

You can remove a Veritas Volume Manager (VxVM) disk that contains no subdisks.

To remove a disk that contains no subdisks from its disk group

- Run the `vxdiskadm` program and select `Remove a disk` from the main menu, and respond to the prompts as shown in this example to remove `mydg02`:

  ```
  Enter disk name [<disk>,list,q,?] mydg02

  VxVM NOTICE V-5-2-284 Requested operation is to remove disk
  mydg02 from group mydg.

  Continue with operation? [y,n,q,?] (default: y) y
  VxVM INFO V-5-2-268 Removal of disk mydg02 is complete.
  Clobber disk headers? [y,n,q,?] (default: n) y
  ```

  Enter `y` to remove the disk completely from VxVM control. If you do not want to remove the disk completely from VxVM control, enter `n`.

Renaming a disk

Veritas Volume Manager (VxVM) gives the disk a default name when you add the disk to VxVM control, unless you specify a VxVM disk name. VxVM uses the VxVM disk name to identify the location of the disk or the disk type.

To rename a disk

- Type the following command:

  ```
  # vxedit [-g diskgroup] rename old_diskname new_diskname
  ```

  By default, VxVM names subdisk objects after the VxVM disk on which they are located. Renaming a VxVM disk does not automatically rename the subdisks on that disk.

  For example, you might want to rename disk `mydg03`, as shown in the following output from `vxdisk list`, to `mydg02`:

  ```
  # vxdisk list

  DEVICE       TYPE            DISK         GROUP        STATUS
  sdb          auto:sliced     mydg01       mydg         online
  sdc          auto:sliced     mydg03       mydg         online
  sdd          auto:sliced     -            -            online
  ```

  You would use the following command to rename the disk.

  ```
  # vxedit -g mydg rename mydg03 mydg02
  ```

  To confirm that the name change took place, use the `vxdisk list` command again:

  ```
  # vxdisk list

  DEVICE       TYPE            DISK         GROUP        STATUS
  sdb          auto:sliced     mydg01       mydg         online
  sdc          auto:sliced     mydg02       mydg         online
  sdd          auto:sliced     -            -            online
  ```

Event monitoring

This chapter includes the following topics:

About the Dynamic Multi-Pathing (DMP) event source daemon (vxesd)

The event source daemon (`vxesd`) is a Dynamic Multi-Pathing (DMP) component process that receives notifications of any device-related events that are used to take appropriate actions. The benefits of `vxesd` include:

- Monitoring of SAN fabric events and proactive error detection (SAN event)
  See "Fabric Monitoring and proactive error detection" on page 309.
- Logging of DMP events for troubleshooting (DMP event)
  See "DMP event logging" on page 310.
- Automated device discovery (OS event)
- Discovery of SAN components and HBA-array port connectivity (Fibre Channel and iSCSI)
  See "Dynamic Multi-Pathing (DMP) discovery of iSCSI and SAN Fibre Channel topology" on page 310.

See “Starting and stopping the Dynamic Multi-Pathing (DMP) event source daemon” on page 311.

Fabric Monitoring and proactive error detection

DMP takes a proactive role in detecting errors on paths.

The DMP event source daemon `vxesd` uses the Storage Networking Industry Association (SNIA) HBA API library to receive SAN fabric events from the HBA.

DMP checks devices that are suspect based on the information from the SAN events, even if there is no active I/O. New I/O is directed to healthy paths while DMP verifies the suspect devices.

During startup, `vxesd` queries the HBA (by way of the SNIA library) to obtain the SAN topology. The `vxesd` daemon determines the Port World Wide Names (PWWN) that correspond to each of the device paths that are visible to the operating system. After the `vxesd` daemon obtains the topology, `vxesd` registers with the HBA for SAN event notification. If LUNs are disconnected from a SAN, the HBA notifies `vxesd` of the SAN event, specifying the PWWNs that are affected. The `vxesd` daemon uses this event information and correlates it with the previous topology information to determine which set of device paths have been affected.

The `vxesd` daemon sends the affected set to the `vxconfigd` daemon (DDL) so that the device paths can be marked as suspect.

When the path is marked as suspect, DMP does not send new I/O to the path unless it is the last path to the device. In the background, the DMP restore task checks the accessibility of the paths on its next periodic cycle using a SCSI inquiry probe. If the SCSI inquiry fails, DMP disables the path to the affected LUNs, which is also logged in the event log.

If the LUNs are reconnected at a later time, the HBA informs `vxesd` of the SAN event. When the DMP restore task runs its next test cycle, the disabled paths are checked with the SCSI probe and re-enabled if successful.

Note: If `vxesd` receives an HBA LINK UP event, the DMP restore task is restarted and the SCSI probes run immediately, without waiting for the next periodic cycle. When the DMP restore task is restarted, it starts a new periodic cycle. If the disabled paths are not accessible by the time of the first SCSI probe, they are re-tested on the next cycle (300s by default).

The fabric monitor functionality is enabled by default. The value of the `dmp_monitor_fabric` tunable is persistent across restarts.

To display the current value of the dmp_monitor_fabric tunable, use the following command:

```
# vxdmpadm gettune dmp_monitor_fabric
```

To disable the Fabric Monitoring functionality, use the following command:

```
# vxdmpadm settune dmp_monitor_fabric=off
```

To enable the Fabric Monitoring functionality, use the following command:

```
# vxdmpadm settune dmp_monitor_fabric=on
```

Dynamic Multi-Pathing (DMP) discovery of iSCSI and SAN Fibre Channel topology

The vxesd builds a topology of iSCSI and Fibre Channel (FC) devices that are visible to the host. The vxesd daemon uses the SNIA Fibre Channel HBA API to obtain the SAN topology. If IMA is not available, then the iSCSI management CLI is used to obtain the iSCSI SAN topology.

To display the hierarchical listing of Fibre Channel and iSCSI devices, use the following command:

```
# vxddladm list
```

See the vxddladm(1M) manual page.

DMP event logging

See "About the Dynamic Multi-Pathing (DMP) event source daemon (vxesd)" on page 308.

The event source daemon (vxesd) is a Dynamic Multi-Pathing (DMP) component process that receives notifications of any device-related events that are used to take appropriate actions.

DMP notifies vxesd of major events, and vxesd logs the event in a log file. These events include:

- Marking paths or dmpnodes enabled
- Marking paths or dmpnodes disabled
- Throttling of paths
- I/O error analysis

- HBA and SAN events

You can change the level of detail that is displayed in the system or console log about the DMP events. Use the tunable `dmp_log_level`. Valid values are 1 through 9. The default level is 1.

```
# vxdmpadm settune dmp_log_level=X
```

The current value of `dmp_log_level` can be displayed with:

```
# vxdmpadm gettune dmp_log_level
```

For details on the various log levels, see the `vxdmpadm`(1M) manual page.

Starting and stopping the Dynamic Multi-Pathing (DMP) event source daemon

By default, Dynamic Multi-Pathing (DMP) starts the event source daemon, `vxesd`, at boot time.

To stop the `vxesd` daemon, use the `vxddladm` utility:

```
# vxddladm stop eventsource
```

To start the `vxesd` daemon, use the `vxddladm` utility:

```
# vxddladm start eventsource [logfile=logfilename]
```

To view the status of the `vxesd` daemon, use the `vxddladm` utility:

```
# vxddladm status eventsource
```

Section 4

Administering Storage Foundation

Chapter 13

Administering sites and remote mirrors

This chapter includes the following topics:

About sites and remote mirrors

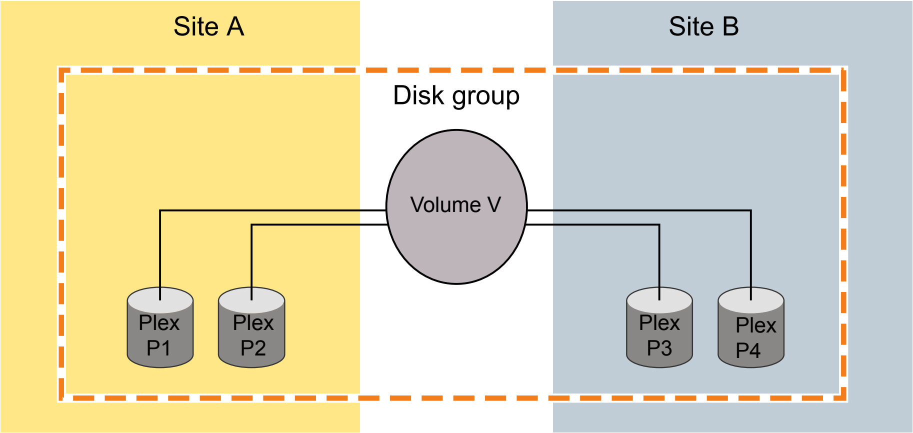
In a Remote Mirror configuration (also known as a campus cluster or stretch cluster) the hosts and storage that would usually be located in one place, are instead divided between two or more sites.

For applications and services to function correctly at a site when other sites have become inaccessible, at least one complete plex of each volume must be configured at each site (site-based allocation), and the consistency of the data in the plexes at each site must be ensured (site consistency).

By tagging disks with site names, storage can be allocated from the correct location when creating, resizing or relocating a volume, and when changing a volume's layout.

Figure 13-1 shows an example of a site-consistent volume with two plexes configured at each of two sites.

Figure 13-1 Site-consistent volume with two plexes at each of two sites



The allocated storage for plexes P1 and P2 is tagged with site A and the allocated storage for plexes P3 and P4 is tagged with site B.

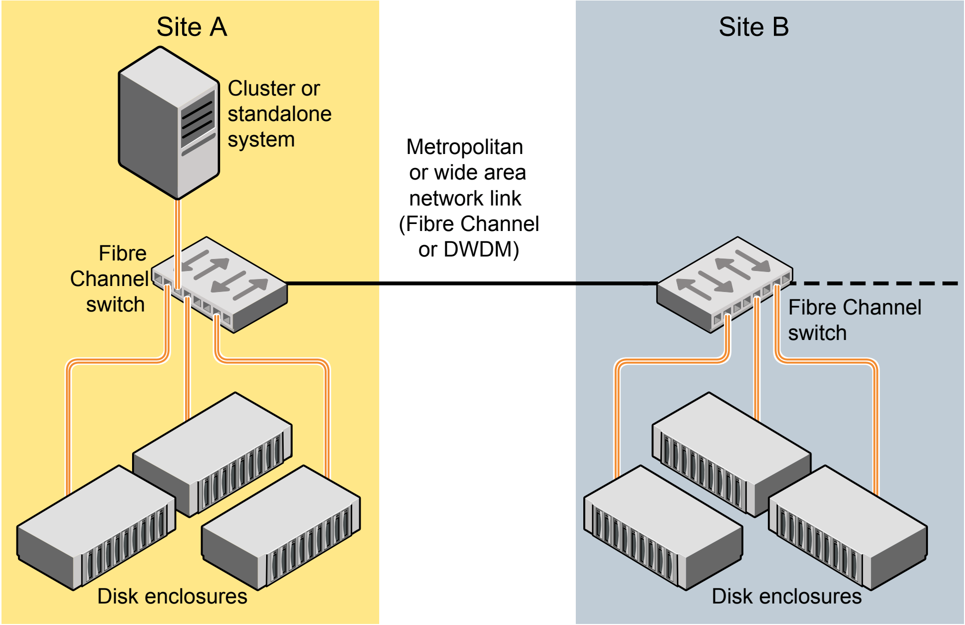
Although not shown in this figure, DCO log volumes are also mirrored across the sites, and disk group configuration copies are distributed across the sites.

Site consistency means that the data in the plexes for a volume must be consistent at each site. The site consistency of a volume is ensured by detaching a site when its last complete plex fails at that site. If a site fails, all its plexes are detached and the site is said to be detached. If site consistency is not on, only the plex that fails is detached. The remaining volumes and their plexes on that site are not detached.

To enhance read performance, VxVM will service reads from the plexes at the local site where an application is running if the `siteread` read policy is set on a volume. Writes are written to plexes at all sites.

Figure 13-2 shows a configuration with remote storage only (that is also supported).

Figure 13-2 Example of a two-site configuration with remote storage only


Site A
Cluster or standalone system
Fibre Channel switch
Disk enclosures
Metropolitan or wide area network link (Fibre Channel or DWDM)
Site B
Fibre Channel switch
Disk enclosures

About site-based allocation

Site-based allocation policies are enforced by default in a site-configured disk group. Site-based allocation requires that each volume has at least one plex at each site that is configured in the disk group. When a new volume is created in a site-configured disk group, the `allsites` attribute is set to `on` by default. The `allsites` attribute indicates that the volume must have at least one plex on each configured site in the disk group. For new volumes, the read policy is set to `siteread` by default.

If mirroring across sites is not required, or is not possible (as is the case for RAID-5 volumes), specify the `allsites=off` attribute to the `vxassist` command. If sites are configured in the disk group, a plex will always be confined to a site and will not span across sites. This enforcement cannot be overridden.

Before adding a new site to a disk group, be sure to meet the following requirements:

- Disks from the site being added (site tagged) are present or added to the disk group.
- Each existing volume with `allsites` set in the disk group must have at least one plex at the site being added. If this condition is not met, the command to

add the site to the disk group fails. If the `-f` option is specified, the command does not fail, but instead it sets the `allsites` attribute for the volume to `off`.

About site consistency

Site consistency means that at any point in time, the data at each site is consistent with the application for a given set of volumes. A site-consistent volume must have at least one plex, or mirror, on each configured site in the disk group. The site consistency is ensured by detaching a site when a site-consistent volume loses its last complete plex on that site. The site detach detaches all the plexes on that site and also disallows further configuration updates to the configuration copies on that site. Turn on this behavior by setting the `siteconsistent` attribute to `on` on the desired volumes.

If you set the `siteconsistent` attribute to `off`, only the plex that fails is detached. The plexes for the remaining volumes on that site are not detached.

The `siteconsistent` attribute is also present at the disk group level and can be used to turn on or off the site consistency functionality in the disk group boundary. In addition, if you turn on the `siteconsistent` attribute for a disk group, each new volume created in the disk group inherits the `siteconsistent` attribute of the disk group, by default. Setting the `siteconsistent` attribute on a disk group does not affect `siteconsistent` attributes for existing volumes. You can also control the site consistency on individual volumes.

By default, a volume inherits the value that is set on its disk group.

By default, creating a site-consistent volume also creates an associated version 20 DCO volume, and enables Persistent FastResync on the volume. This allows faster recovery of the volume during the reattachment of a site.

See “Configuring site consistency on a volume” on page 324.

Before setting site consistency on a disk group, be sure to meet the following requirements:

- A license enabling the Site Awareness feature must be installed on all the hosts in the Remote Mirror configuration.
- At least two sites must be configured in the disk group before site consistency is turned on.
  See “Making an existing disk group site consistent” on page 318.
- All the disks in a disk group must be registered to one of the sites before you can set the `siteconsistent` attribute on the disk group.

About site tags

In a Remote Mirror configuration, each storage device in the disk group must be tagged with site information. The site tag indicates to which site the device is associated. VxVM provides a facility to tag VxVM-initialized disks with an arbitrary name-value pair. The tag name `site` is reserved by VxVM and is used to identify the site information of tagged disks. The command `vxdisk settag` can be used to tag multiple disks or all disks from an enclosure or disks from multiple enclosures. The tagging can be restricted to disks in a disk group by specifying the disk group with the command.

You can use automatic site tagging to assign site tags to disks when adding them to a disk group. When automatic site tagging is on, newly added disks or LUNs inherit the site tag from the site-enclosure mapping stored in the disk group. To use automatic site tagging, turn on automatic site tagging for a disk group, and then assign the site names to the enclosures in the disk group. Any disks or LUNs in that disk group inherit the tag from the enclosure to which they belong.

About the site read policy

To enhance read performance, VxVM will service reads from the plexes at the local site where an application is running, if the `siteread` read policy is set on a volume. `siteread` is the default read policy for volumes that have sites configured. Writes are written to plexes at all sites. By tagging hosts with site information, VxVM identifies which hosts belong to which site. Reads initiated by a host from one site are then satisfied by disks which are tagged with the same site. Tagging hosts and disks with correct site information gives you maximum read performance when `siteread` read policy is used.

If a license enabling the Site Awareness feature is installed on all the hosts in the Remote Mirror configuration, the disk group is configured for site consistency with several sites enabled, and the `allsites=on` attribute is specified for a volume, the default read policy is `siteread`.

If the `siteread` policy is not set, use the following command to set the read policy to `siteread` policy on a volume:

```
# vxvol [-g diskgroup] rdpol siteread volume
```

This command has no effect if a site name has not been set for the host.

See “Changing the read policy for mirrored volumes” on page 159.

Making an existing disk group site consistent

The site consistency feature requires that a license enabling the site awareness feature has been installed on all hosts at all sites that participate in the configuration.

To make an existing disk group site consistent

1 Ensure that the disk group is updated to at least version 140, by running the `vxdg upgrade` command on it:

```
# vxdg upgrade diskgroup
```

2 On each host that can access the disk group, define the site name:

```
# vxdctl set site=sitename
```

3 Tag all the disks in the disk group with the appropriate site name:

```
# vxdisk [-g diskgroup] settag site=sitename disk1 disk2
```

Or, to tag all the disks in a specified enclosure, use the following command:

```
# vxdisk [-g diskgroup] settag site=sitename encl:encl_name
```

4 Use the `vxdg move` command to move any unsupported RAID-5 volumes to another disk group. Alternatively, use the `vxassist convert` commands to convert the volumes to a supported layout such as `mirror` or `mirror-stripe`. You can use the `site` and `mirror=site` storage allocation attribute to ensure that the plexes are created on the correct storage.

5 Use the `vxevac` command to ensure that the volumes have at least one plex at each site. You can use the `site` and `mirror=site` storage allocation attribute to ensure that the plexes are created on the correct storage.

6 Register a site record for each site with the disk group:

```
# vxdg -g diskgroup addsite sitename
```

7 Turn on site consistency for the disk group:

```
# vxdg -g diskgroup set siteconsistent=on
```

8 Turn on the `allsites` flag for the volume which requires data replication to each site:

```
# vxvol [-g diskgroup] set allsites=on volume
```

9 Turn on site consistency for each existing volume in the disk group for which site consistency is needed. You also need to attach `DCOv30` or higher if it is not attached already. `DCOv30` or higher is required to ensure that site detach and reattach are instantaneous.

```
# vxvol [-g diskgroup] set siteconsistent=on volume ...
```

Configuring a new disk group as a Remote Mirror configuration

Note: The Remote Mirror feature requires that a license enabling the Site Awareness feature has been installed on all hosts at all sites that participate in the configuration.

This section describes setting up a new disk group. To configure an existing disk group as a Remote Mirror configuration, additional steps may be required.

See "Making an existing disk group site consistent" on page 318.

Setting up a new disk group for a Remote Mirror configuration

1 Define the site name for each host that can access the disk group.

```
# vxdctl set site=sitename
```

To verify the site name assigned to the host, use the following command:

```
# vxdctl list
```

2 Create the disk group with storage from each site.

3 Register a site record to the disk group, for each site.

```
# vxdg -g diskgroup [-f] addsite sitename
```

4 Do one of the following:

- To tag all disks regardless of the disk group, do the following:
  Assign a site name to the disks or enclosures. You can set site tags at the disk level, or at the enclosure level. If you specify one or more enclosures,

the site tag applies to the disks in that enclosure that are within the disk group. Enter the following command:

```
# vxdisk [-g diskgroup] settag site=sitename \
  disk disk1... |encl:encl_name encl:encl_name1...
```

where the disks can be specified either by the disk access name or the disk media name.

- To autotag new disks added to the disk group based on the enclosure to which they belong, perform the following steps in the order presented. These steps are limited to disks in a single group.
    - Set the `autotagging` policy to `on` for the disk group, if required.

      Automatic tagging is the default setting, so this step is only required if the autotagging policy was previously disabled. To turn on autotagging, enter the following command:

      ```
      # vxdg [-g diskgroup] set autotagging=on
      ```

    - Add site-enclosure mapping information to the diskgroup for each site-enclosure combination. Enter the following command:

      ```
      # vxdg [-g diskgroup] settag encl:encl_name1 site=sitename1
      ```

      As a result of this command, all disks of enclosure `encl_name1` in the specified disk group are tagged with site information.

5 Turn on the site consistency requirement for a disk group:

```
# vxdg -g diskgroup set siteconsistent=on
```

Fire drill — testing the configuration

Warning: To avoid potential loss of service or data, it is recommended that you do not use these procedures on a live system.

After validating the consistency of the volumes and disk groups at your sites, you should validate the procedures that you will use in the event of the various possible types of failure. A fire drill lets you test that a site can be brought up cleanly during recovery from a disaster scenario such as site failure.

Simulating site failure

To simulate the failure of a site, use the following command to detach all the devices at a specified site:

```
# vxdg -g diskgroup [-f] detachsite sitename
```

The `-f` option must be specified if any plexes configured on storage at the site are currently online.

After the site is detached, the application should run correctly on the available site. This step verifies that the primary site is fine. Continue the fire drill by verifying the secondary site.

Verifying the secondary site

After detaching the site from primary site, verify whether the application starts correctly on a secondary site. The fire drill ensures that the application can run on the secondary if disaster strikes the primary site. These procedures assume that the application is running correctly before the fire drill operation begins.

To verify the secondary site, import the detached site on a different host using the following command:

```
# vxdg -o site=sitename import dgname
```

Then start the application. If the application runs correctly on the secondary site, this step verifies the integrity of the secondary site.

Recovery from simulated site failure

After verifying the data on the secondary for a simulated site failure, deport the disk group from the secondary site. Then reattach the site back to the primary host.

Use the following commands to reattach a site and recover the disk group:

```
# vxdg -g diskgroup [-o overridessb] reattachsite sitename
# vxrecover -g diskgroup
```

It may be necessary to specify the `-o overridessb` option if a serial split-brain condition is indicated.

Changing the site name

You can change the site name, or tag, that is used to identify each site in a Remote Mirror configuration. Renaming the site changes the site record in the disk group.

The site name is also changed for all of the disks and enclosures that are tagged with the existing site name.

After you rename a site, you need to explicitly change the site name for each host that belongs to that site.

See "Resetting the site name for a host" on page 322.

To rename the site

- Specify the new site name as follows:

  ```
  # vxdg [-g diskgroup] renamesite old_sitename new_sitename
  ```

Resetting the site name for a host

If you rename a site, you need to explicitly set each host to refer to the new site name.

To reset a site name for a host

1. Remove the site name from a host:

   ```
   # vxdctl [-F] unset site
   ```

   The `-F` option is required if any imported disk groups are registered to the site.

2. Set the new site name for the host.

   ```
   # vxdctl set site=sitename
   ```

   The name that has been assigned to a site is stored in the `/etc/vx/volboot` file.

Administering the Remote Mirror configuration

After the Remote Mirror site is configured, refer to the following sections for additional tasks to maintain the configuration.

Configuring site tagging for disks or enclosures

To set up a Remote Mirror configuration, specify to which site each storage device in the disk group belongs. Assign a site tag to one or more disks or enclosures. If the disk or enclosure does not belong to a disk group, you must use this method to assign a site tag.

To tag disks or enclosures with a site name

- Assign a site name to one or more disks or enclosures, using the following command:

  ```
  # vxdisk [-g diskgroup] settag site=sitename \
    disk disk1...|encl:encl_name encl:encl_name1...
  ```

  where the disks can be specified either by the disk access name or the disk media name.

To display the disks or enclosures registered to a site

- To check which disks or enclosures are registered to a site, use the following command:

  ```
  # vxdisk [-g diskgroup] listtag
  ```

To remove the site tag from a disk or enclosure

- To remove the site tag from a disk or enclosure, use the following command:

  ```
  # vxdisk rmtag site=sitename \
  disk disk1...|encl:encl_name encl:encl_name1...
  ```

Configuring automatic site tagging for a disk group

Configure automatic site tagging if you want disks or LUNs to inherit the tag from the enclosure. After you turn on automatic site tagging for a disk group, assign the site names to the enclosures in the disk group. Any disks or LUNs added to that disk group inherit the tag from the enclosure to which they belong.

To configure automatic site tagging for a disk group

1. Set the autotagging policy to **on** for the disk group. Automatic tagging is the default setting, so this step is only required if the autotagging policy was previously disabled.

   To turn on autotagging, use the following command:

   ```
   # vxdg [-g diskgroup] set autotagging=on
   ```

2. Assign the site name to an enclosure within the disk group, using the following command:

   ```
   # vxdg [-g diskgroup] settag encl:encl_name site=sitename
   ```

To list the site tags for a disk group

- To list the site tags for a disk group, use the following command:

  ```
  # vxdg  [-q] [-o tag=name|~name[=value|~value] \
  listtag [diskgroup ...]
  ```

To remove a site tag from an enclosure or a disk group

- To remove a site tag from a disk group, use the following command:

  ```
  # vxdg [-g diskgroup] rmtag [encl:encl_name] site=sitename
  ```

Configuring site consistency on a volume

To set the site consistency requirement when creating a volume, specify the `siteconsistent` attribute to the `vxassist make` command, for example:

```
# vxassist [-g diskgroup] make volume size \
  nmirror=4 siteconsistent={on|off}
```

By default, a volume inherits the value that is set on its disk group.

By default, creating a site-consistent volume also creates an associated version 30 DCO volume, and enables Persistent FastResync on the volume. This allows faster recovery of the volume during the reattachment of a site.

To turn on the site consistency requirement for an existing volume, use the following form of the `vxvol` command:

```
# vxvol [-g diskgroup] set siteconsistent=on volume
```

To turn off the site consistency requirement for a volume, use the following command:

```
# vxvol [-g diskgroup] set siteconsistent=off volume
```

The `siteconsistent` attribute and the `allsites` attribute must be set to `off` for RAID-5 volumes in a site-consistent disk group.

Examples of storage allocation by specifying sites

Table 13-1 shows examples of how to use sites with the `vxassist` command to allocate storage. These examples assume that the disk group, `ccdg`, has been enabled for site consistency with disks configured at two sites, `site1` and `site2`.

Also, ccdg01, ccdg02, and ccdg03 are dm names of disks tagged with site site1. ccdg09, ccdg10, and ccdg11 are dm names of disks tagged with site site2.

Table 13-1 Examples of storage allocation by specifying sites

Command	Description

# vxassist -g ccdg make vol 2g \ 	
nmirror=2
	Create a volume with one mirror at each site. The nmirror keyword is optional. If the nmirror keyword is specified, it must equal the number of sites.

# vxassist -g ccdg -o ordered \ make vol 2g \ layout=mirror-stripe ncol=3 \ ccdg01 ccdg02 ccdg03 ccdg09 \ ccdg10 ccdg11	Create a mirrored-stripe volume specifying allocation order to validate redundancy across the sites. The named disks must be tagged with the appropriate site name, and there must be sufficient disks at each site to create the volume.

# vxassist -g ccdg make vol 2g \ 	
nmirror=2 ccdg01 ccdg09
	Create a volume with one mirror on each of the named disks. The named disks must be tagged with the appropriate site name, and there must be sufficient disks at each site to create the volume.

# vxassist -g ccdg make vol 2g \ 	
nmirror=2 siteconsistent=off \
allsites=off
	Create a mirrored volume that is not site consistent. Both mirrors can be allocated from any available storage in the disk group, but the storage for each mirror is confined to a single site.

# vxassist -g ccdg make vol 2g \ 	
nmirror=2 site:site2 \
siteconsistent=off \
allsites=off
	Create a mirrored volume that is not site consistent. Both mirrors are allocated from any available storage in the disk group that is tagged as belonging to site2.

Table 13-1 Examples of storage allocation by specifying sites *(continued)*

Command	Description

`# vxassist -g ccdg make vol 2g \`
`nmirror=2 \!site:site1 \`
`siteconsistent=off \`
`allsites=off`
	Create a mirrored volume that is not site consistent. Both mirrors are allocated from any available storage in the disk group that is tagged as not belonging to `site1`.
	Note: The ! character is a special character in some shells. This example shows how to escape it in a bash shell.

`# vxassist -g ccdg mirror vol \`
`site:site1`
	Add a mirror at a specified site. The command fails if there is insufficient storage available at the site. This command does not affect the `allsites` or `siteconsistent` of a volume.

`# vxassist -g ccdg remove \`
`mirror vol site:site1`
	Remove a mirror from a volume at a specified site. If the volume has the `allsites` attribute set to `on`, the command fails if this would remove the last remaining plex at a site.

`# vxassist -g ccdg growto vol \`
`4g`
	Grow a volume. Each mirror of a volume is grown using the same site storage to which it belongs. If there is not enough storage to grow a mirror on each site, the command fails.

Displaying site information

To display the site name for a host

- To determine to which site a host belongs, use the following command on the host:

  ```
  # vxdctl list | grep siteid
  siteid: building1
  ```

To display the disks or enclosures registered to a site

- To check which disks or enclosures are registered to a site, use the following command:

```
# vxdisk [-g diskgroup] listtag
```

To display the setting for automatic site tagging for a disk group

- To determine whether automatic site tagging is on for a disk group, use the following command:

```
# vxprint -g diskgroup -F"%autotagging" diskgroup
```

To verify whether site consistency has been enabled for a disk group

- To verify whether site consistency has been enabled for a disk group, use the following command:

```
# vxdg list diskgroup | grep siteconsistent
flags: siteconsistent
```

To verify whether site consistency has been enabled for a volume

- To verify whether site consistency has been enabled for a volume, use the following command:

```
# vxprint -g diskgroup -F"%siteconsistent" vol
```

To identify which site a plex or mirror is allocated from

- To identify which site a plex or mirror is allocated from, use the following command:

```
# vxprint -g diskgroup -F"%site" plex
```

To list the site tags for a disk group

- To list the site tags for a disk group, use the following command:

```
# vxdg  [-q] [-o tag=name|~name[=value|~value] \
listtag [diskgroup ...]
```

Failure and recovery scenarios

Table 13-2 lists the possible failure scenarios and recovery procedures for the Remote Mirror feature.

Table 13-2 Failure scenarios and recovery procedures

Failure scenario	Recovery procedure
Disruption of network link between sites.	See "Recovering from a loss of site connectivity" on page 328.
Failure of storage at a site.	See "Recovering from storage failure" on page 328.
Failure of both hosts and storage at a site.	See "Recovering from site failure" on page 329.
Disruption of connectivity to storage at all sites for hosts at a site	See "Recovering from disruption to connectivity to storage at all sites from the hosts at a site" on page 329.

Recovering from a loss of site connectivity

At the chosen site, use the following commands to reattach a site and recover the disk group:

```
# vxdg -g diskgroup -o overridessb reattachsite sitename
# vxrecover -g diskgroup
```

In the case that the host systems are configured at a single site with only storage at the remote sites, the usual resynchronization mechanism of VxVM is used to recover the remote plexes when the storage comes back on line.

See "Handling conflicting configuration copies" on page 703.

Recovering from storage failure

If storage fails at a site, the plexes that are configured on that storage are detached locally if a site-consistent volume still has other mirrors available at the site. The hot-relocation feature of VxVM will attempt to recreate the failed plexes on other available storage in the disk group. If no plexes of a site-consistent volume remain in operation at a site, and hot-relocation cannot recreate the plexes at that site, the site is detached. Because site connectivity has not been lost, applications running on hosts at the site can still access data at the other sites.

When the storage comes back online, the `vxattachd` reattaches the site automatically.

See "Automatic site reattachment" on page 329.

If the `vxattachd` is not running, use the following commands to reattach a site and recover the disk group:

```
# vxdg -g diskgroup reattachsite sitename
# vxrecover -g diskgroup
```

For more information about recovering a disk group, refer to the *Veritas InfoScale Troubleshooting Guide*.

Recovering from site failure

If all the hosts and storage fail at a site, use the following commands to reattach the site after it comes back online, and to recover the disk group:

```
# vxdg -g diskgroup reattachsite sitename
# vxrecover -g diskgroup
```

Recovering from disruption to connectivity to storage at all sites from the hosts at a site

In this scenario, hosts at a site lose connectivity to the storage at all sites. For example, hosts in Site A lose connectivity to the storage at both Site A and Site B.

In this case, no site is detached. I/O fails on the hosts that lose connectivity to the storage at all sites. The application should be failed over to hosts on the other site.

Restore the connectivity to the storage, and then bring the applications online.

Automatic site reattachment

The automatic site reattachment daemon, `vxattachd`, provides automatic reattachment of sites. The `vxattachd` daemon uses the `vxnotify` mechanism to monitor storage coming back online on a site after a previous failure, and to restore redundancy of mirrors across sites.

If the hot-relocation daemon, `vxrelocd`, is running, `vxattachd` attempts to reattach the site, and allows `vxrelocd` to try to use the available disks in the disk group to relocate the failed subdisks. If `vxrelocd` succeeds in relocating the failed subdisks, it starts the recovery of the plexes at the site. When all the plexes have been recovered, the plexes are put into the ACTIVE state, and the state of the site is set to ACTIVE.

If `vxrelocd` is not running, `vxattachd` reattaches a site only when all the disks at that site become accessible. After reattachment succeeds, `vxattachd` sets the site state to ACTIVE, and initiates recovery of the plexes. When all the plexes have been recovered, the plexes are put into the ACTIVE state.

Note: vxattachd does not try to reattach a site that you have explicitly detached by using the vxdg detachsite command.

The automatic site reattachment feature is enabled by default. The vxattachd daemon uses email to notify root of any attempts to reattach sites and to initiate recovery of plexes at those sites.

To send mail to other users, add the user name to the line that starts vxattachd in the following startup script, and reboot the system.

For RHEL 7, SLES 12, and supported RHEL-compatible distributions:

```
/etc/vx/vxvm-recover
```

For earlier versions of RHEL, SLES and supported RHEL-compatible distributions:

```
/etc/init.d/vxvm-recover
```

If you do not want a site to be recovered automatically, kill the vxattachd daemon, and prevent it from restarting. If you stop vxattachd, the automatic plex reattachment also stops. To kill the daemon, run the following command from the command line:

```
# ps -afe
```

Locate the process table entry for vxattachd, and kill it by specifying its process ID:

```
# kill -9 PID
```

If there is no entry in the process table for vxattachd, the automatic site reattachment feature is disabled.

To prevent the automatic site reattachment feature from being restarted, comment out the line that starts vxattachd in the vxvm-recover startup script.

Section 5

Optimizing I/O performance

Veritas File System I/O

This chapter includes the following topics:

About Veritas File System I/O

VxFS processes two basic types of file system I/O:

- Sequential
- Random or I/O that is not sequential

For sequential I/O, VxFS employs a read-ahead policy by default when the application is reading data. For writing, it allocates contiguous blocks if possible. In most cases, VxFS handles I/O that is sequential through buffered I/O. VxFS handles random or nonsequential I/O using direct I/O without buffering.

VxFS provides a set of I/O cache advisories for use when accessing files.

See the *Veritas File System Programmer's Reference Guide*.

See the `vxfsio`(7) manual page.

Buffered and Direct I/O

VxFS responds with read-ahead for sequential read I/O. This results in buffered I/O. The data is prefetched and retained in buffers for the application. The data buffers are commonly referred to as VxFS buffer cache. This is the default VxFS behavior.

On the other hand, direct I/O does not buffer the data when the I/O to the underlying device is completed. This saves system resources like memory and CPU usage. Direct I/O is possible only when alignment and sizing criteria are satisfied.

See “Direct I/O requirements” on page 333.

All of the supported platforms have a VxFS buffered cache. Each platform also has either a page cache or its own buffer cache. These caches are commonly known as the file system caches.

Direct I/O does not use these caches. The memory used for direct I/O is discarded after the I/O is complete, and is therefore not buffered.

Direct I/O

Direct I/O is an unbuffered form of I/O. If the `VX_DIRECT` advisory is set, the user is requesting direct data transfer between the disk and the user-supplied buffer for reads and writes. This bypasses the kernel buffering of data, and reduces the CPU overhead associated with I/O by eliminating the data copy between the kernel buffer and the user's buffer. This also avoids taking up space in the buffer cache that might be better used for something else. The direct I/O feature can provide significant performance gains for some applications.

The direct I/O and `VX_DIRECT` advisories are maintained on a per-file-descriptor basis.

Direct I/O requirements

For an I/O operation to be performed as direct I/O, it must meet certain alignment criteria. The alignment constraints are usually determined by the disk driver, the disk controller, and the system memory management hardware and software.

The requirements for direct I/O are as follows:

- The starting file offset must be aligned to a 512-byte boundary.
- The ending file offset must be aligned to a 512-byte boundary, or the length must be a multiple of 512 bytes.
- The memory buffer must start on an 8-byte boundary.

Direct I/O versus synchronous I/O

Because direct I/O maintains the same data integrity as synchronous I/O, it can be used in many applications that currently use synchronous I/O. If a direct I/O request does not allocate storage or extend the file, the inode is not immediately written.

Direct I/O CPU overhead

The CPU cost of direct I/O is about the same as a raw disk transfer. For sequential I/O to very large files, using direct I/O with large transfer sizes can provide the same speed as buffered I/O with much less CPU overhead.

If the file is being extended or storage is being allocated, direct I/O must write the inode change before returning to the application. This eliminates some of the performance advantages of direct I/O.

Discovered Direct I/O

Discovered Direct I/O is a file system tunable that is set using the `vxtunefs` command. When the file system gets an I/O request larger than the `discovered_direct_iosz`, it tries to use direct I/O on the request. For large I/O sizes, Discovered Direct I/O can perform much better than buffered I/O.

Discovered Direct I/O behavior is similar to direct I/O and has the same alignment constraints, except writes that allocate storage or extend the file size do not require writing the inode changes before returning to the application.

Unbuffered I/O

If the `VX_UNBUFFERED` advisory is set, I/O behavior is the same as direct I/O with the `VX_DIRECT` advisory set, so the alignment constraints that apply to direct I/O also apply to unbuffered I/O. For unbuffered I/O, however, if the file is being extended, or storage is being allocated to the file, inode changes are not updated synchronously before the write returns to the user. The `VX_UNBUFFERED` advisory is maintained on a per-file-descriptor basis.

Data synchronous I/O

If the `VX_DSYNC` advisory is set, the user is requesting data synchronous I/O. In synchronous I/O, the data is written, and the inode is written with updated times and, if necessary, an increased file size. In data synchronous I/O, the data is transferred to disk synchronously before the write returns to the user. If the file is not extended by the write, the times are updated in memory, and the call returns to the user. If the file is extended by the operation, the inode is written before the write returns.

The direct I/O and VX_DSYNC advisories are maintained on a per-file-descriptor basis.

Data synchronous I/O vs. synchronous I/O

Like direct I/O, the data synchronous I/O feature can provide significant application performance gains. Because data synchronous I/O maintains the same data integrity as synchronous I/O, it can be used in many applications that currently use synchronous I/O. If the data synchronous I/O does not allocate storage or extend the file, the inode is not immediately written. The data synchronous I/O does not have any alignment constraints, so applications that find it difficult to meet the alignment constraints of direct I/O should use data synchronous I/O.

If the file is being extended or storage is allocated, data synchronous I/O must write the inode change before returning to the application. This case eliminates the performance advantage of data synchronous I/O.

Concurrent I/O

Concurrent I/O (`VX_CONCURRENT`) allows multiple processes to read from or write to the same file without blocking other `read`(2) or `write`(2) calls. POSIX semantics requires `read` and `write` calls to be serialized on a file with other `read` and `write` calls. With POSIX semantics, a `read` call either reads the data before or after the `write` call occurred. With the `VX_CONCURRENT` advisory set, the `read` and `write` operations are not serialized as in the case of a character device. This advisory is generally used by applications that require high performance for accessing data and do not perform overlapping writes to the same file. It is the responsibility of the application or the running threads to coordinate the `write` activities to the same file when using Concurrent I/O.

Concurrent I/O can be enabled in the following ways:

- By specifying the `VX_CONCURRENT` advisory flag for the file descriptor in the `VX_SETCACHE` ioctl command. Only the `read`(2) and `write`(2) calls occurring through this file descriptor use concurrent I/O. The `read` and `write` operations occurring through other file descriptors for the same file will still follow the POSIX semantics.
  See `vxfsio`(7) manual page.
- By using the `cio` mount option. The `read`(2) and `write`(2) operations occurring on all of the files in this particular file system will use concurrent I/O.
  See “cio mount option” on page 175.
  See the `mount_vxfs`(1M) manual page.

Cache advisories

VxFS allows an application to set cache advisories for use when accessing files. VxFS cache advisories enable applications to help monitor the buffer cache and provide information on how better to tune the buffer cache to improve performance gain.

The cache advisory let's you know whether you could have avoided a later re-read of block X if the buffer cache had been a little larger. Conversely, the cache advisory can also let you know that you could safely reduce the buffer cache size without putting block X into jeopardy.

These advisories are in memory only and do not persist across reboots. Some advisories are currently maintained on a per-file, not a per-file-descriptor, basis. Only one set of advisories can be in effect for all accesses to the file. If two conflicting applications set different advisories, both must use the advisories that were last set.

All advisories are set using the `VX_SETCACHE` ioctl command. The current set of advisories can be obtained with the `VX_GETCACHE` ioctl command.

See the `vxfsio`(7) manual page.

Freezing and thawing a file system

Freezing a file system is a necessary step for obtaining a stable and consistent image of the file system at the volume level. Consistent volume-level file system images can be obtained and used with a file system snapshot tool. The freeze operation flushes all buffers and pages in the file system cache that contain dirty metadata and user data. The operation then suspends any new activity on the file system until the file system is thawed.

The `VX_FREEZE` ioctl command is used to freeze a file system. Freezing a file system temporarily blocks all I/O operations to a file system and then performs a sync on the file system. When the `VX_FREEZE` ioctl is issued, all access to the file system is blocked at the system call level. Current operations are completed and the file system is synchronized to disk.

When the file system is frozen, any attempt to use the frozen file system, except for a `VX_THAW` ioctl command, is blocked until a process executes the `VX_THAW` ioctl command or the time-out on the freeze expires.

Getting the I/O size

VxFS provides the `VX_GET_IOPARAMETERS` ioctl to get the recommended I/O sizes to use on a file system. This ioctl can be used by the application to make decisions about the I/O sizes issued to VxFS for a file or file device.

See the `vxtunefs`(1M) and `vxfsio`(7) manual pages.

About Veritas InfoScale product components database accelerators

The major concern in any environment is maintaining respectable performance or meeting performance service level agreements (SLAs). Veritas InfoScale product components improve the overall performance of database environments in a variety of ways.

Table 14-1 Veritas InfoScale product components database accelerators

Veritas InfoScale database accelerator	Supported databases	Use cases and considerations
Oracle Disk Manager (ODM)	Oracle	■ To improve Oracle performance and manage system bandwidth through an improved Application Programming Interface (API) that contains advanced kernel support for file I/O. ■ To use Oracle Resilvering and turn off Veritas Volume Manager Dirty Region Logging (DRL) to increase performance, use ODM. ■ To reduce the time required to restore consistency, freeing more I/O bandwidth for business-critical applications, use SmartSync recovery accelerator.
Cached Oracle Disk Manager (Cached OD M)	Oracle	To enable selected I/O to use caching to improve ODM I/O performance, use Cached ODM.

Table 14-1 Veritas InfoScale product components database accelerators *(continued)*

Veritas InfoScale database accelerator	Supported databases	Use cases and considerations
Concurrent I/O	DB2 Sybase	Concurrent I/O (CIO) is optimized for DB2 and Sybase environments To achieve improved performance for databases run on VxFS file systems without restrictions on increasing file size, use Veritas InfoScale Concurrent I/O.

These database accelerator technologies enable database performance equal to raw disk partitions, but with the manageability benefits of a file system. With the Dynamic Multi-pathing (DMP) feature of Storage Foundation, performance is maximized by load-balancing I/O activity across all available paths from server to array. DMP supports all major hardware RAID vendors, hence there is no need for third-party multi-pathing software, reducing the total cost of ownership.

Veritas InfoScale database accelerators enable you to manage performance for your database with more precision.

For details about using ODM and Cached ODM for Oracle, see *Veritas InfoScale Storage and Availability Management for Oracle Databases*.

For details about using Concurrent I/O for DB2, see *Veritas InfoScale Storage and Availability Management for DB2 Databases*.

Chapter 15

Veritas Volume Manager I/O

This chapter includes the following topics:

Veritas Volume Manager throttling of administrative I/O

Veritas Volume Manager (VxVM) provides throttling of administrative I/O. During heavy I/O loads, VxVM throttles I/O that it creates to do administrative operations. This behavior ensures that the administrative I/Os do not affect the application I/O performance. When the application I/O load is lighter, VxVM increases the bandwidth usage for administrative I/O operations.

VxVM automatically manages the I/O throttling for administrative tasks, based on its perceived load on the storage. Currently, I/O throttling is supported for the copy operations which use ATOMIC_COPY and involve one destination mirror. The I/O throttling is transparent, and does not change the command usage or output. The following commands are supported:

- `vxassist mirror`
- `vxassist snapcreate`
- `vxevac`
- `vxplex att`
- `vxplex cp`

- `vxplex mv`
- `vxsnap addmir`
- `vxsnap reattach`
- `vxsd mv`

The administrative I/O operations allocate memory for I/O from a separate memory pool. You can tune the maximum size of this pool with the tunable parameter, `vol_max_adminio_poolsz`.

For example, to change the `vol_max_adminio_poolsz` parameter to 256MB, use tbe following command:

```
# vxtune vol_max_adminio_poolsz 256M
```

See the `vxtune`(1M) command for more info.

Managing application I/O workloads using maximum IOPS settings

When multiple applications use a common storage subsystem, it is important to balance application I/O requests in a way that allows multiple applications to co-exist in a shared environment. You can address this need by setting a maximum threshold on the I/O operations per second (IOPS) for the volumes of an application. The volumes of an application are grouped to form an application volume group. The maximum IOPS limit determines the maximum number of I/Os processed per second collectively by all the volumes in an application volume group.

When an I/O request comes in from an application, it is serviced by the volumes in the group until the application volume group reaches the IOPS limit. When the group exceeds this limit for a specified time interval, further I/O requests on the group are queued. The queued I/Os are taken up on priority in the next time interval along with new I/O requests from the application.

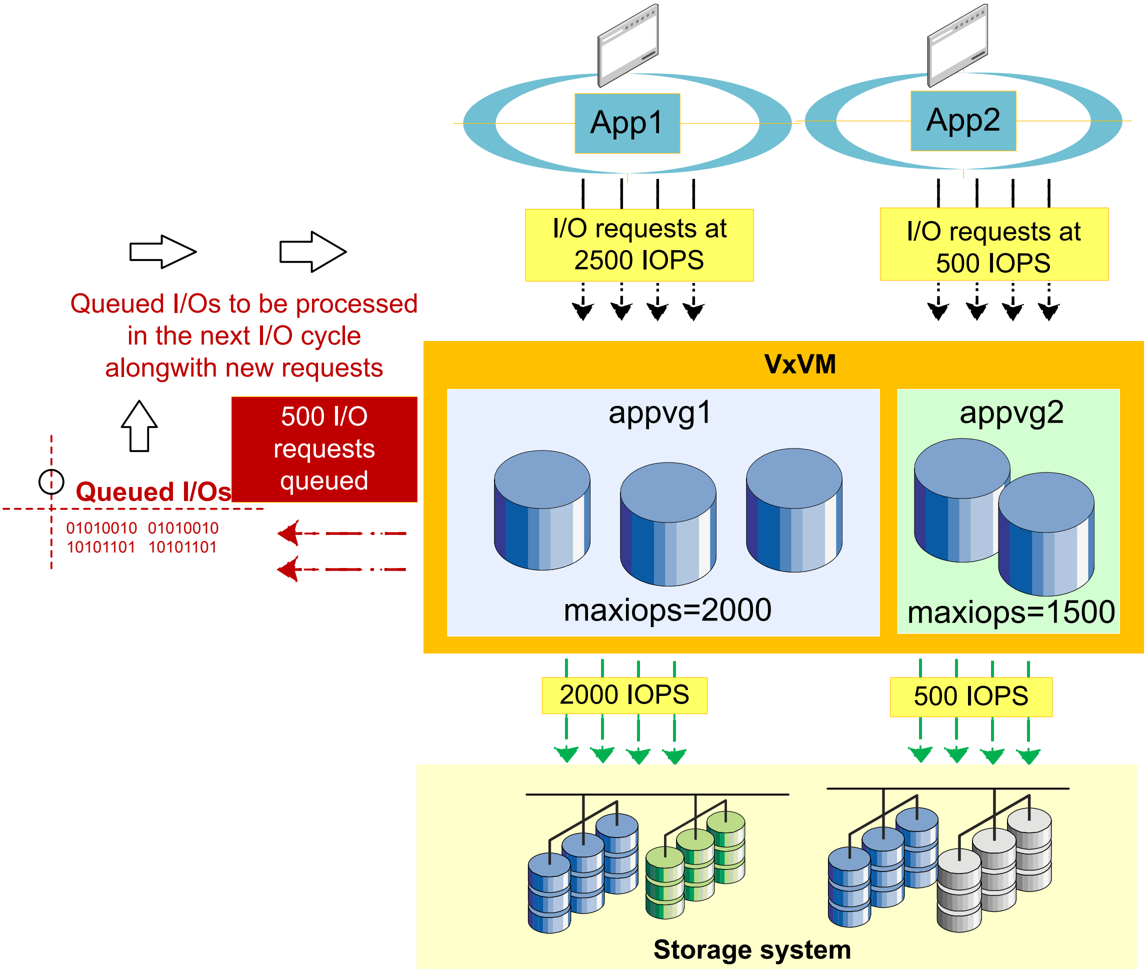
You will want to consider the following factors when you set the maximum IOPS threshold:

- Storage capacity of the shared subsystem
- Number of active applications
- I/O requirements of the individual applications
- VxVM adminstrative I/Os

For instance, if the volumes have associated VxVM instant or space-optimized snapshot volumes, you will want to set a smaller IOPS threshold on the source volumes.

Figure 15-1 illustrates the process.

Figure 15-1 Managing application I/O workloads



About application volume groups

An application volume group is a logical grouping of volumes associated with an application. The group may contain one or more volumes. All the volumes in the application volume group must be selected from the same disk group. The volumes may belong to a private or shared disk group. Set the maximum IOPS threshold on the application volume group to balance multiple application I/O workloads. The IOPS value is set as a combined threshold for all the volumes in the application volume group.

You can construct multiple such application volume groups, bringing together volumes that have similar I/O load characteristics. For example, the I/O load characteristic of redo log volumes of a database application will differ from that of the application's archive volumes and will therefore need different IOPS thresholds. In such scenarios, create separate application volume groups and set respective IOPS thresholds. A volume cannot be shared between multiple application volume groups for setting maximum IOPS limits.

In clustered environments, you can create an application volume group and set its threshold from any node in the cluster. The application volume group is visible on all nodes and the same threshold value is propagated to all nodes in the cluster. The threshold, however, applies to each node independently.

The `vxvolgrp` command can be used to manage the application volume group and its settings.

See the `vxvolgrp`(1M) manual page.

The `vxstat` command provides a detailed view of IOPS statistics on each application volume group.

See the `vxstat`(1M) manual page.

Creating application volume groups

Identify the volumes that are not already part of an existing application volume group. Ensure that the volumes you select belong to the same disk group.

You can use the `vxvolgrp` command to create application volume groups.

See the `vxvolgrp`(1M) manual page.

To create application volume groups

1 Create the application volume group.

   For example, the following command creates an application volume group *datavol_grp* consisting of volumes *datavol1, datavol2, datavol3*. *oradg* is the name of the disk group from which the volumes are selected.

```
# vxvolgrp -g oradg make datavol_grp \
datavol1 datavol2 datavol3
```

2 Verify that the application volume group is created successfully.

   For example:

```
# vxvolgrp -g oradg list datavol_grp
Volume Group:                   datavol_grp
volume(s):                      datavol1 datavol2 datavol3
```

Viewing the list of application volume groups

You can use the `vxvolgrp` command to:

- View the list of existing application volume groups
- View detailed information about a specific application volume group

See the `vxvolgrp`(1M) manual page.

To view the list of existing application volume groups:

```
# vxvolgrp -g dgname list
```

To view the list of application volume groups created from volumes in the *oradg* disk group:

```
# vxvolgrp -g oradg list
Volume Group:                   datavol_grp
volume(s):                      datavol1 datavol2 datavol3

Volume Group:                   logvol_grp
volume(s):                      logvol1 logvol2 logvol3 logvol4
```

For example, to view information about the application volume group *datavol_grp*:

```
# vxvolgrp -g oradg list datavol_grp
Volume Group:                   datavol_grp
volume(s):                      datavol1 datavol2 datavol3
```

Setting the maximum IOPS threshold on application volume groups

The maximum IOPS threshold is set on the application volume group associated with an application. This is a group threshold that applies to all volumes in the group. You can update the IOPS setting at any time, even when an application is running.

You can use the `vxvolgrp` command to set a maximum IOPS value for the group.

See the `vxvolgrp`(1M) manual page.

Note: You can not set a threshold for individual volumes in the group.

To set the maximum IOPS threshold on application volume groups

1 Set a maximum IOPS value for the application volume group.

For example, the following command sets a maximum IOPS threshold of *1000* on the application volume group *datavol_grp*:

```
# vxvolgrp -g oradg set datavol_grp maxiops=1000
```

2 Verify that the maximum IOPS threshold is set successfully.

For example:

```
# vxvolgrp -g oradg list datavol_grp
Volume Group:                    datavol_grp
volume(s):                       datavol1 datavol2 datavol3
volume group attributes:         maxiops=1000
```

Viewing the IOPS statistics for application volume groups

The `-G` option of the `vxstat` command provides a detailed view of the IOPS statistics for an application volume group. You can obtain a detailed view of the statistics per second or an average of the accumulated statistics per second over a specified time interval.

For clustered environments, the detailed statistics is displayed for each node in the cluster.

See the `vxstat`(1M) manual page.

You can view the following information:

MaxIOPS	The maximum IOPS threshold set for an application volume group

ServicedIOPS	The number of I/Os per second processed for an application volume group
QueuedIOPS	The number of I/Os per second throttled for an application volume group

For example, to view the IOPS statistics per second for existing application volume groups:

```
# vxstat -g oradg -G  <<<< average per second output
                                       AVG PER SECOND VOLUMEGROUP STATISTICS
TYP NAME           MaxIOPS     IncomingIOPS    ServicedIOPS       QueuedIOPS(Transient)
grp datavol_grp     1000          897             893                56
grp logvol_grp      600           599             598               404
```

To view the IOPS statistics per second for the application volume group *datavol_grp*:

```
# vxstat -g oradg -G datavol_grp
                                       AVG PER SECOND VOLUMEGROUP STATISTICS
TYP NAME          MaxIOPS     ServicedIOPS       QueuedIOPS(Transient)
grp datavol_grp  1000             980               360
```

To view the average IOPS statistics over a 3-second interval for the application volume group *datavol_grp*:

```
# vxstat -g oradg -G datavol_grp -i 3
                                       AVG PER SECOND VOLUMEGROUP STATISTICS
TYP NAME                               MaxIOPS   ServicedIOPS  QueuedIOPS(Transient)

vmr720-23 Tue 12 Jan 2016 02:42:52 PM UTC
grp datavol_grp                           1000   1000           386

vmr720-23 Tue 12 Jan 2016 02:42:55 PM UTC
grp datavol_grp                           1000   1000           393

vmr720-23 Tue 12 Jan 2016 02:42:58 PM UTC
grp datavol_grp                           1000   1000           393

vmr720-23 Tue 12 Jan 2016 02:43:01 PM UTC
grp datavol_grp                           1000   1000           391

vmr720-23 Tue 12 Jan 2016 02:43:04 PM UTC
grp datavol_grp                           1000    589           129
```

```
vmr720-23 Tue 12 Jan 2016 02:43:07 PM UTC
grp datavol_grp                          1000      0           0
```

Removing the maximum IOPS setting from application volume groups

You can use the `vxvolgrp` command to remove the maximum IOPS setting.

See the `vxvolgrp`(1M) manual page.

For example, to remove the maximum IOPS setting from application volume group *datavol_grp*:

```
# vxvolgrp -g oradg clear datavol_grp maxiops
# vxvolgrp -g oradg list datavol_grp
Volume Group:                   datavol_grp
volume(s):                      datavol1 datavol2 datavol3
```

Adding volumes to an application volume group

You can add volumes to an existing application volume group. Ensure that you select a volume from within the disk group. It must not belong to any other application volume group. VxVM tags the new volume with the name of the application volume group and the corresponding maximum IOPS value.

You can use the `vxvolgrp` command to add volumes to an application volume group.

See the `vxvolgrp`(1M) manual page.

For example, to add volumes to the application volume group *logvol_grp*:

```
# vxvolgrp -g oradg addvol logvol_grp \
logvol3 logvol4
# vxvolgrp -g oradg list logvol_grp
Volume Group:                   logvol_grp
volume(s):                      logvol1 logvol2 logvol3 logvol4
volume group attributes:        maxiops=1200
```

Removing volumes from an application volume group

When you remove a volume from an application volume group, VxVM clears the application volume group name and the IOPS threshold from the volume. When you remove the only remaining volume in the group, the application volume group is deleted.

You can use the `vxvolgrp` command to remove volumes to an application volume group.

See the vxvolgrp(1M) manual page.

For example, to remove a volume from the application volume group *datavol_grp*:

```
# vxvolgrp -g oradg rmvol datavol_grp datavol3
# vxvolgrp -g oradg list datavol_grp
Volume Group:                   datavol_grp
volume(s):                      datavol1 datavol2
volume group attributes:        maxiops=1000
```

Removing an application volume group

Removing an application volume group frees up the volumes for use by other application volume groups. The IOPS setting and associated tags are removed from the volumes.

You can use the vxvolgrp command to remove an application volume group.

See the vxvolgrp(1M) manual page.

For example, to remove the application volume group *logvol_grp*:

```
# vxvolgrp -g oradg remove logvol_grp
```

Section 6

Using Point-in-time copies

Understanding point-in-time copy methods

This chapter includes the following topics:

About point-in-time copies

Storage Foundation offers a flexible and efficient means of managing business-critical data. Storage Foundation lets you capture an online image of an actively changing database at a given instant, called a point-in-time copy.

More and more, the expectation is that the data must be continuously available (24x7) for transaction processing, decision making, intellectual property creation, and so forth. Protecting the data from loss or destruction is also increasingly important. Formerly, data was taken out of service so that the data did not change while data backups occured; however, this option does not meet the need for minimal down time.

A point-in-time copy enables you to maximize the online availability of the data. You can perform system backup, upgrade, or perform other maintenance tasks on the point-in-time copies. The point-in-time copies can be processed on the same host as the active data, or a different host. If required, you can offload processing of the point-in-time copies onto another host to avoid contention for system resources on your production server. This method is called off-host processing. If implemented correctly, off-host processing solutions have almost no impact on the performance of the primary production system.

For more information about how to use point-in-time copies for particular use cases, see the *Veritas InfoScale Solutions Guide*.

When to use point-in-time copies

The following typical activities are suitable for point-in-time copy solutions implemented using Veritas InfoScale FlashSnap:

- Data backup —Many enterprises require 24 x 7 data availability. They cannot afford the downtime involved in backing up critical data offline. By taking snapshots of your data, and backing up from these snapshots, your business-critical applications can continue to run without extended downtime or impacted performance.

- Providing data continuity —To provide continuity of service in the event of primary storage failure, you can use point-in-time copy solutions to recover application data. In the event of server failure, you can use point-in-time copy solutions in conjunction with the high availability cluster functionality of SFCFSHA or SFHA.

- Decision support analysis and reporting—Operations such as decision support analysis and business reporting may not require access to real-time information. You can direct such operations to use a replica database that you have created from snapshots, rather than allow them to compete for access to the primary database. When required, you can quickly resynchronize the database copy with the data in the primary database.

- Testing and training—Development or service groups can use snapshots as test data for new applications. Snapshot data provides developers, system testers and QA groups with a realistic basis for testing the robustness, integrity and performance of new applications.

- Database error recovery—Logic errors caused by an administrator or an application program can compromise the integrity of a database. You can recover a database more quickly by restoring the database files by using Storage Checkpoints or a snapshot copy than by full restoration from tape or other backup media.

Use Storage Checkpoints to quickly roll back a database instance to an earlier point in time.

- Cloning data—You can clone your file system or application data. This functionality enable you to quickly and efficiently provision virtual desktops.

All of the snapshot solutions mentioned above are also available on the disaster recovery site, in conjunction with Volume Replicator.

For more information about snapshots with replication, see the *Veritas InfoScale Replication Administrator's Guide*.

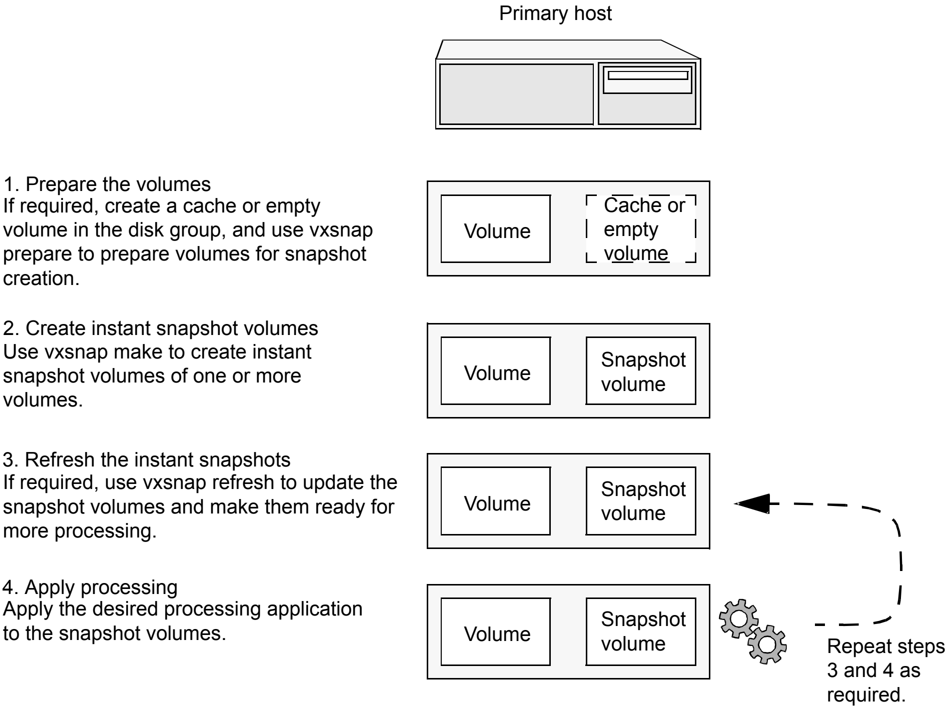
Storage Foundation provides several point-in-time copy solutions that support your needs, including the following use cases:

- Creating a replica database for decision support.
- Backing up and recovering a database with snapshots.
- Backing up and recovering an off-host cluster file system
- Backing up and recovering an online database.

Implementing point-in time copy solutions on a primary host

Figure 16-1 illustrates the steps that are needed to set up the processing solution on the primary host.

Figure 16-1 Using snapshots and FastResync to implement point-in-time copy solutions on a primary host



1. Prepare the volumes
If required, create a cache or empty volume in the disk group, and use vxsnap prepare to prepare volumes for snapshot creation.

2. Create instant snapshot volumes
Use vxsnap make to create instant snapshot volumes of one or more volumes.

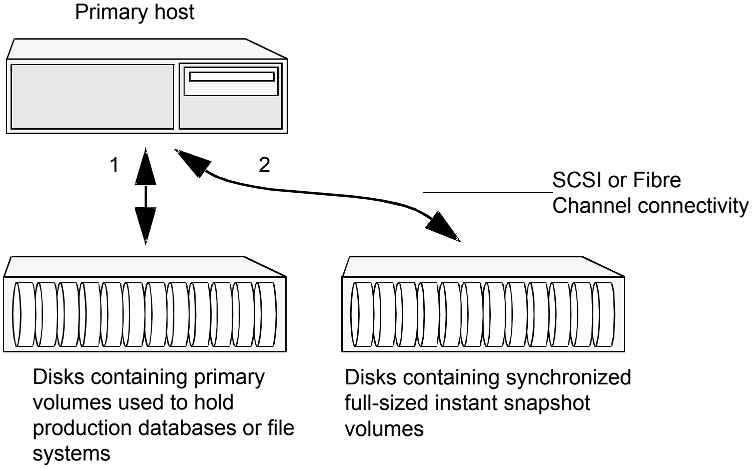
3. Refresh the instant snapshots
If required, use vxsnap refresh to update the snapshot volumes and make them ready for more processing.

4. Apply processing
Apply the desired processing application to the snapshot volumes.

Note: The Disk Group Split/Join functionality is not used. As all processing takes place in the same disk group, synchronization of the contents of the snapshots from the original volumes is not usually required unless you want to prevent disk contention. Snapshot creation and updating are practically instantaneous.

Figure 16-2 shows the suggested arrangement for implementing solutions where the primary host is used and disk contention is to be avoided.

Figure 16-2 Example point-in-time copy solution on a primary host


Primary host
1
2
SCSI or Fibre Channel connectivity
Disks containing primary volumes used to hold production databases or file systems
Disks containing synchronized full-sized instant snapshot volumes

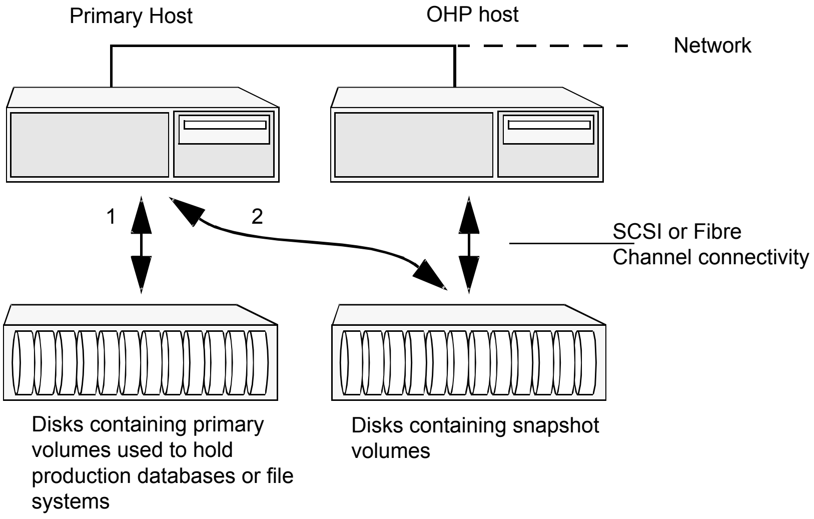
In this setup, it is recommended that separate paths (shown as 1 and 2) from separate controllers be configured to the disks containing the primary volumes and the snapshot volumes. This avoids contention for disk access, but the primary host's CPU, memory and I/O resources are more heavily utilized when the processing application is run.

Note: For space-optimized or unsynchronized full-sized instant snapshots, it is not possible to isolate the I/O pathways in this way. This is because such snapshots only contain the contents of changed regions from the original volume. If applications access data that remains in unchanged regions, this is read from the original volume.

Implementing off-host point-in-time copy solutions

Figure 16-3 illustrates that, by accessing snapshot volumes from a lightly loaded host (shown here as the OHP host), CPU- and I/O-intensive operations for online backup and decision support are prevented from degrading the performance of the primary host that is performing the main production activity (such as running a database).

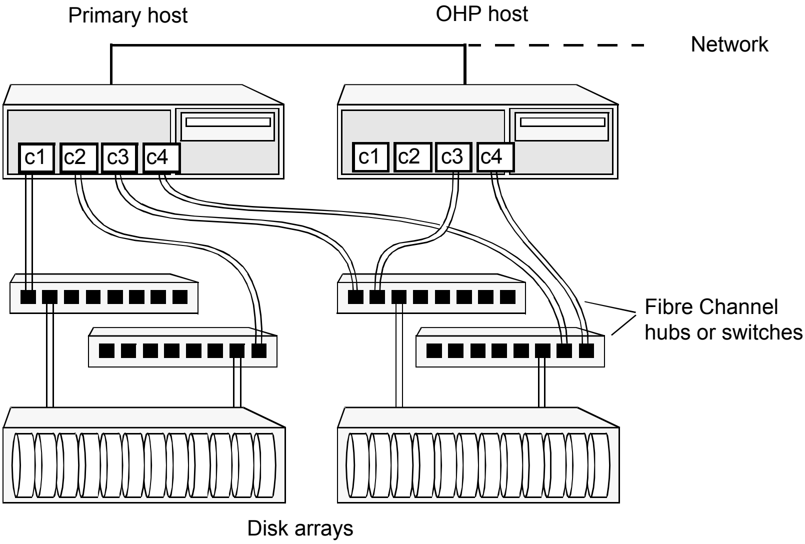
Figure 16-3 Example implementation of an off-host point-in-time copy solution


Primary Host
OHP host
Network
1
2
SCSI or Fibre Channel connectivity
Disks containing primary volumes used to hold production databases or file systems
Disks containing snapshot volumes

Also, if you place the snapshot volumes on disks that are attached to host controllers other than those for the disks in the primary volumes, it is possible to avoid contending with the primary host for I/O resources. To implement this, paths 1 and 2 shown in the Figure 16-3 should be connected to different controllers.

Figure 16-4 shows an example of how you might achieve such connectivity using Fibre Channel technology with 4 Fibre Channel controllers in the primary host.

Figure 16-4 Example connectivity for off-host solution using redundant-loop access



This layout uses redundant-loop access to deal with the potential failure of any single component in the path between a system and a disk array.

Note: On some operating systems, controller names may differ from what is shown here.

Figure 16-5 shows how off-host processing might be implemented in a cluster by configuring one of the cluster nodes as the OHP node.

Figure 16-5 Example implementation of an off-host point-in-time copy solution using a cluster node

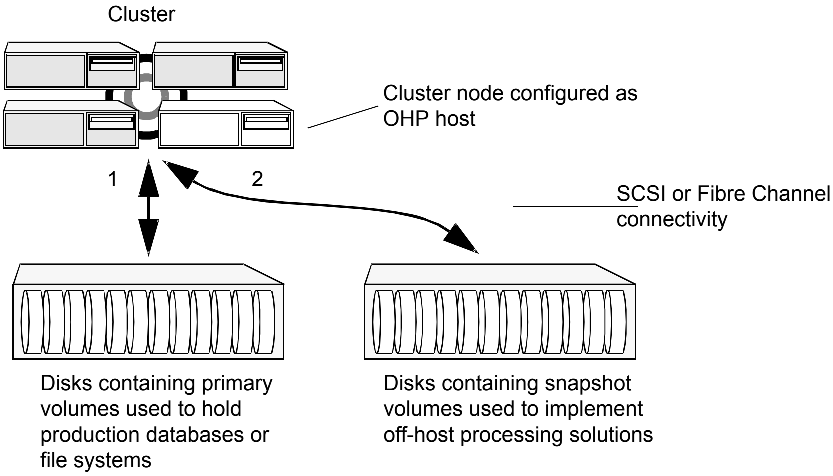
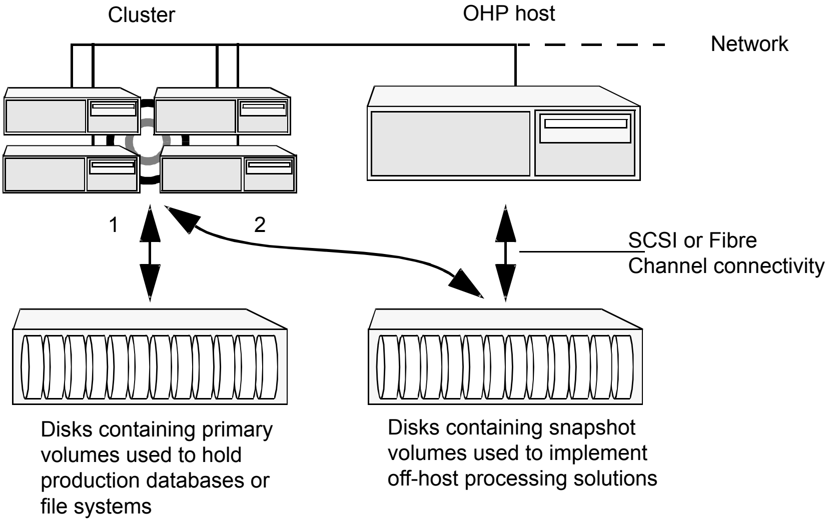

Cluster
Cluster node configured as OHP host
1
2
SCSI or Fibre Channel connectivity
Disks containing primary volumes used to hold production databases or file systems
Disks containing snapshot volumes used to implement off-host processing solutions

Figure 16-6 shows an alternative arrangement, where the OHP node could be a separate system that has a network connection to the cluster, but which is not a cluster node and is not connected to the cluster's private network.

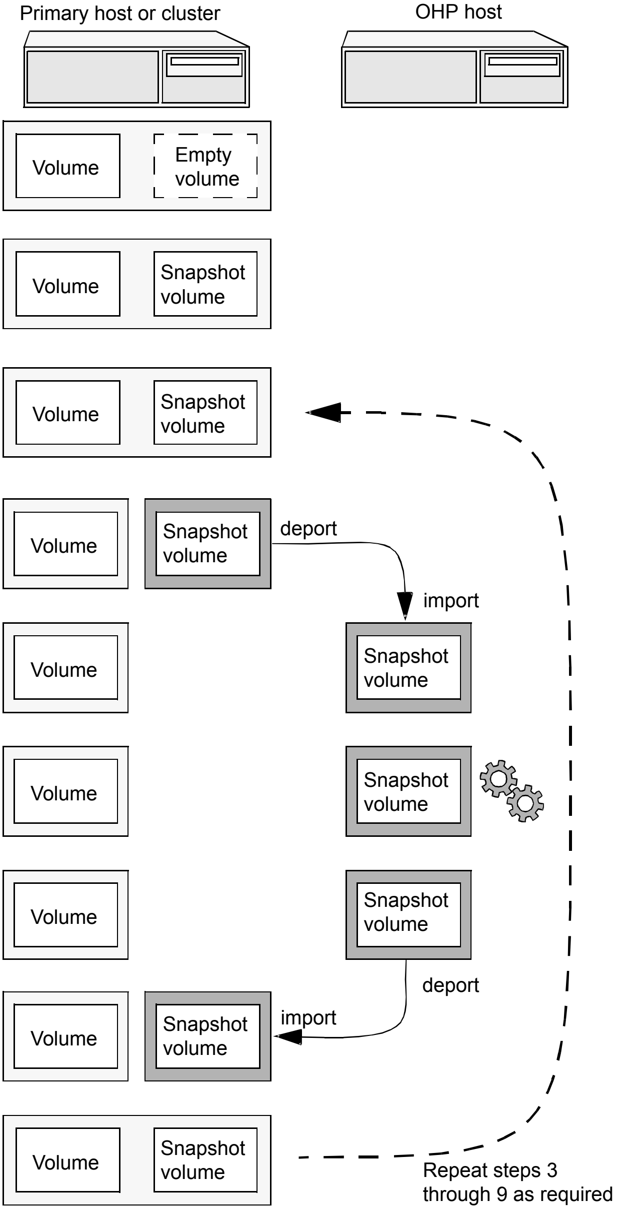
Figure 16-6 Example implementation of an off-host point-in-time copy solution using a separate OHP host


Cluster
OHP host
Network
1
2
SCSI or Fibre Channel connectivity
Disks containing primary volumes used to hold production databases or file systems
Disks containing snapshot volumes used to implement off-host processing solutions

Note: For off-host processing, the example scenarios in this document assume that a separate OHP host is dedicated to the backup or decision support role. For clusters, it may be simpler, and more efficient, to configure an OHP host that is not a member of the cluster.

Figure 16-7 illustrates the steps that are needed to set up the processing solution on the primary host.

Figure 16-7 Implementing off-host processing solutions



Repeat steps 3 through 9 as required

1. Prepare the volumes
If required, create an empty volume in the disk group, and use vxsnap prepare to prepare volumes for snapshot creation.

2. Create snapshot volumes
Use vxsnap make to create synchronized snapshot volumes. (Use vxsnap print to check the status of synchronization.)

3. Refresh snapshot mirrors
If required, use vxsnap refresh to update the snapshot volumes. (Use vxsnap print to check the status of synchronization.)

4. Split and deport disk group
Use vxdg split to move the disks containing the snapshot volumes to a separate disk group. Use vxdg deport to deport this disk group.

5. Import disk group
Use vxdg import to import the disk group containing the snapshot volumes on the OHP host.

6. Apply off-host processing
Apply the desired off-host processing application to the snapshot volume on the OHP host.

7. Deport disk group
Use vxdg deport to deport the disk group containing the snapshot volumes from the OHP host.

8. Import disk group
Use vxdg import to import the disk group containing the snapshot volumes on the primary host.

9. Join disk groups
Use vxdg join to merge the disk group containing the snapshot volumes with the original volumes' disk group.

Disk Group Split/Join is used to split off snapshot volumes into a separate disk group that is imported on the OHP host.

Note: As the snapshot volumes are to be moved into another disk group and then imported on another host, their contents must first be synchronized with the parent volumes. On reimporting the snapshot volumes, refreshing their contents from the original volume is speeded by using FastResync.

About Storage Foundation point-in-time copy technologies

This topic introduces the point-in-time copy solutions that you can implement using the Veritas FlashSnap™ technology. Veritas FlashSnap technology requires a Veritas InfoScale Enterprise or Storage licenses.

Veritas InfoScale FlashSnap offers a flexible and efficient means of managing business critical data. It allows you to capture an online image of actively changing data at a given instant: a point-in-time copy. You can perform system backup, upgrade and other maintenance tasks on point-in-time copies while providing continuous availability of your critical data. If required, you can offload processing of the point-in-time copies onto another host to avoid contention for system resources on your production server.

The following kinds of point-in-time copy solution are supported by the FlashSnap license:

- Volume-level solutions. There are several types of volume-level snapshots. These features are suitable for solutions where separate storage is desirable to create the snapshot. For example, lower-tier storage. Some of these techniques provided exceptional offhost processing capabilities.
- File system-level solutions use the Storage Checkpoint feature of Veritas File System. Storage Checkpoints are suitable for implementing solutions where storage space is critical for:
  - File systems that contain a small number of mostly large files.
  - Application workloads that change a relatively small proportion of file system data blocks (for example, web server content and some databases).
  - Applications where multiple writable copies of a file system are required for testing or versioning.

  See "Storage Checkpoints" on page 365.
- File level snapshots.
  The FileSnap feature provides snapshots at the level of individual files.

Comparison of Point-in-time copy solutions

The following table shows a side-by-side comparison of the Storage Foundation Point-in-time copy solutions.

Table 16-1 Comparison of Point-in-time copy solutions

Solution	Granularity	Location of snapped data	Snapshot technique	Internal content	Exported content	Can be moved off-host	Availability
Instant full-sized snapshot	Volume	Separate volume	Copy on write/ Full copy	Changed regions/ Full volume	Read/Write volume	Yes, after synchronization	Immediate
Instant space optimized snapshot	Volume	Cache object (Separate cache volume)	Copy on write	Changed regions	Read/Write volume	No	Immediate
Linked plex break-off	Volume	Separate volume	Copy on write/ Full copy	Changed regions/ Full volume	Read/Write volume	Yes, after synchronization	Immediate
Plex break-off using vxsnap	Volume	Separate volume	Copy on write/ Full copy	Changed regions/ Full volume	Read/Write volume	Yes, after synchronization	Immediate
Traditional plex break-off using vxassist	Volume	Separate volume	Full copy	Full volume	Read/Write volume	Yes, after synchronization	After full synch-ronization
Storage Checkpoint	File system	Space within file system	Copy on write	Changed file system blocks	Read/Write file system	No	Immediate
File system snapshot	File system	Separate volume	Copy on write	Changed file system blocks	Read-only file system	No	Immediate
FileSnap	File	Space within file system	Copy on write/Lazy copy on write	Changed file system blocks	Read/Write file system	No	Immediate

Volume-level snapshots

A volume snapshot is an image of a Veritas Volume Manager (VxVM) volume at a given point in time. You can also take a snapshot of a volume set.

Volume snapshots allow you to make backup copies of your volumes online with minimal interruption to users. You can then use the backup copies to restore data that has been lost due to disk failure, software errors or human mistakes, or to create replica volumes for the purposes of report generation, application development, or testing.

Volume snapshots can also be used to implement off-host online backup.

Physically, a snapshot may be a full (complete bit-for-bit) copy of the data set, or it may contain only those elements of the data set that have been updated since snapshot creation. The latter are sometimes referred to as allocate-on-first-write snapshots, because space for data elements is added to the snapshot image only when the elements are updated (overwritten) for the first time in the original data set. Storage Foundation allocate-on-first-write snapshots are called space-optimized snapshots.

Persistent FastResync of volume snapshots

If persistent FastResync is enabled on a volume, VxVM uses a FastResync map to keep track of which blocks are updated in the volume and in the snapshot.

When snapshot volumes are reattached to their original volumes, persistent FastResync allows the snapshot data to be quickly refreshed and re-used. Persistent FastResync uses disk storage to ensure that FastResync maps survive both system and cluster crashes. If persistent FastResync is enabled on a volume in a private disk group, incremental resynchronization can take place even if the host is rebooted.

Persistent FastResync can track the association between volumes and their snapshot volumes after they are moved into different disk groups. After the disk groups are rejoined, persistent FastResync allows the snapshot plexes to be quickly resynchronized.

Data integrity in volume snapshots

A volume snapshot captures the data that exists in a volume at a given point in time. As such, VxVM does not have any knowledge of data that is cached in memory by the overlying file system, or by applications such as databases that have files open in the file system. Snapshots are always crash consistent, that is, the snapshot can be put to use by letting the application perform its recovery. This is similar to how the application recovery occurs after a server crash. If the `fsgen` volume usage type is set on a volume that contains a mounted Veritas File System (VxFS), VxVM coordinates with VxFS to flush data that is in the cache to the volume. Therefore, these snapshots are always VxFS consistent and require no VxFS recovery while mounting.

For databases, a suitable mechanism must additionally be used to ensure the integrity of tablespace data when the volume snapshot is taken. The facility to temporarily suspend file system I/O is provided by most modern database software. The examples provided in this document illustrate how to perform this operation. For ordinary files in a file system, which may be open to a wide variety of different applications, there may be no way to ensure the complete integrity of the file data other than by shutting down the applications and temporarily unmounting the file system. In many cases, it may only be important to ensure the integrity of file data that is not in active use at the time that you take the snapshot. However, in all scenarios where application coordinate, snapshots are crash-recoverable.

Third-mirror break-off snapshots

A plex break-off snapshot uses an additional mirror to create the snapshot. Although you can create a plex break-off snapshot for a single plex volume, typically you take a snapshot of a mirrored volume. A mirrored volume has more than one plex or mirror, each of which is a copy of the data. The snapshot operation "breaks off" the plex, which becomes the snapshot volume. You can break off an existing plex or add a new plex specifically to serve as the snapshot mirror. Generally, you want to maintain redundancy for the original volume. If the original volume is a mirrored volume with two plexes, you add a third mirror for the snapshot. Hence, this type of snapshot is also known as a third-mirror snapshot.

The snapshot plex must be on a different disk from the existing plexes in the volume, within the same disk group. The disk must have enough disk space to contain the contents of the existing volume. If you have a one terabyte volume, you must have an additional one terabyte of disk space.

When you create the snapshot, the plexes are separated into two volumes. The original volume retains its original plex or plexes. The snapshot volume contains the snapshot plex. The original volume continues to take on I/O. The snapshot volume retains the data at the point of time when the snapshot was created, until you choose to perform processing on that volume.

You can make multiple snapshots, so you can have multiple copies of the original data.

Third-mirror break-off snapshots are suitable for write-intensive volumes (such as for database redo logs) where the copy-on-write mechanism of space-optimized or full-sized instant snapshots might degrade performance.

Space-optimized instant volume snapshots

Space-optimized snapshots do not contain complete physical images of the original data objects they represent. Space-optimized instant snapshots record changed regions in the original volume to a storage cache. As the original volume is written to, VxVM preserves its data in the cache before the write is committed. As the storage cache typically requires much less storage than the original volume, it is referred to as space-optimized. Space-optimized snapshots consume storage and I/O bandwidth in proportion to how much data on the original volume is updated during the life of the snapshot.

The benefits of space-optimized instant snapshots include immediate availability for use, quick refreshment, and easier configuration and administration. Because space-optimized snapshots consume less storage and I/O bandwidth than full-copy snapshots, you can take the snapshots much more frequently. This makes them well-suited for recovering from data corruption.

Space-optimized snapshots naturally tend to grow with age, as more of the data in the original objects changes, so they are inherently better-suited for shorter lifetimes.

Space-optimized snapshots cannot be taken off-host for auxiliary processing.

How space-optimized instant snapshots work

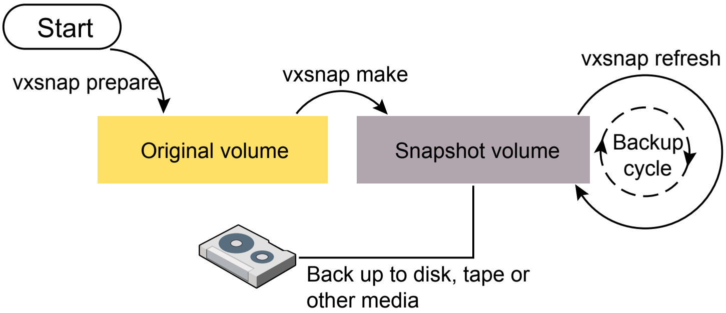
Space-optimized snapshots use a copy-on-write mechanism to make them immediately available for use when they are first created, or when their data is refreshed.

You can configure a single storage cache in a disk group that can be shared by all the volumes in that disk group. If so, the name of the cache that is declared must be the same for each volume’s space-optimized snapshot. The cache is stored on disk and is persistent.

If the cache approaches full, configure VxVM to grow the cache automatically using any available free space in the disk group.

Figure 16-8 shows the instant space-optimized snapshot model.

Figure 16-8 Space-optimized instant snapshot creation and usage in a backup cycle



See “Creating and managing space-optimized instant snapshots” on page 398.

Choices for snapshot resynchronization

When a snapshot volume is reattached to its original volume within a shared disk group, there are two choices for resynchronizing the data in the volume:

- Resynchronize the snapshot from the original volume—updates the snapshot with data from the primary volume that has changed since the snapshot was taken. The snapshot is then again ready to be taken for the purposes of backup or decision support. This type of resynchronization is also known as refreshing the snapshot.
- Resynchronize the original volume from the snapshot—updates the original volume with data from the snapshot volume that has changed since the snapshot was taken. This may be necessary to restore the state of a corrupted database or file system, or to implement upgrades to production software, and is usually much quicker than using alternative approaches such as full restoration from backup media. This type of resynchronization is also known as restoring the snapshot from the copy or replica.

Disk group split/join

One or more volumes, such as snapshot volumes, can be split off into a separate disk group and deported. They are then ready for importing on another host that is dedicated to off-host processing. This host need not be a member of a cluster but it must have access to the disks on which the volumes are configured. At a later stage, the disk group can be deported, re-imported, and joined with the original disk group, or with a different disk group.

Note: As space-optimized instant snapshots only record information about changed regions in the original volume, they cannot be moved to a different disk group. They are therefore unsuitable for the off-host processing applications that are described in this document.

The contents of full-sized instant snapshots must be fully synchronized with the unchanged regions in the original volume before such snapshots can be moved into a different disk group and deported from a host.

Storage Checkpoints

A Storage Checkpoint is a persistent image of a file system at a given instance in time. Storage Checkpoints use a copy-on-write technique to reduce I/O overhead by identifying and maintaining only those file system blocks that have changed since a previous Storage Checkpoint was taken. Storage Checkpoints have the following important features:

- Storage Checkpoints persist across system reboots and crashes.
- A Storage Checkpoint can preserve not only file system metadata and the directory hierarchy of the file system, but also user data as it existed when the Storage Checkpoint was taken.
- After creating a Storage Checkpoint of a mounted file system, you can continue to create, remove, and update files on the file system without affecting the image of the Storage Checkpoint.
- Unlike file system snapshots, Storage Checkpoints are writable.
- To minimize disk space usage, Storage Checkpoints use free space in the file system.

Storage Checkpoints and the Storage Rollback feature of Storage Foundation for Databases enable rapid recovery of databases from logical errors such as database corruption, missing files and dropped table spaces. You can mount successive Storage Checkpoints of a database to locate the error, and then roll back the database to a Storage Checkpoint before the problem occurred.

How Storage Checkpoints differ from snapshots

Storage Checkpoints differ from Veritas File System snapshots in the following ways because they:

- Allow write operations to the Storage Checkpoint itself.
- Persist after a system reboot or failure.

- Share the same pool of free space as the file system.
- Maintain a relationship with other Storage Checkpoints by identifying changed file blocks since the last Storage Checkpoint.
- Can have multiple, read-only Storage Checkpoints that reduce I/O operations and required storage space because the most recent Storage Checkpoint is the only one that accumulates updates from the primary file system.
- Can restore the file system to its state at the time that the Storage Checkpoint was taken.

Various backup and replication solutions can take advantage of Storage Checkpoints. The ability of Storage Checkpoints to track the file system blocks that have changed since the last Storage Checkpoint facilitates backup and replication applications that only need to retrieve the changed data. Storage Checkpoints significantly minimize data movement and may promote higher availability and data integrity by increasing the frequency of backup and replication solutions.

Storage Checkpoints can be taken in environments with a large number of files, such as file servers with millions of files, with little adverse impact on performance. Because the file system does not remain frozen during Storage Checkpoint creation, applications can access the file system even while the Storage Checkpoint is taken. However, Storage Checkpoint creation may take several minutes to complete depending on the number of files in the file system.

How a Storage Checkpoint works

The Storage Checkpoint facility freezes the mounted file system (known as the primary fileset), initializes the Storage Checkpoint, and thaws the file system. Specifically, the file system is first brought to a stable state where all of its data is written to disk, and the freezing process momentarily blocks all I/O operations to the file system. A Storage Checkpoint is then created without any actual data; the Storage Checkpoint instead points to the block map of the primary fileset. The thawing process that follows restarts I/O operations to the file system.

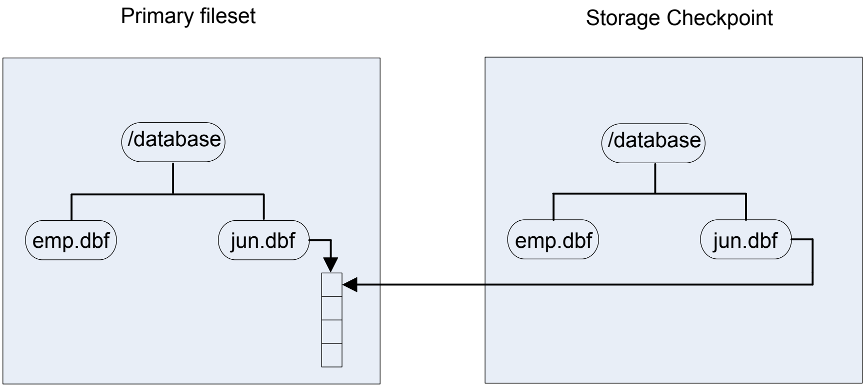
You can create a Storage Checkpoint on a single file system or a list of file systems. A Storage Checkpoint of multiple file systems simultaneously freezes the file systems, creates a Storage Checkpoint on all of the file systems, and thaws the file systems. As a result, the Storage Checkpoints for multiple file systems have the same creation timestamp. The Storage Checkpoint facility guarantees that multiple file system Storage Checkpoints are created on all or none of the specified file systems, unless there is a system crash while the operation is in progress.

Note: The calling application is responsible for cleaning up Storage Checkpoints after a system crash.

A Storage Checkpoint of the primary fileset initially contains only pointers to the existing data blocks in the primary fileset, and does not contain any allocated data blocks of its own.

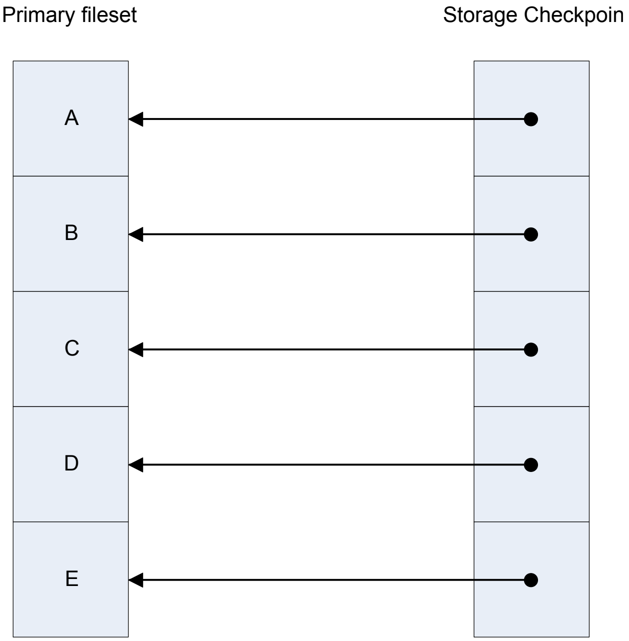
Figure 16-9 shows the file system `/database` and its Storage Checkpoint. The Storage Checkpoint is logically identical to the primary fileset when the Storage Checkpoint is created, but it does not contain any actual data blocks.

Figure 16-9 Primary fileset and its Storage Checkpoint



In Figure 16-10, a square represents each block of the file system. This figure shows a Storage Checkpoint containing pointers to the primary fileset at the time the Storage Checkpoint is taken, as in Figure 16-9.

Figure 16-10 Initializing a Storage Checkpoint



The Storage Checkpoint presents the exact image of the file system by finding the data from the primary fileset. VxFS updates a Storage Checkpoint by using the copy-on-write technique.

See “Copy-on-write” on page 368.

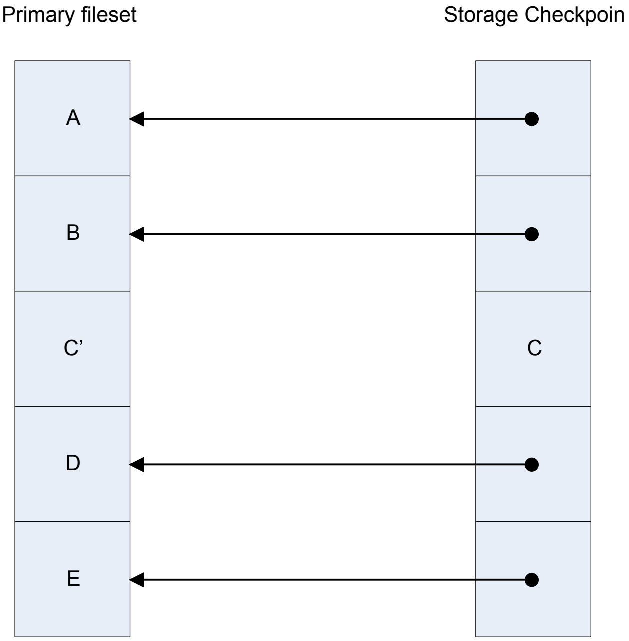
Copy-on-write

In Figure 16-11, the third data block in the primary fileset originally containing C is updated.

Before the data block is updated with new data, the original data is copied to the Storage Checkpoint. This is called the copy-on-write technique, which allows the Storage Checkpoint to preserve the image of the primary fileset when the Storage Checkpoint is taken.

Every update or write operation does not necessarily result in the process of copying data to the Storage Checkpoint because the old data needs to be saved only once. As blocks in the primary fileset continue to change, the Storage Checkpoint accumulates the original data blocks. In this example, subsequent updates to the third data block, now containing C', are not copied to the Storage Checkpoint because the original image of the block containing C is already saved.

Figure 16-11 Updates to the primary fileset



Storage Checkpoint visibility

With the `ckptautomnt` mount option, all Storage Checkpoints are made accessible automatically through a directory in the root directory of the file system that has the special name `.checkpoint`, which does not appear in directory listings. Inside this directory is a directory for each Storage Checkpoint in the file system. Each of these directories behave as a mount of the corresponding Storage Checkpoint, with the following exceptions:

- External applications, such as NFS, see the files as part of the original mount point. Thus, no additional NFS exports are necessary.
- Inode numbers exposed to applications can be made unique, depending on a mount option.

The Storage Checkpoints are automounted internally, but the operating system does not know about the automounting. This means that Storage Checkpoints cannot be mounted manually, and they do not apear in the list of mounted file systems. When Storage Checkpoints are created or deleted, entries in the Storage Checkpoint directory are automatically updated. If a Storage Checkpoint is removed with the `-f` option while a file in the Storage Checkpoint is still in use, the Storage Checkpoint is force unmounted, and all operations on the file fail with the EIO error.

If there is already a file or directory named `.checkpoint` in the root directory of the file system, such as a directory created with an older version of Veritas File System (VxFS) or when Storage Checkpoint visibility feature was disabled, the fake directory providing access to the Storage Checkpoints is not accessible. With this feature enabled, attempting to create a file or directory in the root directory with the name `.checkpoint` fails with the EEXIST error.

Note: If an auto-mounted Storage Checkpoint is in use by an NFS mount, removing the Storage Checkpoint might succeed even without the forced (`-f`) option.

Storage Checkpoints and 64-bit inode numbers

The inode number of a file is the same across Storage Checkpoints. For example, if the file `file1` exists in a file system and a Storage Checkpoint is taken of that file system, running the `stat` command on `file1` in the original file system and in the Storage Checkpoint returns the same value in `st_ino`. The combination of `st_ino` and `st_dev` should uniquely identify every file in a system. This is usually not a problem because Storage Checkpoints get mounted separately, so `st_dev` is different. When accessing files in a Storage Checkpoint through the Storage Checkpoint visibility extension, `st_dev` is the same for all Storage Checkpoints as well as for the original file system. This means files can no longer be identified uniquely by `st_ino` and `st_dev`.

In general, uniquely identifying all files in a system is not necessary. However, there can be some applications that rely on unique identification to function properly. For example, a backup application might check if a file is hard-linked to another file by calling `stat` on both and checking if `st_ino` and `st_dev` are the same. If a backup application were told to back up two clones through the Storage Checkpoint visibility extension at the same time, the application can erroneously deduce that two files are the same even though the files contain different data.

By default, Storage Foundation (SF) does not make inode numbers unique. However, you can specify the `uniqueino` mount option to enable the use of unique 64-bit inode numbers. You cannot change this option during a remount.

Types of Storage Checkpoints

You can create the following types of Storage Checkpoints:

- Data Storage Checkpoints
- Nodata Storage Checkpoints
- Removable Storage Checkpoints
- Non-mountable Storage Checkpoints

Data Storage Checkpoints

A data Storage Checkpoint is a complete image of the file system at the time the Storage Checkpoint is created. This type of Storage Checkpoint contains the file system metadata and file data blocks. You can mount, access, and write to a data Storage Checkpoint just as you would to a file system. Data Storage Checkpoints are useful for backup applications that require a consistent and stable image of an active file system. Data Storage Checkpoints introduce some overhead to the system and to the application performing the write operation. For best results, limit the life of data Storage Checkpoints to minimize the impact on system resources.

See “Showing the difference between a data and a nodata Storage Checkpoint” on page 441.

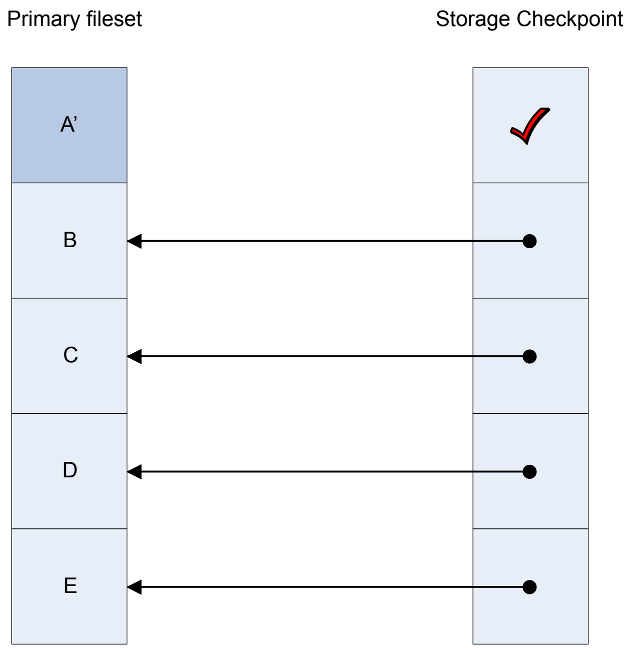
Nodata Storage Checkpoints

A nodata Storage Checkpoint only contains file system metadata—no file data blocks. As the original file system changes, the nodata Storage Checkpoint records the location of every changed block. Nodata Storage Checkpoints use minimal system resources and have little impact on the performance of the file system because the data itself does not have to be copied.

In Figure 16-12, the first block originally containing A is updated.

The original data is not copied to the Storage Checkpoint, but the changed block is marked in the Storage Checkpoint. The marker indicates which data has changed.

Figure 16-12 Updates to a nodata clone



See “Showing the difference between a data and a nodata Storage Checkpoint” on page 441.

Removable Storage Checkpoints

A removable Storage Checkpoint can self-destruct under certain conditions when the file system runs out of space.

See “Storage Checkpoint space management considerations” on page 449.

During user operations such as `create` or `mkdir`, if the file system runs out of space, removable Storage Checkpoints are deleted, even if the Storage Checkpoints are mounted. This ensures that applications can continue without interruptions due to lack of disk space. Non-removable Storage Checkpoints are not automatically removed under such `ENOSPC` conditions. Veritas recommends that you create only removable Storage Checkpoints. However, during certain administrative operations, such as `fsadm`, even if the file system runs out of space, removable Storage Checkpoints are not deleted.

Storage Checkpoints are created as non-removable by default. The default behavior can be changed so that VxFS creates removable Storage Checkpoints by using the `vxtunefs -D ckpt_removable=1` command. With the default set to create removable Storage Checkpoints, non-removable Storage Checkpoints can be created using `fsckptadm -R create` *`ckpt_name mount_point`* command.

See the `vxtunefs`(1M) and `fsckptadm`(1M) manual pages.

Non-mountable Storage Checkpoints

You can create Storage Checkpoints that cannot be mounted by using the `fsckptadm set nomount` command. The `nomount` option can be cleared using the `fsckptadm clear nomount` command.

Use non-mountable Storage Checkpoints as a security feature. This prevents other applications from accessing and modifying the Storage Checkpoint.

See the `fsckptadm`(1M) manual page.

About FileSnaps

A FileSnap is an atomic space-optimized copy of a file in the same name space, stored in the same file system. Veritas File System (VxFS) supports snapshots on file system disk layout Version 8 and later.

FileSnaps provide an ability to snapshot objects that are smaller in granularity than a file system or a volume. The ability to snapshot parts of a file system name space is required for application-based or user-based management of data stored in a file system. This is useful when a file system is shared by a set of users or applications or the data is classified into different levels of importance in the same file system.

All regular file operations are supported on the FileSnap, and VxFS does not distinguish the FileSnap in any way.

Properties of FileSnaps

FileSnaps provide non-root users the ability to snapshot data that they own, without requiring administrator privileges. This enables users and applications to version, backup, and restore their data by scheduling snapshots at appropriate points of their application cycle. Restoring from a FileSnap is as simple as specifying a snapshot as the source file and the original file as the destination file as the arguments for the `vxfilesnap` command.

FileSnap creation locks the source file as read-only and locks the destination file exclusively for the duration of the operation, thus creating the snapshots atomically. The rest of the files in the file system can be accessed with no I/O pause while FileSnap creation is in progress. Read access to the source file is also uninterrupted while the snapshot creation is in progress. This allows for true sharing of a file system by multiple users and applications in a non-intrusive fashion.

The name space relationship between source file and destination file is defined by the user-issued `vxfilesnap` command by specifying the destination file path. Veritas File System (VxFS) neither differentiates between the source file and the destination file, nor does it maintain any internal relationships between these two files. Once the snapshot is completed, the only shared property between the source file and destination file are the data blocks and block map shared by them.

The number of FileSnaps of a file is practically unlimited. The technical limit is the maximum number of files supported by the VxFS file system, which is one billion files per file set. When thousands of FileSnaps are created from the same file and each of these snapshot files is simultaneously read and written to by thousands of threads, FileSnaps scale very well due to the design that results in no contention of the shared blocks when unsharing happens due to an overwrite. The performance seen for the case of unsharing shared blocks due to an overwrite with FileSnaps is closer to that of an allocating write than that of a traditional copy-on-write.

In disk layout version 8 and later, to support block or extent sharing between the files, reference counts are tracked for each shared extent. VxFS processes reference count updates due to sharing and unsharing of extents in a delayed fashion. Also, an extent that is marked shared once will not go back to unshared until all the references are gone. This is to improve the FileSnap creation performance and performance of data extent unsharing. However, this in effect results in the shared block statistics for the file system to be only accurate to the point of the processing of delayed reclamation. In other words, the shared extent statistics on the file system and a file could be stale, depending on the state of the file system.

Concurrent I/O to FileSnaps

FileSnaps design and implementation ensures that concurrent reads or writes to different snapshots of the same file perform as if these were independent files. Even though the extents are shared between snapshots of the same file, the sharing has no negative impact on concurrent I/O.

Copy-on-write and FileSnaps

Veritas File System (VxFS) supports an option to do lazy copy-on-write when a region of a file referred to by a shared extent is overwritten. A typical copy-on-write implementation involves reading the old data, allocating a new block, copying or writing the old data to the new block synchronously, and writing the new data to the new block. This results in a worst case possibility of one or more allocating transactions, followed by a read, followed by a synchronous write and another write that conforms to the I/O behavior requested for the overwrite. This sequence makes typical copy-on-write a costly operation. The VxFS lazy copy-on-write implementation does not copy the old data to the newly allocated block and hence does not have to read the old data either, as long as the new data covers the entire block. This behavior combined with delayed processing of shared extent accounting makes the lazy copy-on-write complete in times comparable to that of an allocating write. However, in the event of a server crash, when the server has not flushed the new data to the newly allocated blocks, the data seen on the overwritten region would be similar to what you would find in the case of an allocating write where the server has crashed before the data is flushed. This is not the default behavior and with the default behavior the data that you find in the overwritten region will be either the new data or the old data.

Reading from FileSnaps

For regular read requests, Veritas File System (VxFS) only caches a single copy of a data page in the page cache for a given shared data block, even though the shared data block could be accessed from any of the FileSnaps or the source file. Once the shared data page is cached, any subsequent requests via any of the FileSnaps or the source file is serviced from the page cache. This eliminates duplicate read requests to the disk, which results in lower I/O load on the array. This also reduces the page cache duplication, which results in efficient usage of system page cache with very little cache churning when thousands of FileSnaps are accessed.

Block map fragmentation and FileSnaps

The block map of the source file is shared by the snapshot file. When data is overwritten on a previously shared region, the block map of the file to which the write happens gets changed. In cases where the shared data extent of a source file is larger than the size of the overwrite request to the same region, the block map of the file that is written to becomes more fragmented.

Backup and FileSnaps

A full backup of a VxFS file system that has shared blocks may require as much space in the target as the number of total logical references to the physical blocks in the source file system. For example, if you have a 20 GB file from which one thousand FileSnaps were created, the total number of logical block references is approximately 20 TB. While the VxFS file system only requires a little over 20 GB of physical blocks to store the file and the file's one thousand snapshots, the file system requires over 20 TB of space on the backup target to back up the file system, assuming the backup target does not have deduplication support.

About snapshot file systems

A snapshot file system is an exact image of a VxFS file system, referred to as the snapped file system, that provides a mechanism for making backups. The snapshot is a consistent view of the file system “snapped" at the point in time the snapshot is made. You can select files to back up from the snapshot using a standard utility such as `cpio` or `cp`, or back up the entire file system image using the `vxdump` or `fscat` utilities.

You use the mount command to create a snapshot file system; the `mkfs` command is not required. A snapshot file system is always read-only. A snapshot file system exists only as long as the snapped file system is mounted, and the snapshot file system ceases to exist when unmounted. A snapped file system cannot be unmounted until all of its snapshots are unmounted. Although it is possible to have multiple snapshots of a file system made at different times, it is not possible to make a snapshot of a snapshot.

Note: A snapshot file system ceases to exist when unmounted. If mounted again, it is actually a fresh snapshot of the snapped file system. A snapshot file system must be unmounted before its dependent snapped file system can be unmounted. Neither the `fuser` command nor the `mount` command will indicate that a snapped file system cannot be unmounted because a snapshot of it exists.

On cluster file systems, snapshots can be created on any node in the cluster, and backup operations can be performed from that node. The snapshot of a cluster file system is accessible only on the node where it is created, that is, the snapshot file system itself cannot be cluster mounted.

See the *Storage Foundation Cluster File System High Availability Administrator's Guide*.

How a snapshot file system works

A snapshot file system is created by mounting an empty disk slice as a snapshot of a currently mounted file system. The bitmap, blockmap and super-block are initialized and then the currently mounted file system is frozen. After the file system to be snapped is frozen, the snapshot is enabled and mounted and the snapped file system is thawed. The snapshot appears as an exact image of the snapped file system at the time the snapshot was made.

See “Freezing and thawing a file system” on page 336.

Initially, the snapshot file system satisfies read requests by finding the data on the snapped file system and returning it to the requesting process. When an inode update or a write changes the data in block n of the snapped file system, the old data is first read and copied to the snapshot before the snapped file system is updated. The bitmap entry for block n is changed from 0 to 1, indicating that the data for block n can be found on the snapshot file system. The blockmap entry for block n is changed from 0 to the block number on the snapshot file system containing the old data.

A subsequent read request for block n on the snapshot file system will be satisfied by checking the bitmap entry for block n and reading the data from the indicated block on the snapshot file system, instead of from block n on the snapped file system. This technique is called copy-on-write. Subsequent writes to block n on the snapped file system do not result in additional copies to the snapshot file system, since the old data only needs to be saved once.

All updates to the snapped file system for inodes, directories, data in files, extent maps, and so forth, are handled in this fashion so that the snapshot can present a consistent view of all file system structures on the snapped file system for the time when the snapshot was created. As data blocks are changed on the snapped file system, the snapshot gradually fills with data copied from the snapped file system.

The amount of disk space required for the snapshot depends on the rate of change of the snapped file system and the amount of time the snapshot is maintained. In the worst case, the snapped file system is completely full and every file is removed and rewritten. The snapshot file system would need enough blocks to hold a copy of every block on the snapped file system, plus additional blocks for the data structures that make up the snapshot file system. This is approximately 101 percent of the size of the snapped file system. Normally, most file systems do not undergo changes at this extreme rate. During periods of low activity, the snapshot should only require two to six percent of the blocks of the snapped file system. During periods of high activity, the snapshot might require 15 percent of the blocks of the snapped file system. These percentages tend to be lower for larger file systems and higher for smaller ones.

Warning: If a snapshot file system runs out of space for changed data blocks, it is disabled and all further attempts to access it fails. This does not affect the snapped file system.

Chapter 17

Administering volume snapshots

This chapter includes the following topics:

About volume snapshots

VxVM can take an image of a volume at a given point in time. This image is called a volume snapshot.

See “Volume-level snapshots” on page 361.

You can also take a snapshot of a volume set.

Snapshot creation using the `vxsnap` command is the preferred mechanism for implementing point-in-time copy solutions in VxVM. Support for traditional third-mirror snapshots that are created using the `vxassist` command may be removed in a future release.

To recover from the failure of instant snapshot commands, see the *Veritas InfoScale Troubleshooting Guide*.

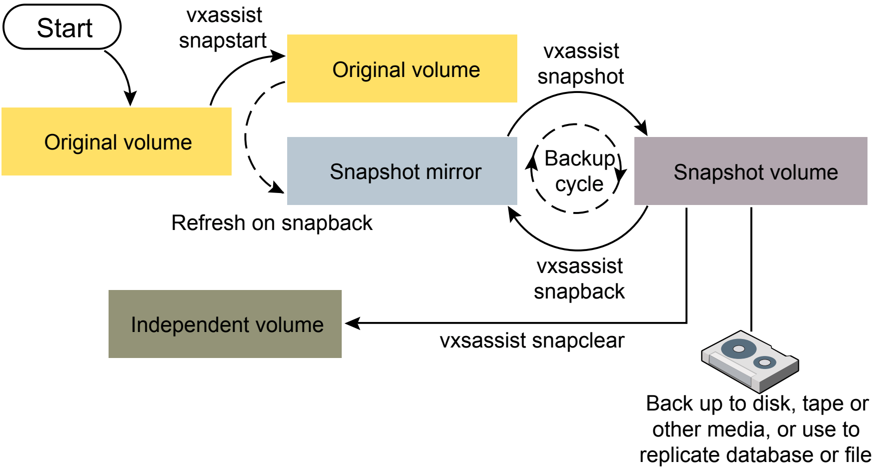
Traditional third-mirror break-off snapshots

The recommended approach to performing volume backup from the command line, or from a script, is to use the `vxsnap` command. The `vxassist snapstart`, `snapwait`, and `snapshot` commands are supported for backward compatibility.

The use of the `vxassist` command to administer traditional (third-mirror break-off) snapshots is not supported for volumes that are prepared for instant snapshot creation. Use the `vxsnap` command instead.

Figure 17-1 shows the traditional third-mirror break-off volume snapshot model that is supported by the `vxassist` command.

Figure 17-1 Third-mirror snapshot creation and usage



The `vxassist snapstart` command creates a mirror to be used for the snapshot, and attaches it to the volume as a snapshot mirror. As is usual when creating a mirror, the process of copying the volume’s contents to the new snapshot plexes can take some time to complete. (The `vxassist snapabort` command cancels this operation and removes the snapshot mirror.)

When the attachment is complete, the `vxassist snapshot` command is used to create a new snapshot volume by taking one or more snapshot mirrors to use as its data plexes. The snapshot volume contains a copy of the original volume’s data at the time that you took the snapshot. If more than one snapshot mirror is used, the snapshot volume is itself mirrored.

The command, `vxassist snapback`, can be used to return snapshot plexes to the original volume from which they were snapped, and to resynchronize the data in the snapshot mirrors from the data in the original volume. This enables you to refresh the data in a snapshot after you use it to make a backup. You can use a variation of the same command to restore the contents of the original volume from a snapshot previously taken.

The FastResync feature minimizes the time and I/O needed to resynchronize the data in the snapshot. If FastResync is not enabled, a full resynchronization of the data is required.

Finally, you can use the `vxassist snapclear` command to break the association between the original volume and the snapshot volume. Because the snapshot relationship is broken, no change tracking occurs. Use this command if you do not need to reuse the snapshot volume to create a new point-in-time copy.

Creating traditional third-mirror break-off snapshots

VxVM provides third-mirror break-off snapshot images of volume devices using `vxassist` and other commands.

To enhance the efficiency and usability of volume snapshots, turn on FastResync.

If Persistent FastResync is required, you must associate a version 0 DCO with the volume.

See "Adding a version 0 DCO and DCO volume" on page 430.

A plex is required that is large enough to store the complete contents of the volume. Alternatively, you can use space-optimized instant snapshots.

The recommended approach to performing volume backup from the command line, or from a script, is to use the `vxsnap` command. The `vxassist snapstart`, `snapwait`, and `snapshot` commands are supported for backward compatibility.

The `vxassist snapshot` procedure consists of two steps:

- Run `vxassist snapstart` to create a snapshot mirror.
- Run `vxassist snapshot` to create a snapshot volume.

The `vxassist snapstart` step creates a write-only backup plex which gets attached to and synchronized with the volume.

If the volume is encrypted with a password or passphrase, you will prompted to enter the password or passphrase.

When synchronized with the volume, the backup plex is ready to be used as a `snapshot` mirror. The end of the update procedure is indicated by the new `snapshot` mirror changing its state to SNAPDONE. This change can be tracked by the `vxassist snapwait` task, which waits until at least one of the mirrors changes its state to SNAPDONE. If the attach process fails, the `snapshot` mirror is removed and its space is released.

Note: If the `snapstart` procedure is interrupted, the snapshot mirror is automatically removed when the volume is started.

Once the `snapshot` mirror is synchronized, it continues being updated until it is detached. You can then select a convenient time at which to create a `snapshot` volume as an image of the existing volume. You can also ask users to refrain from using the system during the brief time required to perform the `snapshot` (typically less than a minute). The amount of time involved in creating the `snapshot` mirror is long in contrast to the brief amount of time that it takes to create the `snapshot` volume.

The online backup procedure is completed by running the `vxassist snapshot` command on a volume with a SNAPDONE mirror. This task detaches the finished `snapshot` (which becomes a normal mirror), creates a new normal volume and attaches the `snapshot` mirror to the `snapshot` volume. The `snapshot` then becomes a normal, functioning volume and the state of the `snapshot` is set to ACTIVE.

To back up a volume using the vxassist command

1. Create a snapshot mirror for a volume using the following command:

   ```
   # vxassist [-b] [-g diskgroup] snapstart [nmirror=N] volume
   ```

   For example, to create a snapshot mirror of a volume called `voldef`, use the following command:

   ```
   # vxassist [-g diskgroup] snapstart voldef
   ```

   The `vxassist snapstart` task creates a write-only mirror, which is attached to and synchronized from the volume to be backed up.

   By default, VxVM attempts to avoid placing snapshot mirrors on a disk that already holds any plexes of a data volume. However, this may be impossible if insufficient space is available in the disk group. In this case, VxVM uses any available space on other disks in the disk group. If the snapshot plexes are placed on disks which are used to hold the plexes of other volumes, this may cause problems when you subsequently attempt to move a snapshot volume into another disk group.

   See "Moving DCO volumes between disk groups" on page 648.

   To override the default storage allocation policy, you can use storage attributes to specify explicitly which disks to use for the snapshot plexes.

   See "Creating a volume on specific disks" on page 151.

   If you start `vxassist snapstart` in the background using the `-b` option, you can use the `vxassist snapwait` command to wait for the creation of the mirror to complete as shown here:

   ```
   # vxassist [-g diskgroup] snapwait volume
   ```

   If `vxassist snapstart` is not run in the background, it does not exit until the mirror has been synchronized with the volume. The mirror is then ready to be used as a plex of a snapshot volume. While attached to the original volume, its contents continue to be updated until you take the snapshot.

   Use the `nmirror` attribute to create as many snapshot mirrors as you need for the snapshot volume. For a backup, you should usually only require the default of one.

   It is also possible to make a snapshot plex from an existing plex in a volume.

   See "Converting a plex into a snapshot plex" on page 384.

2. Choose a suitable time to create a snapshot. If possible, plan to take the snapshot at a time when users are accessing the volume as little as possible.

3. Create a snapshot volume using the following command:

   ```
   # vxassist [-g diskgroup] snapshot [nmirror=N] volume snapshot
   ```

   If required, use the `nmirror` attribute to specify the number of mirrors in the snapshot volume.

   For example, to create a snapshot of `voldef`, use the following command:

   ```
   # vxassist -g mydg snapshot voldef snapvoldef
   ```

   The `vxassist snapshot` task detaches the finished snapshot mirror, creates a new volume, and attaches the snapshot mirror to it. This step should only take a few minutes. The snapshot volume, which reflects the original volume at the time of the snapshot, is now available for backing up, while the original volume continues to be available for applications and users.

   If required, you can make snapshot volumes for several volumes in a disk group at the same time.

   See “Creating multiple snapshots with the vxassist command” on page 385.

4. Clean the temporary volume's contents using an appropriate utility such as `fsck` for non-VxVM file systems and log replay for databases. Because VxVM calls VxFS and places VxFS file systems in a constant state immediately before taking a snapshot, it is not usually necessary to run `fsck` on a VxFS file system on the temporary volume. If a VxFS file system contains a database, it will still be necessary to perform database log replay.

5. If you require a backup of the data in the snapshot, use an appropriate utility or operating system command to copy the contents of the snapshot to tape, or to some other backup medium.

6. When the backup is complete, you have the following choices for what to do with the snapshot volume:

   - Reattach some or all of the plexes of the snapshot volume with the original volume.
     See “Reattaching a snapshot volume” on page 386.

   - If FastResync was enabled on the volume before the snapshot was taken, this speeds resynchronization of the snapshot plexes before the backup cycle starts again at step 3.

   - Dissociate the snapshot volume entirely from the original volume
     See “Dissociating a snapshot volume” on page 387.

   - This may be useful if you want to use the copy for other purposes such as testing or report generation.

- Remove the snapshot volume to save space with this command:

  ```
  # vxedit [-g diskgroup] -rf rm snapshot
  ```

  Dissociating or removing the snapshot volume loses the advantage of fast resynchronization if FastResync was enabled. If there are no further snapshot plexes available, any subsequent snapshots that you take require another complete copy of the original volume to be made.

Converting a plex into a snapshot plex

For a traditional, third-mirror break-off snapshot, you can convert an existing plex in a volume into a snapshot plex. Veritas recommends using the instant snapshot feature rather than converting a plex into a snapshot plex.

Note: A plex cannot be converted into a snapshot plex for layered volumes or for any volume that has an associated instant snap DCO volume.

In some circumstances, you may find it more convenient to convert an existing plex in a volume into a snapshot plex rather than running `vxassist snapstart`. For example, you may want to do this if you are short of disk space for creating the snapshot plex and the volume that you want to snapshot contains more than two plexes.

The procedure can also be used to speed up the creation of a snapshot volume when a mirrored volume is created with more than two plexes and `init=active` is specified.

It is advisable to retain at least two plexes in a volume to maintain data redundancy.

To convert an existing plex into a snapshot plex for a volume on which Persistent FastResync is enabled, use the following command:

```
# vxplex [-g diskgroup] -o dcoplex=dcologplex convert \
  state=SNAPDONE plex
```

dcologplex is the name of an existing DCO plex that is to be associated with the new snapshot plex. You can use the `vxprint` command to find out the name of the DCO volume.

See "Adding a version 0 DCO and DCO volume" on page 430.

For example, to make a snapshot plex from the plex `trivol-03` in the 3-plex volume `trivol`, you would use the following command:

```
# vxplex -o dcoplex=trivol_dco-03 convert state=SNAPDONE \
  trivol-03
```

Here the DCO plex `trivol_dco_03` is specified as the DCO plex for the new snapshot plex.

To convert an existing plex into a snapshot plex in the SNAPDONE state for a volume on which Non-Persistent FastResync is enabled, use the following command:

```
# vxplex [-g diskgroup] convert state=SNAPDONE plex
```

A converted plex is in the SNAPDONE state, and can be used immediately to create a snapshot volume.

Note: The last complete regular plex in a volume, an incomplete regular plex, or a dirty region logging (DRL) log plex cannot be converted into a snapshot plex.

See “Third-mirror break-off snapshots” on page 362.

Creating multiple snapshots with the vxassist command

To make it easier to create snapshots of several volumes at the same time, the snapshot option accepts more than one volume name as its argument, for example:

```
# vxassist [-g diskgroup] snapshot volume1
    volume2 ...
```

By default, the first snapshot volume is named *SNAP-volume*, and each subsequent snapshot is named *SNAPnumber-volume*, where *number* is a unique serial number, and *volume* is the name of the volume for which the snapshot is being taken. This default pattern can be overridden by using the option `-o name=pattern`, as described on the `vxassist`(1M) manual page. For example, the pattern `SNAP%v-%d` reverses the order of the *number* and *volume* components in the name.

To snapshot all the volumes in a single disk group, specify the option `-o allvols` to `vxassist:`

```
# vxassist -g diskgroup -o allvols snapshot
```

This operation requires that all `snapstart` operations are complete on the volumes. It fails if any of the volumes in the disk group do not have a complete snapshot plex in the SNAPDONE state.

Note: The `vxsnap` command provides similiar functionality for creating multiple snapshots.

Reattaching a snapshot volume

The snapback operation merges a snapshot copy of a volume with the original volume. One or more snapshot plexes are detached from the snapshot volume and re-attached to the original volume. The snapshot volume is removed if all its snapshot plexes are snapped back. This task resynchronizes the data in the volume so that the plexes are consistent.

The snapback operation cannot be applied to RAID-5 volumes unless they have been converted to a special layered volume layout by the addition of a DCO and DCO volume.

See “Adding a version 0 DCO and DCO volume” on page 430.

To enhance the efficiency of the snapback operation, enable FastResync on the volume before taking the snapshot

To merge one snapshot plex with the original volume, use the following command:

```
# vxassist [-g diskgroup] snapback snapshot
```

where *snapshot* is the snapshot copy of the volume.

To merge all snapshot plexes in the snapshot volume with the original volume, use the following command:

```
# vxassist [-g diskgroup] -o allplexes snapback snapshot
```

To merge a specified number of plexes from the snapshot volume with the original volume, use the following command:

```
# vxassist [-g diskgroup] snapback nmirror=number snapshot
```

Here the `nmirror` attribute specifies the number of mirrors in the snapshot volume that are to be re-attached.

Once the snapshot plexes have been reattached and their data resynchronized, they are ready to be used in another `snapshot` operation.

By default, the data in the original volume is used to update the snapshot plexes that have been re-attached. To copy the data from the replica volume instead, use the following command:

```
# vxassist [-g diskgroup] -o resyncfromreplica snapback snapshot
```

Warning: Always unmount the snapshot volume (if this is mounted) before performing a snapback. In addition, you must unmount the file system corresponding to the primary volume before using the `resyncfromreplica` option.

Adding plexes to a snapshot volume

If you want to retain the existing plexes in a snapshot volume after a snapback operation, you can create additional snapshot plexes that are to be used for the snapback.

To add plexes to a snapshot volume

1. Use the following vxprint commands to discover the names of the snapshot volume's data change object (DCO) and DCO volume:

   ```
   # DCONAME=`vxprint [-g diskgroup] -F%dco_name snapshot`
   # DCOVOL=`vxprint [-g diskgroup] -F%log_vol $DCONAME`
   ```

2. Use the vxassist mirror command to create mirrors of the existing snapshot volume and its DCO volume:

   ```
   # vxassist -g diskgroup mirror snapshot
   # vxassist -g diskgroup mirror $DCOVOL
   ```

   The new plex in the DCO volume is required for use with the new data plex in the snapshot.

3. Use the vxprint command to find out the name of the additional snapshot plex:

   ```
   # vxprint -g diskgroup snapshot
   ```

4. Use the vxprint command to find out the record ID of the additional DCO plex:

   ```
   # vxprint -g diskgroup -F%rid $DCOVOL
   ```

5. Use the vxedit command to set the dco_plex_rid field of the new data plex to the name of the new DCO plex:

   ```
   # vxedit -g diskgroup set dco_plex_rid=dco_plex_rid new_plex
   ```

   The new data plex is now ready to be used to perform a snapback operation.

Dissociating a snapshot volume

The link between a snapshot and its original volume can be permanently broken so that the snapshot volume becomes an independent volume. Use the following command to dissociate the snapshot volume, *snapshot*:

```
# vxassist snapclear snapshot
```

Displaying snapshot information

The `vxassist snapprint`command displays the associations between the original volumes and their respective replicas (snapshot copies):

```
# vxassist snapprint [volume]
```

Output from this command is shown in the following examples:

```
# vxassist -g mydg snapprint v1

V  NAME          USETYPE      LENGTH
SS SNAPOBJ       NAME         LENGTH     %DIRTY
DP NAME          VOLUME       LENGTH     %DIRTY

v  v1            fsgen        20480
ss SNAP-v1_snp   SNAP-v1      20480      4
dp v1-01         v1           20480      0
dp v1-02         v1           20480      0

v  SNAP-v1       fsgen        20480
ss v1_snp        v1           20480      0
```

```
# vxassist -g mydg snapprint v2

V  NAME          USETYPE      LENGTH
SS SNAPOBJ       NAME         LENGTH     %DIRTY
DP NAME          VOLUME       LENGTH     %DIRTY

v  v2            fsgen        20480
ss --            SNAP-v2      20480      0
dp v2-01         v2           20480      0

v  SNAP-v2       fsgen        20480
ss --            v2           20480      0
```

In this example, Persistent FastResync is enabled on volume `v1`, and Non-Persistent FastResync on volume `v2`. Lines beginning with `v`, `dp` and `ss` indicate a volume, detached plex and snapshot plex respectively. The `%DIRTY` field indicates the percentage of a snapshot plex or detached plex that is dirty with respect to the original volume. Notice that no snap objects are associated with volume `v2` or with its snapshot volume `SNAP-v2`.

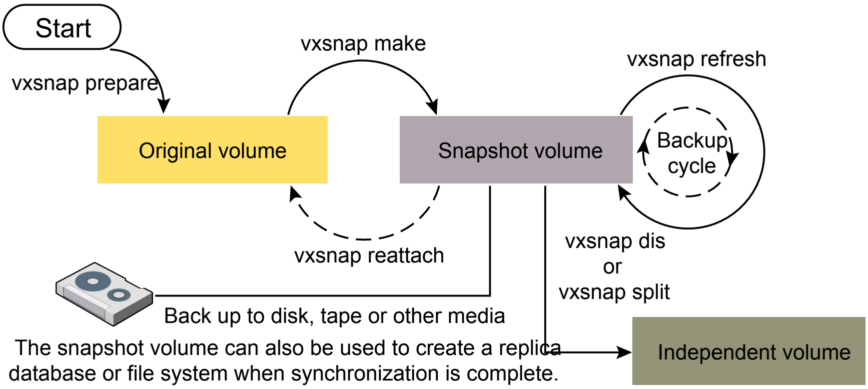
If a volume is specified, the `snapprint` command displays an error message if no FastResync maps are enabled for that volume.

Full-sized instant snapshots

Full-sized instant snapshots are a variation on the third-mirror volume snapshot model that make a snapshot volume available for I/O access as soon as the snapshot plexes have been created.

Figure 17-2 shows the full-sized instant volume snapshot model.

Figure 17-2 Full-sized instant snapshot creation and usage in a backup cycle


Start
vxsnap prepare
vxsnap make
vxsnap refresh
Original volume
Snapshot volume
Backup cycle
vxsnap reattach
vxsnap dis or vxsnap split
Back up to disk, tape or other media
The snapshot volume can also be used to create a replica database or file system when synchronization is complete.
Independent volume

To create an instant snapshot, use the `vxsnap make` command. This command can either be applied to a suitably prepared empty volume that is to be used as the snapshot volume, or it can be used to break off one or more synchronized plexes from the original volume.

You can make a backup of a full-sized instant snapshot, instantly refresh its contents from the original volume, or attach its plexes to the original volume, without completely synchronizing the snapshot plexes from the original volume.

VxVM uses a copy-on-write mechanism to ensure that the snapshot volume preserves the contents of the original volume at the time that the snapshot is taken. Any time that the original contents of the volume are about to be overwritten, the original data in the volume is moved to the snapshot volume before the write proceeds. As time goes by, and the contents of the volume are updated, its original contents are gradually relocated to the snapshot volume.

If a read request comes to the snapshot volume, yet the data resides on the original volume (because it has not yet been changed), VxVM automatically and transparently reads the data from the original volume.

If desired, you can perform either a background (non-blocking) or foreground (blocking) synchronization of the snapshot volume. This is useful if you intend to move the snapshot volume into a separate disk group for off-host processing, or you want to turn the snapshot volume into an independent volume.

The `vxsnap refresh` command allows you to update the data in a snapshot, for example, before taking a backup.

The command `vxsnap reattach` attaches snapshot plexes to the original volume, and resynchronizes the data in these plexes from the original volume. Alternatively, you can use the `vxsnap restore` command to restore the contents of the original volume from a snapshot that you took at an earlier point in time. You can also choose whether or not to keep the snapshot volume after restoration of the original volume is complete.

By default, the FastResync feature of VxVM is used to minimize the time and I/O needed to resynchronize the data in the snapshot mirror. FastResync must be enabled to create instant snapshots.

See "Creating and managing full-sized instant snapshots" on page 401.

An empty volume must be prepared for use by full-sized instant snapshots and linked break-off snapshots.

See "Creating a volume for use as a full-sized instant or linked break-off snapshot" on page 396.

Creating instant snapshots

Note: You need a Veritas InfoScale Storage or Veritas InfoScale Enterprise license to use this feature.

VxVM allows you to make instant snapshots by using the `vxsnap` command.

You can also take instant snapshots of RAID-5 volumes that have been converted to a special layered volume layout by the addition of a DCO and DCO volume.

A plex in a full-sized instant snapshot requires as much space as the original volume. If you instead make a space-optimized instant snapshot of a volume, this only requires enough storage to record the original contents of the parent volume as they are changed during the life of the snapshot.

The recommended approach to perform volume backup from the command line, or from a script, is to use the `vxsnap` command. The `vxsnap prepare` and `make` tasks allow you to back up volumes online with minimal disruption to users.

`vxsnap prepare` creates a DCO and DCO volume and associates this with the original volume. It also enables Persistent FastResync.

vxsnap make creates an instant snapshot that is immediately available for making a backup. After the snapshot has been taken, read requests for data in the instant snapshot volume are satisfied by reading either from a non-updated region of the original volume, or from the copy of the original contents of an updated region that have been recorded by the snapshot.

Note: Synchronization of a full-sized instant snapshot from the original volume is enabled by default. If you specify the syncing=no attribute to vxsnap make, this disables synchronization, and the contents of the instant snapshot are unlikely ever to become fully synchronized with the contents of the original volume at the point in time that the snapshot was taken. In such a case, the snapshot cannot be used for off-host processing, nor can it become an independent volume.

You can immediately retake a full-sized or space-optimized instant snapshot at any time by using the vxsnap refresh command. If a fully synchronized instant snapshot is required, the new resynchronization must first complete.

To create instant snapshots of volume sets, use volume set names in place of volume names in the vxsnap command.

See "Creating instant snapshots of volume sets" on page 409.

When using the vxsnap prepare or vxassist make commands to make a volume ready for instant snapshot operations, if the specified region size exceeds half the value of the tunable voliomem_maxpool_sz , the operation succeeds but gives a warning such as the following (for a system where voliomem_maxpool_sz is set to 12MB):

```
VxVM vxassist WARNING V-5-1-0 Specified regionsize is
larger than the limit on the system
(voliomem_maxpool_sz/2=6144k).
```

If this message is displayed, vxsnap make, refresh and restore operations on such volumes fail as they might potentially hang the system. Such volumes can be used only for break-off snapshot operations using the reattach and make operations.

To make the volumes usable for instant snapshot operations, use vxsnap unprepare on the volume, and then use vxsnap prepare to re-prepare the volume with a region size that is less than half the size of voliomem_maxpool_sz (in this example, 1MB):

```
# vxsnap -g mydg -f unprepare vol1
# vxsnap -g mydg prepare vol1 regionsize=1M
```

See "Creating instant snapshots of volume sets" on page 409.

See "Creating and managing space-optimized instant snapshots" on page 398.

See "Creating and managing full-sized instant snapshots" on page 401.

See "Creating and managing third-mirror break-off snapshots" on page 403.

See "Creating and managing linked break-off snapshot volumes" on page 406.

Adding an instant snap DCO and DCO volume

To prepare a volume for instant snapshots, an instant snap data change object (DCO) and DCO volume must be associated with that volume. This procedure also enables Persistent FastResync on the volume.

The following procedure is required only if the volume does not have an instant snap DCO volume.

By default, volumes on thin provisioning LUNs are created with an instant snap DCO volume.

To add an instant snap DCO and DCO volume

1. Verify that the volume has an instant snap data change object (DCO) and DCO volume, and that FastResync is enabled on the volume:

   ```
   # vxprint -g volumedg -F%instant volume
   # vxprint -g volumedg -F%fastresync volume
   ```

   If both commands return a value of `on`, skip to step 3. Otherwise continue with step 2.

2. To prepare a volume for instant snapshots, use the following command:

   ```
   # vxsnap [-g diskgroup] prepare volume [regionsize=size] \
     [ndcomirs=number] [alloc=storage_attributes]
   ```

   Run the `vxsnap prepare` command on a volume only if it does not have an instant snap DCO volume.

   For example, to prepare the volume, `myvol`, in the disk group, `mydg`, use the following command:

   ```
   # vxsnap -g mydg prepare myvol regionsize=128k ndcomirs=2 \
     alloc=mydg10,mydg11
   ```

   This example creates a DCO object and redundant DCO volume with two plexes located on disks `mydg10` and `mydg11`, and associates them with `myvol`. The region size is also increased to 128KB from the default size of 64KB. The region size must be a power of 2, and be greater than or equal to 16KB. A smaller value requires more disk space for the change maps, but the finer granularity provides faster resynchronization.

3. If you need several space-optimized instant snapshots for the volumes in a disk group, you may find it convenient to create a single shared cache object in the disk group rather than a separate cache object for each snapshot.

   See “Creating a shared cache object” on page 394.

   For full-sized instant snapshots and linked break-off snapshots, you must prepare a volume that is to be used as the snapshot volume. This volume must be the same size as the data volume for which the snapshot is being created, and it must also have the same region size.

   See “Creating a volume for use as a full-sized instant or linked break-off snapshot” on page 396.

Creating a shared cache object

If you need several space-optimized instant snapshots for the volumes in a disk group, you can create a single shared cache object in the disk group rather than a separate cache object for each snapshot.

To create a shared cache object

1. Decide on the following characteristics that you want to allocate to the cache volume that underlies the cache object:

   - The cache volume size should be sufficient to record changes to the parent volumes during the interval between snapshot refreshes. A suggested value is 10% of the total size of the parent volumes for a refresh interval of 24 hours.
   - The cache volume can be mirrored for redundancy.
   - If the cache volume is mirrored, space is required on at least as many disks as it has mirrors. These disks should not be shared with the disks used for the parent volumes. The disks should not be shared with disks used by critical volumes to avoid impacting I/O performance for critical volumes, or hindering disk group split and join operations.

2. Having decided on its characteristics, use the `vxassist` command to create the cache volume. The following example creates a mirrored cache volume, `cachevol`, with size 1GB in the disk group, `mydg`, on the disks `mydg16` and `mydg17`:

   ```
   # vxassist -g mydg make cachevol 1g layout=mirror \
     init=active mydg16 mydg17
   ```

   The attribute `init=active` makes the cache volume immediately available for use.

3. Use the `vxmake cache` command to create a cache object on top of the cache volume that you created in the previous step:

   ```
   # vxmake [-g diskgroup] cache cache_object \
     cachevolname=volume [regionsize=size] [autogrow=on] \
     [highwatermark=hwmk] [autogrowby=agbvalue] \
     [maxautogrow=maxagbvalue]]
   ```

   If the region size, `regionsize`, is specified, it must be a power of 2, and be greater than or equal to 16KB (`16k`). If not specified, the region size of the cache is set to 64KB.

   All space-optimized snapshots that share the cache must have a region size that is equal to or an integer multiple of the region size set on the cache. Snapshot creation also fails if the original volume’s region size is smaller than the cache’s region size.

   If the region size of a space-optimized snapshot differs from the region size of the cache, this can degrade the system’s performance compared to the case where the region sizes are the same.

   To prevent the cache from growing automatically, specify `autogrow=off`. By default, `autogrow=on`.

   In the following example, the cache object, `cobjmydg`, is created over the cache volume, `cachevol`, the region size of the cache is set to 32KB, and the `autogrow` feature is enabled:

   ```
   # vxmake -g mydg cache cobjmydg cachevolname=cachevol \
     regionsize=32k autogrow=on
   ```

4. Enable the cache object using the following command:

   ```
   # vxcache [-g diskgroup] start cache_object
   ```

   For example to start the cache object, `cobjmydg`:

   ```
   # vxcache -g mydg start cobjmydg
   ```

   See “Removing a cache” on page 423.

Creating a volume for use as a full-sized instant or linked break-off snapshot

To create an empty volume for use by a full-sized instant snapshot or a linked break-off snapshot

1 Use the `vxprint` command on the original volume to find the required size for the snapshot volume.

```
# LEN=`vxprint [-g diskgroup] -F%len volume`
```

The command as shown assumes a Bourne-type shell such as `sh`, `ksh` or `bash`. You may need to modify the command for other shells such as `csh` or `tcsh`.

2 Use the `vxprint` command on the original volume to discover the name of its DCO:

```
# DCONAME=`vxprint [-g diskgroup] -F%dco_name volume`
```

3 Use the vxprint command on the DCO to discover its region size (in blocks):

```
# RSZ=`vxprint [-g diskgroup] -F%regionsz $DCONAME`
```

4 Use the vxassist command to create a volume, *snapvol*, of the required size and redundancy, together with an instant snap DCO volume with the correct region size:

```
# vxassist [-g diskgroup] make snapvol $LEN \
  [layout=mirror nmirror=number] logtype=dco drl=off \
  dcoversion=30 [ndcomirror=number] regionsz=$RSZ \
  init=active [storage_attributes]
```

Storage attributes give you control over the devices, including disks and controllers, which vxassist uses to configure a volume.

See "Creating a volume on specific disks" on page 151.

Specify the same number of DCO mirrors (ndcomirror) as the number of mirrors in the volume (nmirror). The init=active attribute makes the volume available immediately. You can use storage attributes to specify which disks should be used for the volume.

As an alternative to creating the snapshot volume and its DCO volume in a single step, you can first create the volume, and then prepare it for instant snapshot operations as shown here:

```
# vxassist [-g diskgroup] make snapvol $LEN \
  [layout=mirror nmirror=number] init=active \
  [storage_attributes]
# vxsnap [-g diskgroup] prepare snapvol [ndcomirs=number] \
  regionsize=$RSZ [storage_attributes]
```

Upgrading the instant snap Data Change Objects (DCOs) and DCO volumes for a VxVM volume

Instant snap DCOs, formerly known as version 30 DCOs, support the creation of instant snapshots for VxVM volumes. Upgrade the instant snap DCOS and DCO volumes to ensure compatability with the latest version of VxVM. The upgrade operation can be performed while the volumes are online.

The upgrade operation does not support upgrade from version 0 DCOs.

To upgrade the instant snap DCOs for all volumes in the disk group

1. Make sure that the disk group is at least version 180. To upgrade the disk group:

   ```
   # vxdg upgrade diskgroup
   ```

2. Use the following command to upgrade the instant snap DCOs for all volumes in the disk group:

   ```
   # vxsnap -g diskgroup upgradeall
   ```

   Where:*diskgroup* is the disk group that contains the volumes to be upgraded.

   For additional options to the upgradeall operation, see the `vxsnap`(1M) manual page.

To upgrade the instant snap DCOs for specified volumes

1. Make sure that the disk group is at least version 180. To upgrade the disk group:

   ```
   # vxdg upgrade diskgroup
   ```

2. To upgrade the DCOs, specify one or more volumes or volume sets to the following command:

   ```
   # vxsnap [-g diskgroup] upgrade
   [volume1|volset1][volume2|volset2...]
   ```

   Where:*diskgroup* is the disk group that contains the volumes to be upgraded.

   For additional options to the upgrade operation, see the `vxsnap`(1M) manual page.

Creating and managing space-optimized instant snapshots

Space-optimized instant snapshots are not suitable for write-intensive volumes (such as for database redo logs) because the copy-on-write mechanism may degrade performance.

To split the volume and snapshot into separate disk groups (for example, to perform off-host processing), you must use a fully synchronized full-sized instant, third-mirror break-off or linked break-off snapshot (which do not require a cache object). You cannot use a space-optimized instant snapshot.

Creation of space-optimized snapshots that use a shared cache fails if the region size specified for the volume is smaller than the region size set on the cache.

If the region size of a space-optimized snapshot differs from the region size of the cache, this can degrade the system's performance compared to the case where the region sizes are the same.

See "Creating a shared cache object" on page 394.

The attributes for a snapshot are specified as a tuple to the `vxsnap make` command. This command accepts multiple tuples. One tuple is required for each snapshot that is being created. Each element of a tuple is separated from the next by a slash character (`/`). Tuples are separated by white space.

To create and manage a space-optimized instant snapshot

1. Use the `vxsnap make` command to create a space-optimized instant snapshot. This snapshot can be created by using an existing cache object in the disk group, or a new cache object can be created.

   - To create a space-optimized instant snapshot, *snapvol*, that uses a named shared cache object:

     ```
     # vxsnap [-g diskgroup] make source=vol/newvol=snapvol\
     /cache=cacheobject [alloc=storage_attributes]
     ```

     For example, to create the space-optimized instant snapshot, `snap3myvol`, of the volume, `myvol`, in the disk group, `mydg`, on the disk `mydg14`, and which uses the shared cache object, `cobjmydg`, use the following command:

     ```
     # vxsnap -g mydg make source=myvol/newvol=snap3myvol\
     /cache=cobjmydg alloc=mydg14
     ```

     The DCO is created on the specified allocation.

   - To create a space-optimized instant snapshot, *snapvol*, and also create a cache object for it to use:

     ```
     # vxsnap [-g diskgroup] make source=vol/newvol=snapvol\
     [/cachesize=size][/autogrow=yes][/ncachemirror=number]\
       [alloc=storage_attributes]
     ```

     The `cachesize` attribute determines the size of the cache relative to the size of the volume. The `autogrow` attribute determines whether VxVM will automatically enlarge the cache if it is in danger of overflowing. By default, `autogrow=on` and the cache is automatically grown.

     If `autogrow` is enabled, but the cache cannot be grown, VxVM disables the oldest and largest snapshot that is using the same cache, and releases its cache space for use.

The `ncachemirror` attribute specifies the number of mirrors to create in the cache volume. For backup purposes, the default value of 1 should be sufficient.
For example, to create the space-optimized instant snapshot, `snap4myvol`, of the volume, `myvol`, in the disk group, `mydg`, on the disk `mydg15`, and which uses a newly allocated cache object that is 1GB in size, but which can automatically grow in size, use the following command:

```
# vxsnap -g mydg make source=myvol/new=snap4myvol\
/cachesize=1g/autogrow=yes alloc=mydg15
```

If a cache is created implicitly by specifying `cachesize`, and `ncachemirror` is specified to be greater than 1, a DCO is attached to the cache volume to enable dirty region logging (DRL). DRL allows fast recovery of the cache backing store after a system crash. The DCO is allocated on the same disks as those that are occupied by the DCO of the source volume. This is done to allow the cache and the source volume to remain in the same disk group for disk group move, split and join operations.

2. Clean the temporary volume's contents using an appropriate utility such as `fsck` for non-VxVM file systems and log replay for databases. Because VxVM calls VxFS and places VxFS file systems in a constant state immediately before taking a snapshot, it is not usually necessary to run `fsck` on a VxFS file system on the temporary volume. If a VxFS file system contains a database, it will still be necessary to perform database log replay.

3. To backup the data in the snapshot, use an appropriate utility or operating system command to copy the contents of the snapshot to tape, or to some other backup medium.

4. You now have the following options:

   - Refresh the contents of the snapshot. This creates a new point-in-time image of the original volume ready for another backup. If synchronization was already in progress on the snapshot, this operation may result in large portions of the snapshot having to be resynchronized.
     See “Refreshing an instant space-optimized snapshot” on page 413.

   - Restore the contents of the original volume from the snapshot volume. The space-optimized instant snapshot remains intact at the end of the operation.
     See “Restoring a volume from an instant space-optimized snapshot” on page 415.

   - Destroy the snapshot.
     See “Removing an instant snapshot” on page 416.

Creating and managing full-sized instant snapshots

Full-sized instant snapshots are not suitable for write-intensive volumes (such as for database redo logs) because the copy-on-write mechanism may degrade the performance of the volume.

For full-sized instant snapshots, you must prepare a volume that is to be used as the snapshot volume. This must be the same size as the volume for which the snapshot is being created, and it must also have the same region size.

See "Creating a volume for use as a full-sized instant or linked break-off snapshot" on page 396.

The attributes for a snapshot are specified as a tuple to the `vxsnap make` command. This command accepts multiple tuples. One tuple is required for each snapshot that is being created. Each element of a tuple is separated from the next by a slash character (`/`). Tuples are separated by white space.

To create and manage a full-sized instant snapshot

1 To create a full-sized instant snapshot, use the following form of the `vxsnap make` command:

```
# vxsnap [-g diskgroup] make source=volume/snapvol=snapvol\
[/snapdg=snapdiskgroup] [/syncing=off]
```

The command specifies the volume, *snapvol*, that you prepared earlier.

For example, to use the prepared volume, `snap1myvol`, as the snapshot for the volume, `myvol`, in the disk group, `mydg`, use the following command:

```
# vxsnap -g mydg make source=myvol/snapvol=snap1myvol
```

For full-sized instant snapshots that are created from an empty volume, background synchronization is enabled by default (equivalent to specifying the `syncing=on` attribute). To move a snapshot into a separate disk group, or to turn it into an independent volume, you must wait for its contents to be synchronized with those of its parent volume.

You can use the `vxsnap syncwait` command to wait for the synchronization of the snapshot volume to be completed, as shown here:

```
# vxsnap [-g diskgroup] syncwait snapvol
```

For example, you would use the following command to wait for synchronization to finish on the snapshot volume, snap2myvol:

```
# vxsnap -g mydg syncwait snap2myvol
```

This command exits (with a return code of zero) when synchronization of the snapshot volume is complete. The snapshot volume may then be moved to another disk group or turned into an independent volume.

See "Controlling instant snapshot synchronization" on page 419.

If required, you can use the following command to test if the synchronization of a volume is complete:

```
# vxprint [-g diskgroup] -F%incomplete snapvol
```

This command returns the value `off` if synchronization of the volume, *snapvol*, is complete; otherwise, it returns the value `on`.

You can also use the `vxsnap print` command to check on the progress of synchronization.

See "Displaying snapshot information" on page 388.

If you do not want to move the snapshot into a separate disk group, or to turn it into an independent volume, specify the `syncing=off` attribute. This avoids unnecessary system overhead. For example, to turn off synchronization when creating the snapshot of the volume, `myvol`, you would use the following form of the `vxsnap make` command:

```
# vxsnap -g mydg make source=myvol/snapvol=snap1myvol\
/syncing=off
```

2. Clean the temporary volume's contents using an appropriate utility such as `fsck` for non-VxVM file systems and log replay for databases. Because VxVM calls VxFS and places VxFS file systems in a constant state immediately before taking a snapshot, it is not usually necessary to run `fsck` on a VxFS file system on the temporary volume. If a VxFS file system contains a database, it will still be necessary to perform database log replay.

3. To backup the data in the snapshot, use an appropriate utility or operating system command to copy the contents of the snapshot to tape, or to some other backup medium.

4. You now have the following options:

   - Refresh the contents of the snapshot. This creates a new point-in-time image of the original volume ready for another backup. If synchronization was already in progress on the snapshot, this operation may result in large portions of the snapshot having to be resynchronized.
     See "Refreshing an instant space-optimized snapshot" on page 413.

   - Reattach some or all of the plexes of the snapshot volume with the original volume.

See “Reattaching an instant full-sized or plex break-off snapshot” on page 413.

- Restore the contents of the original volume from the snapshot volume. You can choose whether none, a subset, or all of the plexes of the snapshot volume are returned to the original volume as a result of the operation.
  See “Restoring a volume from an instant space-optimized snapshot” on page 415.
- Dissociate the snapshot volume entirely from the original volume. This may be useful if you want to use the copy for other purposes such as testing or report generation. If desired, you can delete the dissociated volume.
  See “Dissociating an instant snapshot” on page 415.
- If the snapshot is part of a snapshot hierarchy, you can also choose to split this hierarchy from its parent volumes.
  See “Splitting an instant snapshot hierarchy” on page 416.

Creating and managing third-mirror break-off snapshots

Break-off snapshots are suitable for write-intensive volumes, such as database redo logs.

To turn one or more existing plexes in a volume into a break-off instant snapshot volume, the volume must be a non-layered volume with a `mirror` or `mirror-stripe` layout, or a RAID-5 volume that you have converted to a special layered volume and then mirrored. The plexes in a volume with a `stripe-mirror` layout are mirrored at the subvolume level, and cannot be broken off.

The attributes for a snapshot are specified as a tuple to the `vxsnap make` command. This command accepts multiple tuples. One tuple is required for each snapshot that is being created. Each element of a tuple is separated from the next by a slash character (`/`). Tuples are separated by white space.

To create and manage a third-mirror break-off snapshot

1 To create the snapshot, you can either take some of the existing ACTIVE plexes in the volume, or you can use the following command to add new snapshot mirrors to the volume:

```
# vxsnap [-b] [-g diskgroup] addmir volume [nmirror=N] \
  [alloc=storage_attributes]
```

By default, the vxsnap addmir command adds one snapshot mirror to a volume unless you use the nmirror attribute to specify a different number of mirrors. The mirrors remain in the SNAPATT state until they are fully synchronized. The -b option can be used to perform the synchronization in the background. Once synchronized, the mirrors are placed in the SNAPDONE state.

For example, the following command adds 2 mirrors to the volume, vol1, on disks mydg10 and mydg11:

```
# vxsnap -g mydg addmir vol1 nmirror=2 alloc=mydg10,mydg11
```

If you specify the -b option to the vxsnap addmir command, you can use the vxsnap snapwait command to wait for synchronization of the snapshot plexes to complete, as shown in this example:

```
# vxsnap -g mydg snapwait vol1 nmirror=2
```

2. To create a third-mirror break-off snapshot, use the following form of the `vxsnap make` command.

   ```
   # vxsnap [-g diskgroup] make source=volume[/newvol=snapvol]\
   {/plex=plex1[,plex2,...]|/nmirror=number}
   ```

   Either of the following attributes may be specified to create the new snapshot volume, *snapvol*, by breaking off one or more existing plexes in the original volume:

`plex`	Specifies the plexes in the existing volume that are to be broken off.
`nmirror`	Specifies how many plexes are to be broken off. This attribute can only be used with plexes that are in the `SNAPDONE` state. (Such plexes could have been added to the volume by using the `vxsnap addmir` command.)

   Snapshots that are created from one or more `ACTIVE` or `SNAPDONE` plexes in the volume are already synchronized by definition.

   For backup purposes, a snapshot volume with one plex should be sufficient.

   For example, to create the instant snapshot volume, `snap2myvol`, of the volume, `myvol`, in the disk group, `mydg`, from a single existing plex in the volume, use the following command:

   ```
   # vxsnap -g mydg make source=myvol/newvol=snap2myvol/nmirror=1
   ```

   The next example shows how to create a mirrored snapshot from two existing plexes in the volume:

```
# vxsnap -g mydg make source=myvol/newvol=snap2myvol/plex=myvol-03,myvol-04
```

3. Clean the temporary volume's contents using an appropriate utility such as `fsck` for non-VxVM file systems and log replay for databases. Because VxVM calls VxFS and places VxFS file systems in a constant state immediately before taking a snapshot, it is not usually necessary to run `fsck` on a VxFS file system on the temporary volume. If a VxFS file system contains a database, it will still be necessary to perform database log replay.

4. To backup the data in the snapshot, use an appropriate utility or operating system command to copy the contents of the snapshot to tape, or to some other backup medium.

5. You now have the following options:

- Refresh the contents of the snapshot. This creates a new point-in-time image of the original volume ready for another backup. If synchronization was already in progress on the snapshot, this operation may result in large portions of the snapshot having to be resynchronized.
  See "Refreshing an instant space-optimized snapshot" on page 413.
- Reattach some or all of the plexes of the snapshot volume with the original volume.
  See "Reattaching an instant full-sized or plex break-off snapshot" on page 413.
- Restore the contents of the original volume from the snapshot volume. You can choose whether none, a subset, or all of the plexes of the snapshot volume are returned to the original volume as a result of the operation.
  See "Restoring a volume from an instant space-optimized snapshot" on page 415.
- Dissociate the snapshot volume entirely from the original volume. This may be useful if you want to use the copy for other purposes such as testing or report generation. If desired, you can delete the dissociated volume.
  See "Dissociating an instant snapshot" on page 415.
- If the snapshot is part of a snapshot hierarchy, you can also choose to split this hierarchy from its parent volumes.
  See "Splitting an instant snapshot hierarchy" on page 416.

Creating and managing linked break-off snapshot volumes

Linked break-off snapshots are suitable for write-intensive volumes. Specifically, they are used for off-host processing, because the snapshot could be in a different disk group to start with and could avoid disk group split/join operations

For linked break-off snapshots, you must prepare a volume that is to be used as the snapshot volume. This must be the same size as the volume for which the snapshot is being created, and it must also have the same region size.

See "Creating a volume for use as a full-sized instant or linked break-off snapshot" on page 396.

The attributes for a snapshot are specified as a tuple to the `vxsnap make` command. This command accepts multiple tuples. One tuple is required for each snapshot that is being created. Each element of a tuple is separated from the next by a slash character (`/`). Tuples are separated by white space.

To create and manage a linked break-off snapshot

1. Use the following command to link the prepared snapshot volume, *snapvol*, to the data volume:

   ```
   # vxsnap [-g diskgroup] [-b] addmir volume mirvol=snapvol \
     [mirdg=snapdg]
   ```

   The optional `mirdg` attribute can be used to specify the snapshot volume's current disk group, *snapdg*. The `-b` option can be used to perform the synchronization in the background. If the `-b` option is not specified, the command does not return until the link becomes `ACTIVE`.

   For example, the following command links the prepared volume, `prepsnap`, in the disk group, `mysnapdg`, to the volume, `vol1`, in the disk group, `mydg`:

   ```
   # vxsnap -g mydg -b addmir vol1 mirvol=prepsnap mirdg=mysnapdg
   ```

   If the `-b` option is specified, you can use the `vxsnap snapwait` command to wait for the synchronization of the linked snapshot volume to complete, as shown in this example:

   ```
   # vxsnap -g mydg snapwait vol1 mirvol=prepsnap mirdg=mysnapvoldg
   ```

2. To create a linked break-off snapshot, use the following form of the `vxsnap make` command.

   ```
   # vxsnap [-g diskgroup] make   source=volume/snapvol=snapvol\
   [/snapdg=snapdiskgroup]
   ```

   The `snapdg` attribute must be used to specify the snapshot volume's disk group if this is different from that of the data volume.

   For example, to use the prepared volume, `prepsnap`, as the snapshot for the volume, `vol1`, in the disk group, `mydg`, use the following command:

   ```
   # vxsnap -g mydg make source=vol1/snapvol=prepsnap/snapdg=mysnapdg
   ```

3. Clean the temporary volume's contents using an appropriate utility such as `fsck` for non-VxVM file systems and log replay for databases. Because VxVM calls VxFS and places VxFS file systems in a constant state immediately before taking a snapshot, it is not usually necessary to run `fsck` on a VxFS file system on the temporary volume. If a VxFS file system contains a database, it will still be necessary to perform database log replay.

4 To backup the data in the snapshot, use an appropriate utility or operating system command to copy the contents of the snapshot to tape, or to some other backup medium.

5 You now have the following options:

- Refresh the contents of the snapshot. This creates a new point-in-time image of the original volume ready for another backup. If synchronization was already in progress on the snapshot, this operation may result in large portions of the snapshot having to be resynchronized.
  See "Refreshing an instant space-optimized snapshot" on page 413.
- Reattach the snapshot volume with the original volume.
  See "Reattaching a linked break-off snapshot volume" on page 414.
- Dissociate the snapshot volume entirely from the original volume. This may be useful if you want to use the copy for other purposes such as testing or report generation. If desired, you can delete the dissociated volume.
  See "Dissociating an instant snapshot" on page 415.
- If the snapshot is part of a snapshot hierarchy, you can also choose to split this hierarchy from its parent volumes.
  See "Splitting an instant snapshot hierarchy" on page 416.

Creating multiple instant snapshots

You can create multiple instant snapshots for all volumes that form a consistent group. The `vxsnap make` command accepts multiple tuples that define the source and snapshot volumes names as their arguments. For example, to create three instant snapshots, each with the same redundancy, from specified storage, the following form of the command can be used:

```
# vxsnap [-g diskgroup] make source=vol1/snapvol=snapvol1\
  source=vol2/snapvol=snapvol2 source=vol3/snapvol=snapvol3
```

The snapshot volumes (*snapvol1*, *snapvol2* and so on) must have been prepared in advance.

See "Creating a volume for use as a full-sized instant or linked break-off snapshot" on page 396.

The specified source volumes (*vol1*, *vol2* and so on) may be the same volume or they can be different volumes.

If all the snapshots are to be space-optimized and to share the same cache, the following form of the command can be used:

```
# vxsnap [-g diskgroup] make \
  source=vol1/newvol=snapvol1/cache=cacheobj \
  source=vol2/newvol=snapvol2/cache=cacheobj \
  source=vol3/newvol=snapvol3/cache=cacheobj \
  [alloc=storage_attributes]
```

The `vxsnap make` command also allows the snapshots to be of different types, have different redundancy, and be configured from different storage, as shown here:

```
# vxsnap [-g diskgroup] make source=vol1/snapvol=snapvol1 \
  source=vol2[/newvol=snapvol2]/cache=cacheobj\
[/alloc=storage_attributes2][/nmirror=number2]
  source=vol3[/newvol=snapvol3][/alloc=storage_attributes3]\
/nmirror=number3
```

In this example, *snapvol1* is a full-sized snapshot that uses a prepared volume, *snapvol2* is a space-optimized snapshot that uses a prepared cache, and *snapvol3* is a break-off full-sized snapshot that is formed from plexes of the original volume.

An example of where you might want to create mixed types of snapshots at the same time is when taking snapshots of volumes containing database redo logs and database tables:

```
# vxsnap -g mydg make \
  source=logv1/newvol=snplogv1/drl=sequential/nmirror=1 \
  source=logv2/newvol=snplogv2/drl=sequential/nmirror=1 \
  source=datav1/newvol=snpdatav1/cache=mydgcobj/drl=on \
  source=datav2/newvol=snpdatav2/cache=mydgcobj/drl=on
```

In this example, sequential DRL is enabled for the snapshots of the redo log volumes, and normal DRL is applied to the snapshots of the volumes that contain the database tables. The two space-optimized snapshots are configured to share the same cache object in the disk group. Also note that break-off snapshots are used for the redo logs as such volumes are write intensive.

Creating instant snapshots of volume sets

Volume set names can be used in place of volume names with the following `vxsnap` operations on instant snapshots: `addmir`, `dis`, `make`, `prepare`, `reattach`, `refresh`, `restore`, `rmmir`, `split`, `syncpause`, `syncresume`, `syncstart`, `syncstop`, `syncwait`, and `unprepare`.

The procedure for creating an instant snapshot of a volume set is the same as that for a standalone volume. However, there are certain restrictions if a full-sized instant snapshot is to be created from a prepared volume set. A full-sized instant snapshot of a volume set must itself be a volume set with the same number of volumes, and the same volume sizes and index numbers as the parent. For example, if a volume set contains three volumes with sizes 1GB, 2GB and 3GB, and indexes 0, 1 and 2 respectively, then the snapshot volume set must have three volumes with the same sizes matched to the same set of index numbers. The corresponding volumes in the parent and snapshot volume sets are also subject to the same restrictions as apply between standalone volumes and their snapshots.

You can use the `vxvset list` command to verify that the volume sets have identical characteristics as shown in this example:

```
# vxvset -g mydg list vset1

VOLUME        INDEX      LENGTH      KSTATE      CONTEXT
vol_0         0          204800      ENABLED     -
vol_1         1          409600      ENABLED     -
vol_2         2          614400      ENABLED     -

# vxvset -g mydg list snapvset1

VOLUME        INDEX      LENGTH      KSTATE      CONTEXT
svol_0        0          204800      ENABLED     -
svol_1        1          409600      ENABLED     -
svol_2        2          614400      ENABLED     -
```

A full-sized instant snapshot of a volume set can be created using a prepared volume set in which each volume is the same size as the corresponding volume in the parent volume set. Alternatively, you can use the `nmirrors` attribute to specify the number of plexes that are to be broken off provided that sufficient plexes exist for each volume in the volume set.

The following example shows how to prepare a source volume set, `vset1`, and an identical volume set, `snapvset1`, which is then used to create the snapshot:

```
# vxsnap -g mydg prepare vset1
# vxsnap -g mydg prepare snapvset1
# vxsnap -g mydg make source=vset1/snapvol=snapvset1
```

To create a full-sized third-mirror break-off snapshot, you must ensure that each volume in the source volume set contains sufficient plexes. The following example shows how to achieve this by using the `vxsnap` command to add the required number of plexes before breaking off the snapshot:

```
# vxsnap -g mydg prepare vset2
# vxsnap -g mydg addmir vset2 nmirror=1
# vxsnap -g mydg make source=vset2/newvol=snapvset2/nmirror=1
```

See "Adding snapshot mirrors to a volume" on page 411.

To create a space-optimized instant snapshot of a volume set, the commands are again identical to those for a standalone volume as shown in these examples:

```
# vxsnap -g mydg prepare vset3
# vxsnap -g mydg make source=vset3/newvol=snapvset3/cachesize=20m

# vxsnap -g mydg prepare vset4
# vxsnap -g mydg make source=vset4/newvol=snapvset4/cache=mycobj
```

Here a new cache object is created for the volume set, `vset3`, and an existing cache object, `mycobj`, is used for `vset4`.

Adding snapshot mirrors to a volume

If you are going to create a full-sized break-off snapshot volume, you can use the following command to add new snapshot mirrors to a volume:

```
# vxsnap [-b] [-g diskgroup] addmir volume|volume_set \
  [nmirror=N] [alloc=storage_attributes]
```

The volume must have been prepared using the `vxsnap prepare` command.

If a volume set name is specified instead of a volume, the specified number of plexes is added to each volume in the volume set.

By default, the `vxsnap addmir` command adds one snapshot mirror to a volume unless you use the `nmirror` attribute to specify a different number of mirrors. The mirrors remain in the `SNAPATT` state until they are fully synchronized. The `-b` option can be used to perform the synchronization in the background. Once synchronized, the mirrors are placed in the `SNAPDONE` state.

For example, the following command adds 2 mirrors to the volume, `vol1`, on disks `mydg10` and `mydg11`:

```
# vxsnap -g mydg addmir vol1 nmirror=2 alloc=mydg10,mydg11
```

This command is similar in usage to the `vxassist snapstart` command, and supports the traditional third-mirror break-off snapshot model. As such, it does not provide an instant snapshot capability.

Once you have added one or more snapshot mirrors to a volume, you can use the `vxsnap make` command with either the `nmirror` attribute or the `plex` attribute to create the snapshot volumes.

Removing a snapshot mirror

To remove a single snapshot mirror from a volume, use this command:

```
# vxsnap [-g diskgroup] rmmir volume|volume_set
```

For example, the following command removes a snapshot mirror from the volume, `vol1`:

```
# vxsnap -g mydg rmmir vol1
```

This command is similar in usage to the `vxassist snapabort` command.

If a volume set name is specified instead of a volume, a mirror is removed from each volume in the volume set.

Removing a linked break-off snapshot volume

To remove a linked break-off snapshot volume from a volume, use this command:

```
# vxsnap [-g diskgroup] rmmir volume|volume_set mirvol=snapvol \
  [mirdg=snapdiskgroup]
```

The `mirvol` and optional `mirdg` attributes specify the snapshot volume, *snapvol*, and its disk group, *snapdiskgroup*. For example, the following command removes a linked snapshot volume, `prepsnap`, from the volume, `vol1`:

```
# vxsnap -g mydg rmmir vol1 mirvol=prepsnap mirdg=mysnapdg
```

Adding a snapshot to a cascaded snapshot hierarchy

To create a snapshot and push it onto a snapshot hierarchy between the original volume and an existing snapshot volume, specify the name of the existing snapshot volume as the value of the `infrontof` attribute to the `vxsnap make` command. The following example shows how to place the space-optimized snapshot, `thurs_bu`, of the volume, `dbvol`, in front of the earlier snapshot, `wed_bu`:

```
# vxsnap -g dbdg make source=dbvol/newvol=thurs_bu/\
infrontof=wed_bu/cache=dbdgcache
```

Similarly, the next snapshot that is taken, `fri_bu`, is placed in front of `thurs_bu`:

```
# vxsnap -g dbdg make source=dbvol/newvol=fri_bu/\
infrontof=thurs_bu/cache=dbdgcache
```

See "Controlling instant snapshot synchronization" on page 419.

Refreshing an instant space-optimized snapshot

Refreshing an instant snapshot replaces it with another point-in-time copy of a parent volume. To refresh one or more snapshots and make them immediately available for use, use the following command:

```
# vxsnap [-g diskgroup] refresh snapvolume|snapvolume_set \
  [source=volume|volume_set] [snapvol2 [source=vol2]...] \
```

If the source volume is not specified, the immediate parent of the snapshot is used.

Warning: The snapshot that is being refreshed must not be open to any application. For example, any file system configured on the volume must first be unmounted.

Reattaching an instant full-sized or plex break-off snapshot

Using the following command, some or all plexes of an instant snapshot may be reattached to the specified original volume, or to a source volume in the snapshot hierarchy above the snapshot volume:

```
# vxsnap [-g diskgroup] reattach snapvolume|snapvolume_set \
  source=volume|volume_set [nmirror=number]
```

By default, all the plexes are reattached, which results in the removal of the snapshot. If required, the number of plexes to be reattached may be specified as the value assigned to the `nmirror` attribute.

Warning: The snapshot that is being reattached must not be open to any application. For example, any file system configured on the snapshot volume must first be unmounted.

It is possible to reattach a volume to an unrelated volume provided that their volume sizes and region sizes are compatible.

For example the following command reattaches one plex from the snapshot volume, `snapmyvol`, to the volume, `myvol`:

```
# vxsnap -g mydg reattach snapmyvol source=myvol nmirror=1
```

While the reattached plexes are being resynchronized from the data in the parent volume, they remain in the SNAPTMP state. After resynchronization is complete, the plexes are placed in the SNAPDONE state. You can use the vxsnap snapwait command (but not vxsnap syncwait) to wait for the resynchronization of the reattached plexes to complete, as shown here:

```
# vxsnap -g mydg snapwait myvol nmirror=1
```

If the volume and its snapshot have both been resized (to an identical smaller or larger size) before performing the reattachment, a fast resynchronization can still be performed. A full resynchronization is not required. Instant snap DCO volumes are resized proportionately when the associated data volume is resized. For version 0 DCO volumes, the FastResync maps stay the same size, but the region size is recalculated, and the locations of the dirty bits in the existing maps are adjusted. In both cases, new regions are marked as dirty in the maps.

Reattaching a linked break-off snapshot volume

Unlike other types of snapshot, the reattachment operation for linked break-off snapshot volumes does not return the plexes of the snapshot volume to the parent volume. The link relationship is re-established that makes the snapshot volume a mirror of the parent volume, and this allows the snapshot data to be resynchronized.

To reattach a linked break-off snapshot volume, use the following form of the vxsnap reattach command:

```
# vxsnap [-g snapdiskgroup] reattach snapvolume|snapvolume_set \
  source=volume|volume_set [sourcedg=diskgroup]
```

The sourcedg attribute must be used to specify the data volume's disk group if this is different from the snapshot volume's disk group, *snapdiskgroup*.

Warning: The snapshot that is being reattached must not be open to any application. For example, any file system configured on the snapshot volume must first be unmounted.

It is possible to reattach a volume to an unrelated volume provided that their sizes and region sizes are compatible.

For example the following command reattaches the snapshot volume, prepsnap, in the disk group, snapdg, to the volume, myvol, in the disk group, mydg:

```
# vxsnap -g snapdg reattach prepsnap source=myvol sourcedg=mydg
```

After resynchronization of the snapshot volume is complete, the link is placed in the ACTIVE state. You can use the vxsnap snapwait command (but not vxsnap syncwait) to wait for the resynchronization of the reattached volume to complete, as shown here:

```
# vxsnap -g snapdg snapwait myvol mirvol=prepsnap
```

Restoring a volume from an instant space-optimized snapshot

It may sometimes be desirable to reinstate the contents of a volume from a backup or modified replica in a snapshot volume. The following command may be used to restore one or more volumes from the specified snapshots:

```
# vxsnap [-g diskgroup] restore volume|volume_set \
  source=snapvolume|snapvolume_set \
  [[volume2|volume_set2 \
  source=snapvolume2|snapvolume_set2]...]\
 [syncing=yes|no]
```

For a space-optimized instant snapshot, the cached data is used to recreate the contents of the specified volume. The space-optimized instant snapshot remains unchanged by the restore operation.

Warning: For this operation to succeed, the volume that is being restored and the snapshot volume must not be open to any application. For example, any file systems that are configured on either volume must first be unmounted.

It is not possible to restore a volume from an unrelated volume.

The following example demonstrates how to restore the volume, myvol, from the space-optimized snapshot, snap3myvol.

```
# vxsnap -g mydg restore myvol source=snap3myvol
```

Dissociating an instant snapshot

The following command breaks the association between a full-sized instant snapshot volume, *snapvol*, and its parent volume, so that the snapshot may be used as an independent volume:

```
# vxsnap [-f] [-g diskgroup] dis snapvolume|snapvolume_set
```

This operation fails if the snapshot, *snapvol*, has unsynchronized snapshots. If this happens, the dependent snapshots must be fully synchronized from *snapvol*. When no dependent snapshots remain, *snapvol* may be dissociated. The snapshot hierarchy is then adopted by the parent volume of *snapvol*.

See "Controlling instant snapshot synchronization" on page 419.

See "Removing an instant snapshot" on page 416.

The following command dissociates the snapshot, `snap2myvol`, from its parent volume:

```
# vxsnap -g mydg dis snap2myvol
```

Warning: When applied to a volume set or to a component volume of a volume set, this operation can result in inconsistencies in the snapshot hierarchy in the case of a system crash or hardware failure. If the operation is applied to a volume set, the `-f` (force) option must be specified.

Removing an instant snapshot

When you have dissociated a full-sized instant snapshot, you can use the `vxedit` command to delete it altogether, as shown in this example:

```
# vxedit -g mydg -r rm snap2myvol
```

You can also use this command to remove a space-optimized instant snapshot from its cache.

See "Removing a cache" on page 423.

Splitting an instant snapshot hierarchy

Note: This operation is not supported for space-optimized instant snapshots.

The following command breaks the association between a snapshot hierarchy that has the snapshot volume, *snapvol*, at its head, and its parent volume, so that the snapshot hierarchy may be used independently of the parent volume:

```
# vxsnap [-f] [-g diskgroup] split snapvolume|snapvolume_set
```

The topmost snapshot volume in the hierarchy must have been fully synchronized for this command to succeed. Snapshots that are lower down in the hierarchy need not have been fully resynchronized.

See "Controlling instant snapshot synchronization" on page 419.

The following command splits the snapshot hierarchy under snap2myvol from its parent volume:

```
# vxsnap -g mydg split snap2myvol
```

Warning: When applied to a volume set or to a component volume of a volume set, this operation can result in inconsistencies in the snapshot hierarchy in the case of a system crash or hardware failure. If the operation is applied to a volume set, the -f (force) option must be specified.

Displaying instant snapshot information

The vxsnap print command may be used to display information about the snapshots that are associated with a volume.

```
# vxsnap [-g diskgroup] print [vol]
```

This command shows the percentage progress of the synchronization of a snapshot or volume. If no volume is specified, information about the snapshots for all the volumes in a disk group is displayed. The following example shows a volume, vol1, which has a full-sized snapshot, snapvol1 whose contents have not been synchronized with vol1:

```
# vxsnap -g mydg print
NAME      SNAPOBJECT     TYPE    PARENT   SNAPSHOT   %DIRTY   %VALID

vol1      --             volume  --       --         --       100
          snapvol1_snp1  volume  --       snapvol1   1.30     --
snapvol1  vol1_snp1      volume  vol1     --         1.30     1.30
```

The %DIRTY value for snapvol1 shows that its contents have changed by 1.30% when compared with the contents of vol1. As snapvol1 has not been synchronized with vol1, the %VALID value is the same as the %DIRTY value. If the snapshot were partly synchronized, the %VALID value would lie between the %DIRTY value and 100%. If the snapshot were fully synchronized, the %VALID value would be 100%. The snapshot could then be made independent or moved into another disk group.

Additional information about the snapshots of volumes and volume sets can be obtained by using the -n option with the vxsnap print command:

```
# vxsnap [-g diskgroup] -n [-l] [-v] [-x] print [vol]
```

Alternatively, you can use the vxsnap list command, which is an alias for the vxsnap -n print command:

```
# vxsnap [-g diskgroup] [-l] [-v] [-x] list [vol]
```

The following output is an example of using this command on the disk group `dg1`:

```
# vxsnap -g dg -vx list

NAME   DG    OBJTYPE SNAPTYPE  PARENT  PARENTDG  SNAPDATE        CHANGE_DATA    SYNCED_DATA
vol    dg1   vol     -         -       -         -        -                     10G (100%)
svol1  dg2   vol     fullinst  vol     dg1       2006/2/1 12:29  20M (0.2%)     60M (0.6%)
svol2  dg1   vol     mirbrk    vol     dg1       2006/2/1 12:29  120M (1.2%)    10G (100%)
svol3  dg2   vol     volbrk    vol     dg1       2006/2/1 12:29  105M (1.1%)    10G (100%)
svol21 dg1   vol     spaceopt  svol2   dg1       2006/2/1 12:29  52M (0.5%)     52M (0.5%)
vol-02 dg1   plex    snapmir   vol     dg1       -               -              56M (0.6%)
mvol   dg2   vol     mirvol    vol     dg1       -               -              58M (0.6%)
vset1  dg1   vset    -     -   -       -         -                              2G (100%)
v1     dg1   compvol -     -   -       -         -                              1G (100%)
v2     dg1   compvol -     -   -       -         -                              1G (100%)
svset1 dg1   vset    mirbrk    vset    dg1       2006/2/1 12:29  1G (50%)       2G (100%)
sv1    dg1   compvol mirbrk    v1      dg1       2006/2/1 12:29  512M (50%)     1G (100%)
sv2    dg1   compvol mirbrk    v2      dg1       2006/2/1 12:29  512M (50%)     1G (100%)
vol-03 dg1   plex    detmir    vol     dg1       -               20M (0.2%)     -
mvol2  dg2   vol     detvol    vol     dg1       -               20M (0.2%)     -
```

This shows that the volume `vol` has three full-sized snapshots, `svol1`, `svol2` and `svol3`, which are of types full-sized instant (`fullinst`), mirror break-off (`mirbrk`) and linked break-off (`volbrk`). It also has one snapshot plex (`snapmir`), `vol-02`, and one linked mirror volume (`mirvol`), `mvol`. The snapshot `svol2` itself has a space-optimized instant snapshot (`spaceopt`), `svol21`. There is also a volume set, `vset1`, with component volumes `v1` and `v2`. This volume set has a mirror break-off snapshot, `svset1`, with component volumes `sv1` and `sv2`. The last two entries show a detached plex, `vol-03`, and a detached mirror volume, `mvol2`, which have `vol` as their parent volume. These snapshot objects may have become detached due to an I/O error, or, in the case of the plex, by running the `vxplex det` command.

The `CHANGE_DATA` column shows the approximate difference between the current contents of the snapshot and its parent volume. This corresponds to the amount of data that would have to be resynchronized to make the contents the same again.

The `SYNCED_DATA` column shows the approximate progress of synchronization since the snapshot was taken.

The `-l` option can be used to obtain a longer form of the output listing instead of the tabular form.

The `-x` option expands the output to include the component volumes of volume sets.

See the `vxsnap`(1M) manual page for more information about using the `vxsnap print` and `vxsnap list` commands.

Controlling instant snapshot synchronization

Synchronization of the contents of a snapshot with its original volume is not possible for space-optimized instant snapshots.

By default, synchronization is enabled for the `vxsnap reattach`, `refresh` and `restore` operations on instant snapshots. Otherwise, synchronization is disabled unless you specify the `syncing=yes` attribute to the `vxsnap` command.

Table 17-1 shows the commands that are provided for controlling the synchronization manually.

Table 17-1 Commands for controlling instant snapshot synchronization

Command	Description

`vxsnap [-g` *`diskgroup`*`] syncpause \`
`vol``|`*`vol_set`*
	Pause synchronization of a volume.

`vxsnap [-g` *`diskgroup`*`] syncresume \`
`vol``|`*`vol_set`*
	Resume synchronization of a volume.

`vxsnap [-b] [-g` *`diskgroup`*`] syncstart \`
`vol``|`*`vol_set`*
	Start synchronization of a volume. The `-b` option puts the operation in the background.

`vxsnap [-g` *`diskgroup`*`] syncstop \`
`vol``|`*`vol_set`*
	Stop synchronization of a volume.

`vxsnap [-g` *`diskgroup`*`] syncwait \`
`vol``|`*`vol_set`*
	Exit when synchronization of a volume is complete. An error is returned if the *vol* or *vol_set* is invalid (for example, it is a space-optimized snapshot), or if the *vol* or *vol_set* is not being synchronized.
	Note: You cannot use this command to wait for synchronization of reattached plexes to complete.

The commands that are shown in Table 17-1 cannot be used to control the synchronization of linked break-off snapshots.

The `vxsnap snapwait` command is provided to wait for the link between new linked break-off snapshots to become ACTIVE, or for reattached snapshot plexes to reach the SNAPDONE state following resynchronization.

See "Creating and managing linked break-off snapshot volumes" on page 406.

See "Reattaching an instant full-sized or plex break-off snapshot" on page 413.

See "Reattaching a linked break-off snapshot volume" on page 414.

Improving the performance of snapshot synchronization

The following optional arguments to the `-o` option are provided to help optimize the performance of synchronization when using the `make`, `refresh`, `restore` and `syncstart` operations with full-sized instant snapshots:

`iosize=`*`size`*

Specifies the size of each I/O request that is used when synchronizing the regions of a volume. Specifying a larger size causes synchronization to complete sooner, but with greater impact on the performance of other processes that are accessing the volume. The default size of 1m (1MB) is suggested as the minimum value for high-performance array and controller hardware. The specified value is rounded to a multiple of the volume's region size.

`slow=`*`iodelay`*

Specifies the delay in milliseconds between synchronizing successive sets of regions as specified by the value of `iosize`. This can be used to change the impact of synchronization on system performance. The default value of *iodelay* is 0 milliseconds (no delay). Increasing this value slows down synchronization, and reduces the competition for I/O bandwidth with other processes that may be accessing the volume.

Options may be combined as shown in the following examples:

```
# vxsnap -g mydg -o iosize=2m,slow=100 make \
  source=myvol/snapvol=snap2myvol/syncing=on
```

```
# vxsnap -g mydg -o iosize=10m,slow=250 syncstart snap2myvol
```

Note: The `iosize` and `slow` parameters are not supported for space-optimized snapshots.

Listing the snapshots created on a cache

To list the space-optimized instant snapshots that have been created on a cache object, use the following command:

```
# vxcache [-g diskgroup] listvol cache_object
```

The snapshot names are printed as a space-separated list ordered by timestamp. If two or more snapshots have the same timestamp, these snapshots are sorted in order of decreasing size.

Tuning the autogrow attributes of a cache

The `highwatermark`, `autogrowby` and `maxautogrow` attributes determine how the VxVM cache daemon (`vxcached`) maintains the cache if the `autogrow` feature has been enabled and `vxcached` is running:

- When cache usage reaches the high watermark value, `highwatermark` (default value is 90 percent), `vxcached` grows the size of the cache volume by the value of `autogrowby` (default value is 20% of the size of the cache volume in blocks). The new required cache size cannot exceed the value of `maxautogrow` (default value is twice the size of the cache volume in blocks).
- When cache usage reaches the high watermark value, and the new required cache size would exceed the value of `maxautogrow`, `vxcached` deletes the oldest snapshot in the cache. If there are several snapshots with the same age, the largest of these is deleted.

If the `autogrow` feature has been disabled:

- When cache usage reaches the high watermark value, `vxcached` deletes the oldest snapshot in the cache. If there are several snapshots with the same age, the largest of these is deleted. If there is only a single snapshot, this snapshot is detached and marked as invalid.

Note: The `vxcached` daemon does not remove snapshots that are currently open, and it does not remove the last or only snapshot in the cache.

If the cache space becomes exhausted, the snapshot is detached and marked as invalid. If this happens, the snapshot is unrecoverable and must be removed. Enabling the `autogrow` feature on the cache helps to avoid this situation occurring. However, for very small caches (of the order of a few megabytes), it is possible for the cache to become exhausted before the system has time to respond and grow the cache. In such cases, you can increase the size of the cache manually.

Alternatively, you can use the `vxcache set` command to reduce the value of `highwatermark` as shown in this example:

```
# vxcache -g mydg set highwatermark=60 cobjmydg
```

You can use the maxautogrow attribute to limit the maximum size to which a cache can grow. To estimate this size, consider how much the contents of each source volume are likely to change between snapshot refreshes, and allow some additional space for contingency.

If necessary, you can use the vxcache set command to change other autogrow attribute values for a cache.

See the vxcache(1M) manual page.

Monitoring and displaying cache usage

You can use the vxcache stat command to display cache usage. For example, to see how much space is used and how much remains available in all cache objects in the diskgroup mydg, enter the following:

```
# vxcache -g mydg stat
```

Growing and shrinking a cache

You can use the vxcache command to increase the size of the cache volume that is associated with a cache object:

```
# vxcache [-g diskgroup] growcacheto cache_object
    size
```

For example, to increase the size of the cache volume associated with the cache object, mycache, to 2GB, you would use the following command:

```
# vxcache -g mydg growcacheto mycache 2g
```

To grow a cache by a specified amount, use the following form of the command shown here:

```
# vxcache [-g diskgroup] growcacheby cache_object
    size
```

For example, the following command increases the size of mycache by 1GB:

```
# vxcache -g mydg growcacheby mycache 1g
```

You can similarly use the shrinkcacheby and shrinkcacheto operations to reduce the size of a cache.

See the vxcache(1M) manual page.

Removing a cache

To remove a cache completely, including the cache object, its cache volume and all space-optimized snapshots that use the cache:

1. Run the following command to find out the names of the top-level snapshot volumes that are configured on the cache object:

   ```
   # vxprint -g diskgroup -vne \
     "v_plex.pl_subdisk.sd_dm_name ~ /cache_object/"
   ```

   where *cache_object* is the name of the cache object.

2. Remove all the top-level snapshots and their dependent snapshots (this can be done with a single command):

   ```
   # vxedit -g diskgroup -r rm snapvol ...
   ```

   where *snapvol* is the name of a top-level snapshot volume.

3. Stop the cache object:

   ```
   # vxcache -g diskgroup stop cache_object
   ```

4. Finally, remove the cache object and its cache volume:

   ```
   # vxedit -g diskgroup -r rm cache_object
   ```

Linked break-off snapshots

A variant of third-mirror break-off snapshots are linked break-off snapshots, which use the `vxsnap addmir` command to link a specially prepared volume with the data volume. The volume that is used for the snapshot is prepared in the same way as for full-sized instant snapshots. However, unlike full-sized instant snapshots, this volume can be set up in a different disk group from the data volume. This makes linked break-off snapshots especially suitable for recurring off-host processing applications as it avoids the disk group split/join administrative step. As with third-mirror break-off snapshots, you must wait for the contents of the snapshot volume to be synchronized with the data volume before you can use the `vxsnap make` command to take the snapshot.

When a link is created between a volume and the mirror that will become the snapshot, separate link objects (similar to snap objects) are associated with the volume and with its mirror. The link object for the original volume points to the mirror volume, and the link object for the mirror volume points to the original volume. All

I/O is directed to both the original volume and its mirror, and a synchronization of the mirror from the data in the original volume is started.

You can use the `vxprint` command to display the state of link objects, which appear as type `ln`. Link objects can have the following states:

ACTIVE
The mirror volume has been fully synchronized from the original volume. The `vxsnap make` command can be run to create a snapshot.

ATTACHING
Synchronization of the mirror volume is in progress. The `vxsnap make` command cannot be used to create a snapshot until the state changes to ACTIVE. The `vxsnap snapwait` command can be used to wait for the synchronization to complete.

BROKEN
The mirror volume has been detached from the original volume because of an I/O error or an unsuccessful attempt to grow the mirror volume. The vxrecover command can be used to recover the mirror volume in the same way as for a DISABLED volume.

If you resize (grow or shrink) a volume, all its `ACTIVE` linked mirror volumes are also resized at the same time. The volume and its mirrors can be in the same disk group or in different disk groups. If the operation is successful, the volume and its mirrors will have the same size.

If a volume has been grown, a resynchronization of the grown regions in its linked mirror volumes is started, and the links remain in the `ATTACHING` state until resynchronization is complete. The `vxsnap snapwait` command can be used to wait for the state to become `ACTIVE.`

When you use the `vxsnap make` command to create the snapshot volume, this removes the link, and establishes a snapshot relationship between the snapshot volume and the original volume.

The `vxsnap reattach` operation re-establishes the link relationship between the two volumes, and starts a resynchronization of the mirror volume.

See "Creating and managing linked break-off snapshot volumes" on page 406.

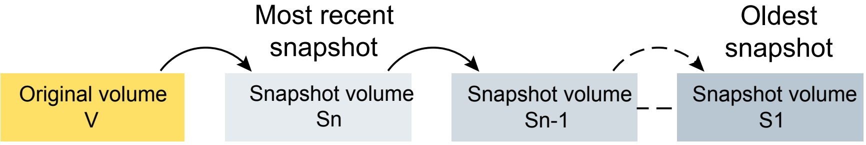
An empty volume must be prepared for use by linked break-off snapshots.

See "Creating a volume for use as a full-sized instant or linked break-off snapshot" on page 396.

Cascaded snapshots

Figure 17-3 shows a snapshot hierarchy, known as a snapshot cascade, that can improve write performance for some applications.

Figure 17-3 Snapshot cascade



Instead of having several independent snapshots of the volume, it is more efficient to make the older snapshots into children of the latest snapshot.

A snapshot cascade is most likely to be used for regular online backup of a volume where space-optimized snapshots are written to disk but not to tape.

A snapshot cascade improves write performance over the alternative of several independent snapshots, and also requires less disk space if the snapshots are space-optimized. Only the latest snapshot needs to be updated when the original volume is updated. If and when required, the older snapshots can obtain the changed data from the most recent snapshot.

A snapshot may be added to a cascade by specifying the `infrontof` attribute to the `vxsnap make` command when the second and subsequent snapshots in the cascade are created. Changes to blocks in the original volume are only written to the most recently created snapshot volume in the cascade. If an attempt is made to read data from an older snapshot that does not exist in that snapshot, it is obtained by searching recursively up the hierarchy of more recent snapshots.

The following points determine whether it is appropriate to use a snapshot cascade:

- Deletion of a snapshot in the cascade takes time to copy the snapshot's data to the next snapshot in the cascade.
- The reliability of a snapshot in the cascade depends on all the newer snapshots in the chain. Thus the oldest snapshot in the cascade is the most vulnerable.
- Reading from a snapshot in the cascade may require data to be fetched from one or more other snapshots in the cascade.

For these reasons, it is recommended that you do not attempt to use a snapshot cascade with applications that need to remove or split snapshots from the cascade. In such cases, it may be more appropriate to create a snapshot of a snapshot as described in the following section.

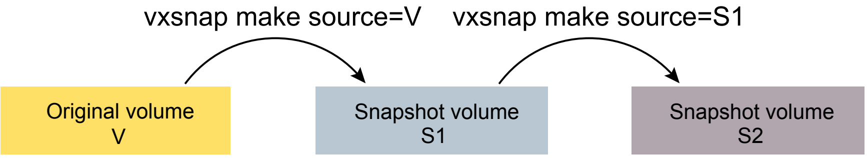
See "Adding a snapshot to a cascaded snapshot hierarchy" on page 412.

Note: Only unsynchronized full-sized or space-optimized instant snapshots are usually cascaded. It is of little utility to create cascaded snapshots if the `infrontof` snapshot volume is fully synchronized (as, for example, with break-off type snapshots).

Creating a snapshot of a snapshot

Figure 17-4 creation of a snapshot of an existing snapshot.

Figure 17-4 Creating a snapshot of a snapshot



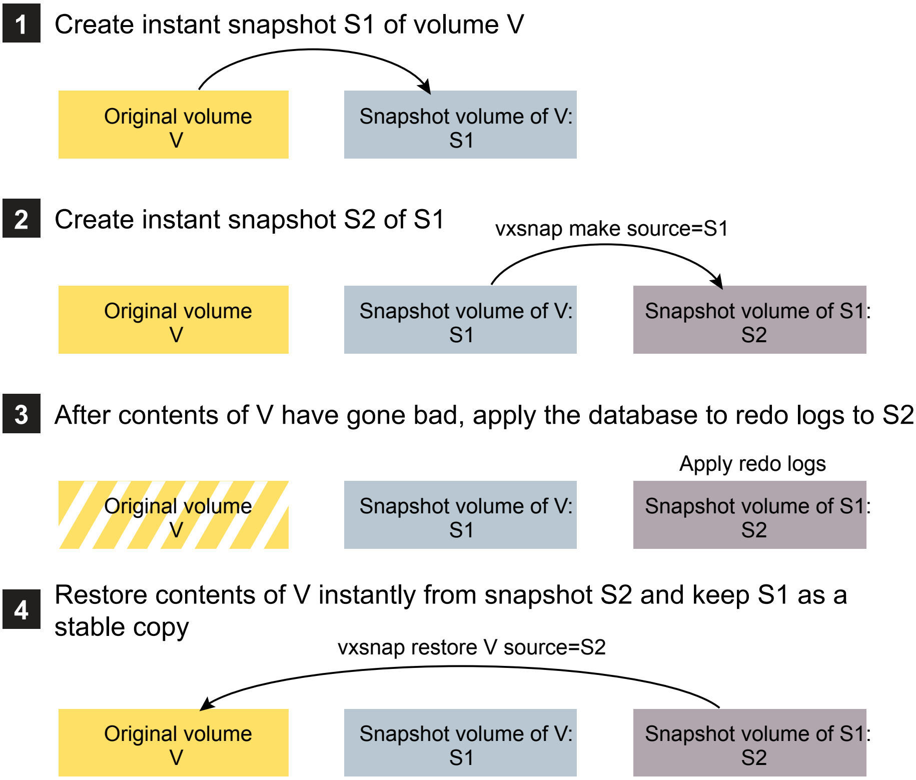
Even though the arrangement of the snapshots in this figure appears similar to a snapshot cascade, the relationship between the snapshots is not recursive. When reading from the snapshot `S2`, data is obtained directly from the original volume, `V`, if it does not exist in `S1` itself.

See Figure 17-3 on page 425.

Such an arrangement may be useful if the snapshot volume, `S1`, is critical to the operation. For example, `S1` could be used as a stable copy of the original volume, `V`. The additional snapshot volume, `S2`, can be used to restore the original volume if that volume becomes corrupted. For a database, you might need to replay a redo log on `S2` before you could use it to restore `V`.

Figure 17-5 shows the sequence of steps that would be required to restore a database.

Figure 17-5 Using a snapshot of a snapshot to restore a database

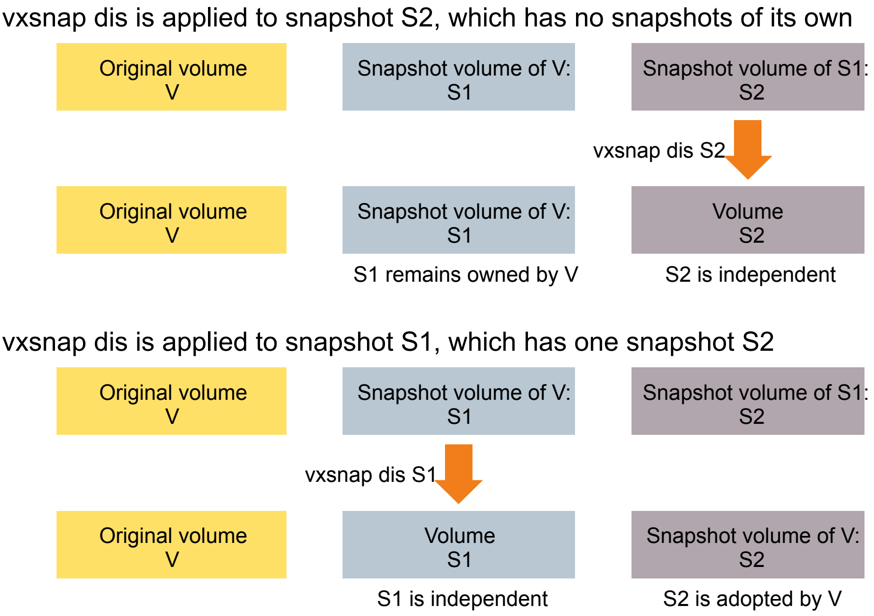

1
Create instant snapshot S1 of volume V
Original volume V
Snapshot volume of V: S1
2
Create instant snapshot S2 of S1
vxsnap make source=S1
Original volume V
Snapshot volume of V: S1
Snapshot volume of S1: S2
3
After contents of V have gone bad, apply the database to redo logs to S2
Apply redo logs
Original volume V
Snapshot volume of V: S1
Snapshot volume of S1: S2
4
Restore contents of V instantly from snapshot S2 and keep S1 as a stable copy
vxsnap restore V source=S2
Original volume V
Snapshot volume of V: S1
Snapshot volume of S1: S2

If you have configured snapshots in this way, you may wish to make one or more of the snapshots into independent volumes. There are two `vxsnap` commands that you can use to do this:

- `vxsnap dis` dissociates a snapshot and turns it into an independent volume. The snapshot to be dissociated must have been fully synchronized from its parent. If a snapshot volume has a child snapshot volume, the child must also have been fully synchronized. If the command succeeds, the child snapshot becomes a snapshot of the original volume.

  Figure 17-6 shows the effect of applying the `vxsnap dis` command to snapshots with and without dependent snapshots.

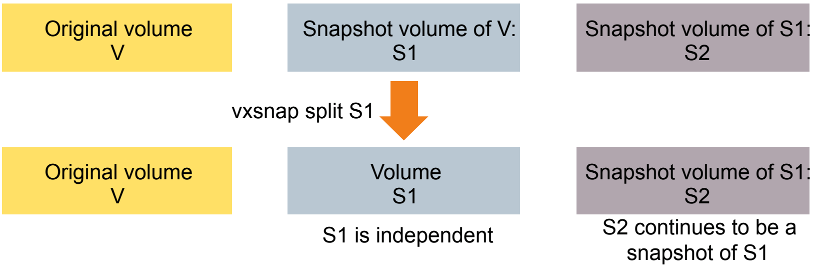
Figure 17-6 Dissociating a snapshot volume



- `vxsnap split` dissociates a snapshot and its dependent snapshots from its parent volume. The snapshot that is to be split must have been fully synchronized from its parent volume.

  Figure 17-7 shows the operation of the `vxsnap split` command.

Figure 17-7 Splitting snapshots



Creating multiple snapshots

To make it easier to create snapshots of several volumes at the same time, both the `vxsnap make` and `vxassist snapshot` commands accept more than one volume name as their argument.

For traditional snapshots, you can create snapshots of all the volumes in a single disk group by specifying the option `-o allvols` to the `vxassist snapshot` command.

By default, each replica volume is named *`SNAPnumber-volume`*, where number is a unique serial number, and *volume* is the name of the volume for which a snapshot is being taken. This default can be overridden by using the option `-o` *`name=pattern`*.

See the `vxassist`(1M) manual page.

See the `vxsnap`(1M) manual page.

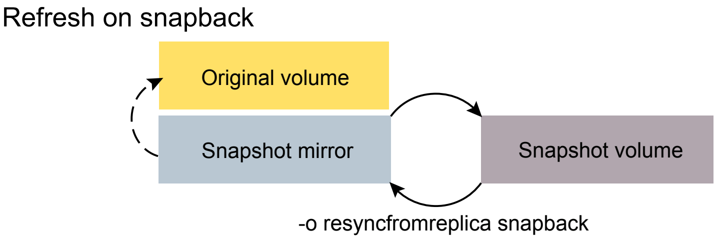
You can create a snapshot of all volumes that form a logical group; for example, all the volumes that conform to a database instance.

Restoring the original volume from a snapshot

For traditional snapshots, the snapshot plex is resynchronized from the data in the original volume during a `vxassist snapback` operation.

Figure 17-8 shows an alternative where the snapshot overwrites the original volume.

Figure 17-8 Resynchronizing an original volume from a snapshot



Specifying the option `-o resyncfromreplica` to `vxassist` resynchronizes the original volume from the data in the snapshot.

Warning: The original volume must not be in use during a `snapback` operation that specifies the option `-o resyncfromreplica` to resynchronize the volume from a snapshot. Stop any application, such as a database, and unmount any file systems that are configured to use the volume.

For instant snapshots, the `vxsnap restore` command may be used to restore the contents of the original volume from an instant snapshot or from a volume derived from an instant snapshot. The volume that is used to restore the original volume can either be a true backup of the contents of the original volume at some point in time, or it may have been modified in some way (for example, by applying a database log replay or by running a file system checking utility such as `fsck`). All synchronization of the contents of this backup must have been completed before the original volume can be restored from it. The original volume is immediately available for use while its contents are being restored.

See "Restoring a volume from an instant space-optimized snapshot" on page 415.

Adding a version 0 DCO and DCO volume

To put Persistent FastResync into effect for a volume, a data change object (DCO) and DCO volume must be associated with that volume. After you add a DCO object and DCO volume to a volume, you can enable Persistent FastResync on the volume.

Note: You need a FastResync license key to use the FastResync feature. Even if you do not have a license, you can configure a DCO object and DCO volume so that snap objects are associated with the original and snapshot volumes.

The procedure in this section describes adding a version 0 layout DCO. A version 0 DCO layout supports traditional (third-mirror break-off) snapshots that are administered with the `vxassist` command. A version 0 DCO layout does not support full-sized or space-optimized instant snapshots.

To add a DCO object and DCO volume to an existing volume

1 Ensure that the disk group containing the existing volume has at least disk group version 90. To check the version of a disk group:

```
# vxdg list diskgroup
```

If required, upgrade the disk group to the latest version:

```
# vxdg upgrade diskgroup
```

2 Turn off Non-Persistent FastResync on the original volume if it is currently enabled:

```
# vxvol [-g diskgroup] set fastresync=off volume
```

If you are uncertain about which volumes have Non-Persistent FastResync enabled, use the following command to obtain a listing of such volumes.

Note: The ! character is a special character in some shells. The following example shows how to escape it in a bash shell.

```
# vxprint [-g diskgroup] -F "%name" \
  -e "v_fastresync=on && \!v_hasdcolog"
```

3 Add a DCO and DCO volume to the existing volume (which may already have dirty region logging (DRL) enabled):

```
# vxassist [-g diskgroup] addlog volume logtype=dco \
  [ndcomirror=number] [dcolen=size] [storage_attributes]
```

For non-layered volumes, the default number of plexes in the mirrored DCO volume is equal to the lesser of the number of plexes in the data volume or 2. For layered volumes, the default number of DCO plexes is always 2. If required, use the `ndcomirror` attribute to specify a different number. It is recommended that you configure as many DCO plexes as there are existing data and snapshot plexes in the volume. For example, specify `ndcomirror=3` when adding a DCO to a 3-way mirrored volume.

The default size of each plex is 132 blocks. You can use the `dcolen` attribute to specify a different *size*. If specified, the size of the plex must be an integer multiple of 33 blocks from 33 up to a maximum of 2112 blocks.

You can specify `vxassist`-style storage attributes to define the disks that can or cannot be used for the plexes of the DCO volume.

See "Specifying storage for version 0 DCO plexes" on page 432.

Specifying storage for version 0 DCO plexes

If the disks that contain volumes and their snapshots are to be moved or split into different disk groups, the disks that contain their respective DCO plexes must be able to accompany them. By default, VxVM attempts to place version 0 DCO plexes on the same disks as the data plexes of the parent volume. However, this may be impossible if there is insufficient space available on those disks. In this case, VxVM uses any available space on other disks in the disk group. If the DCO plexes are placed on disks which are used to hold the plexes of other volumes, this may cause problems when you subsequently attempt to move volumes into other disk groups.

You can use storage attributes to specify explicitly which disks to use for the DCO plexes. If possible, specify the same disks as those on which the volume is configured.

For example, to add a DCO object and DCO volume with plexes on `mydg05` and `mydg06`, and a plex size of 264 blocks to the volume, `myvol`, in the disk group, `mydg`, use the following command:

```
# vxassist -g mydg addlog myvol logtype=dco dcolen=264 mydg05 mydg06
```

To view the details of the DCO object and DCO volume that are associated with a volume, use the `vxprint` command. The following is partial `vxprint` output for the volume named `vol1` (the TUTIL0 and PUTIL0 columns are omitted for clarity):

```
TY  NAME          ASSOC         KSTATE    LENGTH   PLOFFS   STATE  ...
v   vol1          fsgen         ENABLED   1024     -        ACTIVE
pl  vol1-01       vol1          ENABLED   1024     -        ACTIVE
sd  disk01-01     vol1-01       ENABLED   1024     0        -
pl  vol1-02       vol1          ENABLED   1024     -        ACTIVE
sd  disk02-01     vol1-02       ENABLED   1024     0        -
dc  vol1_dco      vol1          -         -        -        -
v   vol1_dcl      gen           ENABLED   132      -        ACTIVE
pl  vol1_dcl-01   vol1_dcl      ENABLED   132      -        ACTIVE
sd  disk03-01     vol1_dcl-01   ENABLED   132      0        -
pl  vol1_dcl-02   vol1_dcl      ENABLED   132      -        ACTIVE
sd  disk04-01     vol1_dcl-02   ENABLED   132        0        -
```

In this output, the DCO object is shown as `vol1_dco`, and the DCO volume as `vol1_dcl` with 2 plexes, `vol1_dcl-01` and `vol1_dcl-02`.

If required, you can use the `vxassist move` command to relocate DCO plexes to different disks. For example, the following command moves the plexes of the DCO volume, `vol1_dcl`, for volume `vol1` from `disk03` and `disk04` to `disk07` and `disk08`.

Note: The ! character is a special character in some shells. The following example shows how to escape it in a bash shell.

```
# vxassist -g mydg move vol1_dcl \!disk03 \!disk04 disk07 disk08
```

See “Moving DCO volumes between disk groups” on page 648.

See the `vxassist`(1M) manual page.

Removing a version 0 DCO and DCO volume

To dissociate a version 0 DCO object, DCO volume and any snap objects from a volume, use the following command:

```
# vxassist [-g diskgroup] remove log volume logtype=dco
```

This completely removes the DCO object, DCO volume and any snap objects. It also has the effect of disabling FastResync for the volume.

Alternatively, you can use the `vxdco` command to the same effect:

```
# vxdco [-g diskgroup] [-o rm] dis dco_obj
```

The default name of the DCO object, *dco_obj*, for a volume is usually formed by appending the string `_dco` to the name of the parent volume. To find out the name of the associated DCO object, use the `vxprint` command on the volume.

To dissociate, but not remove, the DCO object, DCO volume and any snap objects from the volume, `myvol`, in the disk group, `mydg`, use the following command:

```
# vxdco -g mydg dis myvol_dco
```

This form of the command dissociates the DCO object from the volume but does not destroy it or the DCO volume. If the `-o rm` option is specified, the DCO object, DCO volume and its plexes, and any snap objects are also removed.

Warning: Dissociating a DCO and DCO volume disables Persistent FastResync on the volume. A full resynchronization of any remaining snapshots is required when they are snapped back.

See the `vxassist`(1M) manual page.

See the `vxdco`(1M) manual pages.

Reattaching a version 0 DCO and DCO volume

If a version 0 DCO object and DCO volume are not removed by specifying the `-o rm` option to `vxdco`, they can be reattached to the parent volume using the following command:

```
# vxdco [-g diskgroup] att volume dco_obj
```

For example, to reattach the DCO object, `myvol_dco`, to the volume, `myvol`, use the following command:

```
# vxdco -g mydg att myvol myvol_dco
```

See the `vxdco`(1M) manual page.

Administering Storage Checkpoints

This chapter includes the following topics:

About Storage Checkpoints

Veritas File System (VxFS) provides a Storage Checkpoint feature that quickly creates a persistent image of a file system at an exact point in time. Storage Checkpoints significantly reduce I/O overhead by identifying and maintaining only the file system blocks that have changed since the last Storage Checkpoint or backup via a copy-on-write technique.

See “Copy-on-write” on page 368.

Storage Checkpoints provide:

- Persistence through reboots and crashes.
- The ability for data to be immediately writeable by preserving the file system metadata, the directory hierarchy, and user data.

Storage Checkpoints are actually data objects that are managed and controlled by the file system. You can create, remove, and rename Storage Checkpoints because they are data objects with associated names.

See “How a Storage Checkpoint works” on page 366.

Unlike a disk-based mirroring technology that requires a separate storage space, Storage Checkpoints minimize the use of disk space by using a Storage Checkpoint within the same free space available to the file system.

After you create a Storage Checkpoint of a mounted file system, you can also continue to create, remove, and update files on the file system without affecting the logical image of the Storage Checkpoint. A Storage Checkpoint preserves not only the name space (directory hierarchy) of the file system, but also the user data as it existed at the moment the file system image was captured.

You can use a Storage Checkpoint in many ways. For example, you can use them to:

- Create a stable image of the file system that can be backed up to tape.
- Provide a mounted, on-disk backup of the file system so that end users can restore their own files in the event of accidental deletion. This is especially useful in a home directory, engineering, or email environment.
- Create a copy of an application's binaries before installing a patch to allow for rollback in case of problems.
- Create an on-disk backup of the file system in that can be used in addition to a traditional tape-based backup to provide faster backup and restore capabilities.
- Test new software on a point-in-time image of the primary fileset without jeopardizing the live data in the current primary fileset by mounting the Storage Checkpoints as writable.

Storage Checkpoint administration

Storage Checkpoint administrative operations require the `fsckptadm` utility.

See the `fsckptadm`(1M) manual page.

You can use the `fsckptadm` utility to create and remove Storage Checkpoints, change attributes, and ascertain statistical data. Every Storage Checkpoint has an associated name, which allows you to manage Storage Checkpoints; this name is limited to 127 characters and cannot contain a colon (:).

See “Creating a Storage Checkpoint” on page 437.

See “Removing a Storage Checkpoint” on page 438.

Storage Checkpoints require some space for metadata on the volume or set of volumes specified by the file system allocation policy or Storage Checkpoint allocation policy. The `fsckptadm` utility displays an error if the volume or set of volumes does not have enough free space to contain the metadata. You can roughly approximate the amount of space required by the metadata using a method that depends on the disk layout version of the file system.

For disk layout version 7 or later, multiply the number of inodes by 1 byte, and add 1 or 2 megabytes to get the approximate amount of space required. You can determine the number of inodes with the fsckptadm utility.

Use the fsvoladm command to determine if the volume set has enough free space.

See the fsvoladm(1M) manual page.

The following example lists the volume sets and displays the storage sizes in human-friendly units:

```
# fsvoladm -H list /mnt0
devid   size     used    avail   name
0       20 GB    10 GB   10 GB   vol1
1       30 TB    10 TB   20 TB   vol2
```

Creating a Storage Checkpoint

The following example shows the creation of a nodata Storage Checkpoint named thu_7pm on /mnt0 and lists all Storage Checkpoints of the /mnt0 file system:

```
# fsckptadm -n create thu_7pm /mnt0
# fsckptadm list /mnt0
/mnt0
thu_7pm:
 ctime   = Thu 3 Mar 2005 7:00:17 PM PST
 mtime   = Thu 3 Mar 2005 7:00:17 PM PST
 flags   = nodata, largefiles
```

The following example shows the creation of a removable Storage Checkpoint named thu_8pm on /mnt0 and lists all Storage Checkpoints of the /mnt0 file system:

```
# fsckptadm -r create thu_8pm /mnt0
# fsckptadm list /mnt0
/mnt0
thu_8pm:
 ctime   = Thu 3 Mar 2005 8:00:19 PM PST
 mtime   = Thu 3 Mar 2005 8:00:19 PM PST
 flags   = largefiles, removable
thu_7pm:
 ctime   = Thu 3 Mar 2005 7:00:17 PM PST
 mtime   = Thu 3 Mar 2005 7:00:17 PM PST
 flags   = nodata, largefiles
```

Removing a Storage Checkpoint

You can delete a Storage Checkpoint by specifying the remove keyword of the fsckptadm command. Specifically, you can use either the synchronous or asynchronous method of removing a Storage Checkpoint; the asynchronous method is the default method. The synchronous method entirely removes the Storage Checkpoint and returns all of the blocks to the file system before completing the fsckptadm operation. The asynchronous method simply marks the Storage Checkpoint for removal and causes fsckptadm to return immediately. At a later time, an independent kernel thread completes the removal operation and releases the space used by the Storage Checkpoint.

In this example, /mnt0 is a mounted VxFS file system. This example shows the asynchronous removal of the Storage Checkpoint named thu_8pm and synchronous removal of the Storage Checkpoint named thu_7pm. This example also lists all the Storage Checkpoints remaining on the /mnt0 file system after the specified Storage Checkpoint is removed:

```
# fsckptadm remove thu_8pm /mnt0
# fsckptadm list /mnt0
/mnt0
thu_7pm:
 ctime   = Thu 3 Mar 2005 7:00:17 PM PST
 mtime   = Thu 3 Mar 2005 7:00:17 PM PST
 flags   = nodata, largefiles
# fsckptadm -s remove thu_7pm /mnt0
# fsckptadm list /mnt0
/mnt0
```

Accessing a Storage Checkpoint

You can mount Storage Checkpoints using the mount command with the mount option -o ckpt=*ckpt_name*.

See the mount_vxfs(1M) manual page.

Observe the following rules when mounting Storage Checkpoints:

- Storage Checkpoints are mounted as read/write Storage Checkpoints by default.
- If a Storage Checkpoint is currently mounted as a read-only Storage Checkpoint, you can remount it as a writable Storage Checkpoint using the -o remount option.
- To mount a Storage Checkpoint of a file system, first mount the file system itself.
- To unmount a file system, first unmount all of its Storage Checkpoints.

Warning: If you create a Storage Checkpoint for backup purposes, do not mount it as a writable Storage Checkpoint. You will lose the point-in-time image if you accidently write to the Storage Checkpoint.

If older Storage Checkpoints already exist, write activity to a writable Storage Checkpoint can generate copy operations and increased space usage in the older Storage Checkpoints.

A Storage Checkpoint is mounted on a special pseudo device. This pseudo device does not exist in the system name space; the device is internally created by the system and used while the Storage Checkpoint is mounted. The pseudo device is removed after you unmount the Storage Checkpoint. A pseudo device name is formed by appending the Storage Checkpoint name to the file system device name using the colon character (:) as the separator.

For example, if a Storage Checkpoint named `may_23` belongs to the file system residing on the special device `/dev/vx/dsk/fsvol/vol1`, the Storage Checkpoint pseudo device name is:

```
/dev/vx/dsk/fsvol/vol1:may_23
```

- To mount the Storage Checkpoint named `may_23` as a read-only Storage Checkpoint on directory `/fsvol_may_23`, type:

  ```
  # mount -t vxfs -o ckpt=may_23 /dev/vx/dsk/fsvol/vol1:may_23 \
  /fsvol_may_23
  ```

  Note: The `vol1` file system must already be mounted before the Storage Checkpoint can be mounted.

- To remount the Storage Checkpoint named `may_23` as a writable Storage Checkpoint, type:

  ```
  # mount -t vxfs -o ckpt=may_23,remount,rw \
  /dev/vx/dsk/fsvol/vol1:may_23 /fsvol_may_23
  ```

- To mount this Storage Checkpoint automatically when the system starts up, put the following entries in the `/etc/fstab` file:

```
Device-Special-File  Mount-Point    fstype  options     backup-    pass-
                                                        frequency  number
/dev/vx/dsk/fsvol/   /fsvol         vxfs    defaults    0          0
  vol1
```

```
/dev/vx/dsk/fsvol/    /fsvol_may_23   vxfs    ckpt=may_23 0          0
  vol1:may_23
```

- To mount a Storage Checkpoint of a cluster file system, you must also use the `-o cluster` option:

  ```
  # mount -t vxfs -o cluster,ckpt=may_23 \
  /dev/vx/dsk/fsvol/vol1:may_23 /fsvol_may_23
  ```

  You can only mount a Storage Checkpoint cluster-wide if the file system that the Storage Checkpoint belongs to is also mounted cluster-wide. Similarly, you can only mount a Storage Checkpoint locally if the file system that the Storage Checkpoint belongs to is mounted locally.

You can unmount Storage Checkpoints using the `umount` command.

See the `umount`(1M) manual page.

Storage Checkpoints can be unmounted by the mount point or pseudo device name:

```
# umount /fsvol_may_23
# umount /dev/vx/dsk/fsvol/vol1:may_23
```

Note: You do not need to run the `fsck` utility on Storage Checkpoint pseudo devices because pseudo devices are part of the actual file system.

Converting a data Storage Checkpoint to a nodata Storage Checkpoint

A nodata Storage Checkpoint does not contain actual file data. Instead, this type of Storage Checkpoint contains a collection of markers indicating the location of all the changed blocks since the Storage Checkpoint was created.

See “Types of Storage Checkpoints” on page 370.

You can use either the synchronous or asynchronous method to convert a data Storage Checkpoint to a nodata Storage Checkpoint; the asynchronous method is the default method. In a synchronous conversion, `fsckptadm` waits for all files to undergo the conversion process to “nodata" status before completing the operation. In an asynchronous conversion, `fsckptadm` returns immediately and marks the Storage Checkpoint as a nodata Storage Checkpoint even though the Storage Checkpoint's data blocks are not immediately returned to the pool of free blocks in the file system. The Storage Checkpoint deallocates all of its file data blocks in the background and eventually returns them to the pool of free blocks in the file system.

If all of the older Storage Checkpoints in a file system are nodata Storage Checkpoints, use the synchronous method to convert a data Storage Checkpoint to a nodata Storage Checkpoint. If an older data Storage Checkpoint exists in the file system, use the asynchronous method to mark the Storage Checkpoint you want to convert for a delayed conversion. In this case, the actual conversion will continue to be delayed until the Storage Checkpoint becomes the oldest Storage Checkpoint in the file system, or all of the older Storage Checkpoints have been converted to nodata Storage Checkpoints.

Note: You cannot convert a nodata Storage Checkpoint to a data Storage Checkpoint because a nodata Storage Checkpoint only keeps track of the location of block changes and does not save the content of file data blocks.

Showing the difference between a data and a nodata Storage Checkpoint

The following example shows the difference between data Storage Checkpoints and nodata Storage Checkpoints.

Note: A nodata Storage Checkpoint does not contain actual file data.

To show the difference between Storage Checkpoints

1 Create a file system and mount it on `/mnt0`, as in the following example:

```
# mkfs -t vxfs /dev/vx/rdsk/dg1/test0

version 17 layout
62914560 sectors, 31457280 blocks of size 1024, log size 65536 blocks
rcq size    2048 blocks
largefiles  supported
maxlink     supported
WORM not supported
maxts supported

size  65536 blocks, largefiles supported
# mount -t vxfs /dev/vx/dsk/dg1/test0 /mnt0
```

2 Create a small file with a known content, as in the following example:

```
# echo "hello, world" > /mnt0/file
```

3 Create a Storage Checkpoint and mount it on `/mnt0@5_30pm`, as in the following example:

```
# fsckptadm create ckpt@5_30pm /mnt0
# mkdir /mnt0@5_30pm
# mount -t vxfs -o ckpt=ckpt@5_30pm \
  /dev/vx/dsk/dg1/test0:ckpt@5_30pm  /mnt0@5_30pm
```

4 Examine the content of the original file and the Storage Checkpoint file:

```
# cat /mnt0/file
hello, world
# cat /mnt0@5_30pm/file
hello, world
```

5 Change the content of the original file:

```
# echo "goodbye" > /mnt0/file
```

6 Examine the content of the original file and the Storage Checkpoint file. The original file contains the latest data while the Storage Checkpoint file still contains the data at the time of the Storage Checkpoint creation:

```
# cat /mnt0/file
goodbye
# cat /mnt0@5_30pm/file
hello, world
```

7 Unmount the Storage Checkpoint, convert the Storage Checkpoint to a nodata Storage Checkpoint, and mount the Storage Checkpoint again:

```
# umount /mnt0@5_30pm
# fsckptadm -s set nodata ckpt@5_30pm /mnt0
# mount -t vxfs -o ckpt=ckpt@5_30pm \
/dev/vx/dsk/dg1/test0:ckpt@5_30pm /mnt0@5_30pm
```

8 Examine the content of both files. The original file must contain the latest data:

```
# cat /mnt0/file
goodbye
```

You can traverse and read the directories of the nodata Storage Checkpoint; however, the files contain no data, only markers to indicate which block of the file has been changed since the Storage Checkpoint was created:

```
# ls -l /mnt0@5_30pm/file
-rw-r--r--   1 root     other 13 Jul 13 17:13 \
# cat /mnt0@5_30pm/file
cat: /mnt0@5_30pm/file: Input/output error
```

Converting multiple Storage Checkpoints

You can convert Storage Checkpoints to nodata Storage Checkpoints when dealing with older Storage Checkpoints on the same file system.

To convert multiple Storage Checkpoints

1. Create a file system and mount it on `/mnt0`:

```
# mkfs -t vxfs /dev/vx/rdsk/dg1/test0
version 16 layout
134217728 sectors, 67108864 blocks of size 1024, log size 65536 blocks
rcq size 4096 blocks
largefiles supported
maxlink    supported
# mount -t vxfs /dev/vx/dsk/dg1/test0 /mnt0
```

2. Create four data Storage Checkpoints on this file system, note the order of creation, and list them:

```
# fsckptadm create oldest /mnt0
# fsckptadm create older /mnt0
# fsckptadm create old /mnt0
# fsckptadm create latest /mnt0
# fsckptadm list /mnt0
  /mnt0
  latest:
    ctime                =  Mon 26 Jul 11:56:55 2004
    mtime                =  Mon 26 Jul 11:56:55 2004
    flags                =  largefiles
  old:
    ctime                =  Mon 26 Jul 11:56:51 2004
    mtime                =  Mon 26 Jul 11:56:51 2004
    flags                =  largefiles
  older:
    ctime                =   Mon 26 Jul 11:56:46 2004
    mtime                =  Mon 26 Jul 11:56:46 2004
    flags                =  largefiles
  oldest:
    ctime                =  Mon 26 Jul 11:56:41 2004
    mtime                =  Mon 26 Jul 11:56:41 2004
    flags                =  largefiles
```

3 Try to convert synchronously the `latest` Storage Checkpoint to a nodata Storage Checkpoint. The attempt will fail because the Storage Checkpoints older than the `latest` Storage Checkpoint are data Storage Checkpoints, namely the Storage Checkpoints `old`, `older`, and `oldest`:

```
# fsckptadm -s set nodata latest /mnt0
UX:vxfs fsckptadm: ERROR: V-3-24632: Storage Checkpoint
set failed on latest.  File exists (17)
```

4 You can instead convert the `latest` Storage Checkpoint to a nodata Storage Checkpoint in a delayed or asynchronous manner.

```
# fsckptadm set nodata latest /mnt0
```

5 List the Storage Checkpoints, as in the following example. You will see that the `latest` Storage Checkpoint is marked for conversion in the future.

```
# fsckptadm list /mnt0
  /mnt0
  latest:
    ctime                = Mon 26 Jul 11:56:55 2004
    mtime                = Mon 26 Jul 11:56:55
    flags                = nodata, largefiles, delayed
  old:
    ctime                = Mon 26 Jul 11:56:51 2004
    mtime                = Mon 26 Jul 11:56:51 2004
    flags                = largefiles
  older:
    ctime                = Mon 26 Jul 11:56:46 2004
    mtime                = Mon 26 Jul 11:56:46 2004
    flags                = largefiles
  oldest:
    ctime                = Mon 26 Jul 11:56:41 2004
    mtime                = Mon 26 Jul 11:56:41 2004
    flags                = largefiles
```

Creating a delayed nodata Storage Checkpoint

You can create a Storage Checkpoint as a delayed nodata Storage Checkpoint. The creation process detects the presence of the older data Storage Checkpoints and creates the Storage Checkpoint as a delayed nodata Storage Checkpoint. The following example procedure removes an existing Storage Checkpoint named `latest` and recreates the Storage Checkpoint as a delayed nodata Storage Checkpoint.

To create a delayed nodata Storage Checkpoint

1. Remove the `latest` Storage Checkpoint.

```
# fsckptadm remove latest /mnt0
# fsckptadm list /mnt0
/mnt0
old:
  ctime                = Mon 26 Jul 11:56:51 2004
  mtime                = Mon 26 Jul 11:56:51 2004
  flags                = largefiles
older:
  ctime                = Mon 26 Jul 11:56:46 2004
  mtime                = Mon 26 Jul 11:56:46 2004
  flags                = largefiles
oldest:
  ctime                = Mon 26 Jul 11:56:41 2004
  mtime                = Mon 26 Jul 11:56:41 2004
  flags                = largefiles
```

2. Recreate the `latest` Storage Checkpoint as a nodata Storage Checkpoint.

```
# fsckptadm -n create latest /mnt0
# fsckptadm list /mnt0
/mnt0
latest:
  ctime                = Mon 26 Jul 12:06:42 2004
  mtime                = Mon 26 Jul 12:06:42 2004
  flags                = nodata, largefiles, delayed
old:
  ctime                = Mon 26 Jul 11:56:51 2004
  mtime                = Mon 26 Jul 11:56:51 2004
  flags                = largefiles
older:
  ctime                = Mon 26 Jul 11:56:46 2004
  mtime                = Mon 26 Jul 11:56:46 2004
  flags                = largefiles
oldest:
  ctime                = Mon 26 Jul 11:56:41 2004
  mtime                = Mon 26 Jul 11:56:41 2004
  flags                = largefiles
```

3 Convert the `oldest` Storage Checkpoint to a nodata Storage Checkpoint because no older Storage Checkpoints exist that contain data in the file system.

Note: This step can be done synchronously.

```
# fsckptadm -s set nodata oldest /mnt0
# fsckptadm list /mnt0
/mnt0
latest:
  ctime                = Mon 26 Jul 12:06:42 2004
  mtime                = Mon 26 Jul 12:06:42 2004
  flags                = nodata, largefiles, delayed
old:
  ctime                = Mon 26 Jul 11:56:51 2004
  mtime                = Mon 26 Jul 11:56:51 2004
  flags                = largefiles
older:
  ctime                = Mon 26 Jul 11:56:46 2004
  mtime                = Mon 26 Jul 11:56:46 2004
  flags                = largefiles
oldest:
  ctime                = Mon 26 Jul 11:56:41 2004
  mtime                = Mon 26 Jul 11:56:41 2004
  flags                = nodata, largefiles
```

4 Remove the older and old Storage Checkpoints.

```
# fsckptadm remove older /mnt0
# fsckptadm remove old /mnt0
# fsckptadm list /mnt0
/mnt0
latest:
  ctime                = Mon 26 Jul 12:06:42 2004
  mtime                = Mon 26 Jul 12:06:42 2004
  flags                = nodata, largefiles
oldest:
  ctime                = Mon 26 Jul 11:56:41 2004
  mtime                = Mon 26 Jul 11:56:41 2004
  flags                = nodata, largefiles
```

Note: After you remove the older and old Storage Checkpoints, the latest Storage Checkpoint is automatically converted to a nodata Storage Checkpoint because the only remaining older Storage Checkpoint (oldest) is already a nodata Storage Checkpoint:

Enabling and disabling Storage Checkpoint visibility

You enable Storage Checkpoint visibility through the ckptautomnt mount option, which can be set to one of three values: off, ro, or rw. Because enabling Storage Checkpoint visibility prevents manual mounting of clones, the default value is off. Setting the option to ro causes all clones to be automounted as read-only, while rw causes all clones to be automounted as read/write.

If you take a Storage Checkpoint of an existing Storage Checkpoint (instead of the primary file set), the directory for the source Storage Checkpoint in .checkpoint functions as the mount point. For example, to take a Storage Checkpoint of the Storage Checkpoint clone1 in a file system mounted on /mnt, use the following command:

```
# fsckptadm create clone2 /mnt/.checkpoint/clone1
```

By default, Storage Foundation (SF) does not make inode numbers unique. However, you can specify the uniqueino mount option to enable the use of unique 64-bit inode numbers. You cannot change this option during a remount.

The following example enables Storage Checkpoint visibility by causing all clones to be automounted as read/write:

```
# mount -t vxfs -o ckptautomnt=rw /dev/vx/dsk/dg1/vol1 /mnt1
```

Storage Checkpoint space management considerations

Several operations, such as removing or overwriting a file, can fail when a file system containing Storage Checkpoints runs out of space. If the system cannot allocate sufficient space, the operation will fail.

Database applications usually preallocate storage for their files and may not expect a write operation to fail. During user operations such as `create` or `mkdir`, if the file system runs out of space, removable Storage Checkpoints are deleted. This ensures that applications can continue without interruptions due to lack of disk space. Non-removable Storage Checkpoints are not automatically removed under such `ENOSPC` conditions. Veritas recommends that you create only removable Storage Checkpoints. However, during certain administrative operations, such as using the `fsadm` command, using the `qiomkfile` command, and creating a Storage Checkpoint with the `fsckptadm` command, even if the file system runs out of space, removable Storage Checkpoints are not deleted.

When the kernel automatically removes the Storage Checkpoints, it applies the following policies:

- Remove as few Storage Checkpoints as possible to complete the operation.
- Never select a non-removable Storage Checkpoint.
- Select a nodata Storage Checkpoint only when data Storage Checkpoints no longer exist.
- Remove the oldest Storage Checkpoint first.
- Remove a Storage Checkpoint even if it is mounted. New operations on such a removed Storage Checkpoint fail with the appropriate error codes.
- If the oldest Storage Checkpoint is non-removable, then the oldest removable Storage Checkpoint is selected for removal. In such a case, data might be required to be pushed to a non-removable Storage Checkpoint, which might fail and result in the file system getting marked for a FULLFSCK. To prevent this occurrence, Veritas recommends that you only create removable Storage Checkpoints.

Restoring from a Storage Checkpoint

Mountable data Storage Checkpoints on a consistent and undamaged file system can be used by backup and restore applications to restore either individual files or an entire file system. Restoration from Storage Checkpoints can also help recover incorrectly modified files, but typically cannot recover from hardware damage or other file system integrity problems.

Note: For hardware or other integrity problems, Storage Checkpoints must be supplemented by backups from other media.

Files can be restored by copying the entire file from a mounted Storage Checkpoint back to the primary fileset. To restore an entire file system, you can designate a mountable data Storage Checkpoint as the primary fileset using the `fsckpt_restore` command.

See the `fsckpt_restore`(1M) manual page.

When using the `fsckpt_restore` command to restore a file system from a Storage Checkpoint, all changes made to that file system after that Storage Checkpoint's creation date are permanently lost. The only Storage Checkpoints and data preserved are those that were created at the same time, or before, the selected Storage Checkpoint's creation. The file system cannot be mounted at the time that `fsckpt_restore` is invoked.

Note: Individual files can also be restored very efficiently by applications using the `fsckpt_fbmap`(3) library function to restore only modified portions of a files data.

You can restore from a Storage Checkpoint only to a file system that has disk layout Version 6 or later.

The following example restores a file, `file1.txt`, which resides in your home directory, from the Storage Checkpoint `CKPT1` to the device `/dev/vx/dsk/dg1/vol-01`. The mount point for the device is `/home`.

To restore a file from a Storage Checkpoint

1. Create the Storage Checkpoint `CKPT1` of `/home`.

   ```
   $ fsckptadm create CKPT1 /home
   ```

2. Mount Storage Checkpoint `CKPT1` on the directory `/home/checkpoints/mar_4`.

   ```
   $ /opt/VRTS/bin/mount -o ckpt=CKPT1 /dev/vx/dsk/dg1/vol- \
   01:CKPT1 /home/checkpoints/mar_4
   ```

3 Delete the `file1.txt` file from your home directory.

```
$ cd /home/users/me
$ rm file1.txt
```

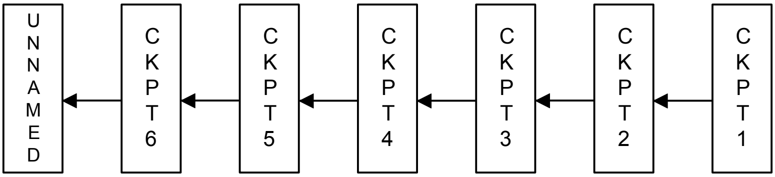
4 Go to the `/home/checkpoints/mar_4/users/me` directory, which contains the image of your home directory.

```
$ cd /home/checkpoints/mar_4/users/me
$ ls -l
-rw-r--r--   1  me   staff  14910   Mar 4   17:09   file1.txt
```

5 Copy the file `file1.txt` to your home directory.

```
$ cp file1.txt /home/users/me
$ cd /home/users/me
$ ls -l
-rw-r--r--    1 me   staff  14910   Mar 4  18:21   file1.txt
```

The following example restores a file system from the Storage Checkpoint `CKPT3`. The filesets listed before the restoration show an unnamed root fileset and six Storage Checkpoints.



To restore a file system from a Storage Checkpoint

1 Run the `fsckpt_restore` command:

```
# fsckpt_restore -l /dev/vx/dsk/dg1/vol2
/dev/vx/dsk/dg1/vol2:
UNNAMED:
  ctime        = Thu 08 May 2004 06:28:26 PM PST
  mtime        = Thu 08 May 2004 06:28:26 PM PST
  flags        = largefiles, file system root
CKPT6:
   ctime        = Thu 08 May 2004 06:28:35 PM PST
  mtime        = Thu 08 May 2004 06:28:35 PM PST
  flags        = largefiles
CKPT5:
  ctime        = Thu 08 May 2004 06:28:34 PM PST
  mtime        = Thu 08 May 2004 06:28:34 PM PST
  flags        = largefiles, nomount
CKPT4:
  ctime        = Thu 08 May 2004 06:28:33 PM PST
  mtime        = Thu 08 May 2004 06:28:33 PM PST
  flags        = largefiles
CKPT3:
  ctime        = Thu 08 May 2004 06:28:36 PM PST
  mtime        = Thu 08 May 2004 06:28:36 PM PST
  flags        = largefiles
CKPT2:
  ctime        = Thu 08 May 2004 06:28:30 PM PST
  mtime        = Thu 08 May 2004 06:28:30 PM PST
  flags        = largefiles
CKPT1:
  ctime        = Thu 08 May 2004 06:28:29 PM PST
  mtime        = Thu 08 May 2004 06:28:29 PM PST
  flags        = nodata, largefiles
```

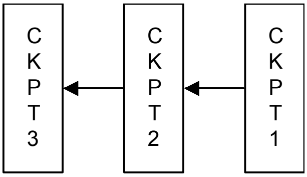
2 In this example, select the Storage Checkpoint `CKPT3` as the new root fileset:

```
Select Storage Checkpoint for restore operation
 or <Control/D> (EOF) to exit
 or <Return> to list Storage Checkpoints: CKPT3
CKPT3:
  ctime        = Thu 08 May 2004 06:28:31 PM PST
  mtime        = Thu 08 May 2004 06:28:36 PM PST
  flags        = largefiles
UX:vxfs fsckpt_restore: WARNING: V-3-24640: Any file system
changes or Storage Checkpoints made after
Thu 08  May 2004 06:28:31 PM PST will be lost.
```

3 Type **y** to restore the file system from CKPT3:

```
Restore the file system from Storage Checkpoint CKPT3 ?
(ynq) y
(Yes)
UX:vxfs fsckpt_restore: INFO: V-3-23760: File system
restored from CKPT3
```

If the filesets are listed at this point, it shows that the former UNNAMED root fileset and CKPT6, CKPT5, and CKPT4 were removed, and that CKPT3 is now the primary fileset. CKPT3 is now the fileset that will be mounted by default.



4 Run the fsckpt_restore command:

```
# fsckpt_restore -l /dev/vx/dsk/dg1/vol2
/dev/vx/dsk/dg1/vol2:
CKPT3:
  ctime       = Thu 08 May 2004 06:28:31 PM PST
  mtime       = Thu 08 May 2004 06:28:36 PM PST
  flags       = largefiles, file system root
CKPT2:
  ctime       = Thu 08 May 2004 06:28:30 PM PST
  mtime       = Thu 08 May 2004 06:28:30 PM PST
  flags       = largefiles
CKPT1:
  ctime       = Thu 08 May 2004 06:28:29 PM PST
  mtime       = Thu 08 May 2004 06:28:29 PM PST
  flags       = nodata, largefiles
Select Storage Checkpoint for restore operation
 or <Control/D> (EOF) to exit
 or <Return> to list Storage Checkpoints:
```

Storage Checkpoint quotas

VxFS provides options to the `fsckptadm` command interface to administer Storage Checkpoint quotas. Storage Checkpoint quotas set the following limits on the amount of space used by all Storage Checkpoints of a primary file set:

hard limit — An absolute limit that cannot be exceeded. If a hard limit is exceeded, all further allocations on any of the Storage Checkpoints fail, but existing Storage Checkpoints are preserved.

soft limit — Must be lower than the hard limit. If a soft limit is exceeded, no new Storage Checkpoints can be created. The number of blocks used must return below the soft limit before more Storage Checkpoints can be created. An alert and console message are generated.

In case of a hard limit violation, various solutions are possible, enacted by specifying or not specifying the `-f` option for the `fsckptadm` utility.

See the `fsckptadm`(1M) manual page.

Specifying or not specifying the `-f` option has the following effects:

- If the `-f` option is not specified, one or many removable Storage Checkpoints are deleted to make space for the operation to succeed. This is the default solution.
- If the `-f` option is specified, all further allocations on any of the Storage Checkpoints fail, but existing Storage Checkpoints are preserved.

Note: Sometimes if a file is removed while it is opened by another process, the removal process is deferred until the last close. Because the removal of a file may trigger pushing data to a “downstream" Storage Checkpoint (that is, the next older Storage Checkpoint), a fileset hard limit quota violation may occur. In this scenario, the hard limit is relaxed to prevent an inode from being marked bad. This is also true for some asynchronous inode operations.

Chapter 19

Administering FileSnaps

This chapter includes the following topics:

FileSnap creation

A single thread creating FileSnaps of the same file can create over ten thousand snapshots per minute. FileSnaps can be used for fast provisioning of new virtual machines by cloning a virtual machine golden image, where the golden image is stored as a file in a VxFS file system or Storage Foundation Cluster File System High Availability (SFCFSHA) file system, which is used as a data store for a virtual environment.

FileSnap creation over Network File System

You can create a FileSnap over Network File System (NFS) by creating a hard link from an existing file to a new file with the extension “::snap:vxfs:”. For example, the following command creates a new file named `file1`, but instead of making `file1` a hard link of `file2`, `file1` will be a FileSnap so that the link count of `file2` will not change:

```
# ln file1 file2::snap:vxfs:
```

This is the equivalent of using the following command:

```
# vxfilesnap -p file1 file2
```

The new file has the same attributes as the old file and shares all of the old file's extents.

An application that uses this namespace extension should check if the file created has the namespace extension, such as `file1::snap:vxfs:` instead of `file1`. This indicates the namespace extension is not supported, either because the file system exported over NFS is not VxFS, the file system is an older version of VxFS, or the file system does not have a license for FileSnaps.

As with the `vxfilesnap` command, FileSnaps must be made within a single file set.

Using FileSnaps

Table 19-1 provides a list of Veritas File System (VxFS) commands that enable you to administer FileSnaps.

Table 19-1

Command	Functionality
`fiostat`	The `fiostat` command has the `-S shared` option to display statistics for each interval. Otherwise, the command displays the accumulated statistics for the entire time interval.
`fsadm`	The `fsadm` command has the `-S` option to report shared block usage in the file system. You can use this option to find out the storage savings achieved through FileSnaps and how much real storage is required if all of the files are full copies. See the `fsadm_vxfs`(1M) manual page.
`fsmap`	The `fsmap` command has the `-c` option to report the count of the total number of physical blocks consumed by a file, and how many of those blocks might not be private to a given file. See the `fsmap`(1) manual page.
`mkfs`	Use the `mkfs` command to make a disk layout with a specific supported version file system by specifying `-o version=`*`supportedVersion`*. By default, the latest supported disk layout version is used. VxFS internally maintains a list of delayed operations on shared extent references and the size of this list (`rcqsize`) defaults to a value that is a function of the file system size, but can be changed when the file system is made. See the `mkfs_vxfs`(1M) manual page.

Table 19-1 *(continued)*

Command	Functionality
vxfilesnap	Use the vxfilesnap command to create a snapshot of a file or set of files or files in a directory. You can also use the vxfilesnap command to restore a older version of the file to the current file. See the vxfilesnap(1) manual page.
vxtunefs	The vxtunefs command supports an option to enable lazy copy-on-write tuneable, lazy_copyonwrite, on the file system, for better performance. See the vxtunefs(1M) manual page.

Using FileSnaps to create point-in-time copies of files

The key to obtaining maximum performance with FileSnaps is to minimize the copy-on-write overhead. You can achieved this by enabling lazy copy-on-write. Lazy copy-on-write is easy to enable and usually results in significantly better performance. If lazy copy-on-write is not a viable option for the use case under consideration, an efficient allocation of the source file can reduce the need of copy-on-write.

Using FileSnaps to provision virtual desktops

Virtual desktop infrastructure (VDI) operating system boot images are a good use case for FileSnaps. The parts of the boot images that can change are user profile, page files (or swap for UNIX/Linux) and application data. You should separate such data from boot images to minimize unsharing. You should allocate a single extent to the master boot image file.

The following example uses a 4 GB master boot image that has a single extent that will be shared by all snapshots.

```
# touch /vdi_images/master_image
# /opt/VRTS/bin/setext -r 4g -f chgsize /vdi_images/master_image
```

The master_image file can be presented as a disk device to the virtual machine for installing the operating system. Once the operating system is installed and configured, the file is ready for snapshots.

Using FileSnaps to optimize write intensive applications for virtual machines

When virtual machines are spawned to perform certain tasks that are write intensive, a significant amount of unsharing can take place. Veritas recommends that you optimize performance by enabling lazy copy-on-write. If the use case does not allow enabling lazy copy-on-write, with careful planning, you can reduce the occurrence of unsharing. The easiest way to reduce unsharing is to separate the application data to a file other than the boot image. If you cannot do this due to the nature of your applications, then you can take actions similar to the following example.

Assume that the disk space required for a boot image and the application data is 20 GB. Out of this, only 4 GB is used by the operating system and the remaining 16 GB is the space for applications to write. Any data or binaries that are required by each instance of the virtual machine can still be part of the first 4 GB of the shared extent. Since most of the writes are expected to take place on the 16 GB portion, you should allocate the master image in such a way that the 16 GB of space is not shared, as shown in the following commands:

```
  # touch /vdi_images/master_image
  # /opt/VRTS/bin/setext -r 4g -f chgsize /vdi_images/master_image
  # dd if=/dev/zero of=/vdi_images/master_image seek=20971520 \
bs=1024 count=1
```

The last command creates a 20 GB hole at the end of the file. Since holes do not have any extents allocated, the writes to hole do not need to be unshared.

Using FileSnaps to create multiple copies of data instantly

It is common to create one or more copies of production data for the purpose of generating reports, mining, and testing. These cases frequently update the copies of the data with the most current data, and one or more copies of the data always exists. FileSnaps can be used to create multiple copies instantly. The application that uses the original data can see a slight performance hit due to the unsharing of data that can take place during updates.

An example to perform FileSnap

An example to perform FileSnap is as follows:

```
# vxfilesnap tfile1 stfile1
# ls -ltr
total 1108
drwxr-xr-x   2 root     root           96  Jul   6 00:41 lost+found
-rw-r--r--   1 root     root       282686  Jul   6 00:43 tfile1
```

```
-rw-r--r--   1 root     root      282686  Jul  6 00:44 stfile1
# ls -ltri
total 1108
3 drwxr-xr-x   2 root     root          96 Jul  6 00:41 lost+found
4 -rw-r--r--   1 root     root      282686 Jul  6 00:43 tfile1
5 -rw-r--r--   1 root     root      282686 Jul  6 00:44 stfile1
```

Comparison of the logical size output of the fsadm -S shared, du, and df commands

The `fsadm -S shared`, `du`, and `df` commands report different values for the size of a FileSnap. The `fsadm -S shared` command displays this size as the "logical size," which is the logical space consumed, in kilobytes, and accounts for both exclusive blocks and shared blocks. This value represents the actual disk space needed if the file system did not have any shared blocks. The value from the `fsadm -S shared` command differs from the output of `du -sk` command since the `du` command does not track the blocks consumed by VxFS structural files. As a result, the output of the `du -sk` command is less than the logical size output reported by the `fsadm -S shared` command.

The following examples show output from the `fsadm -S shared`, `du`, and `df` commands:

```
# mkfs -t vxfs /dev/vx/rdsk/dg/vol3
version 16 layout
134217728 sectors, 67108864 blocks of size 1024, log size 65536 blocks
rcq size   4096 blocks
largefiles supported
maxlink    supported

# mount -t vxfs /dev/vx/dsk/dg/vol3 /mnt

# df -k /mnt
Filesystem           1K-blocks    Used   Available Use% Mounted on
/dev/vx/dsk/dg1/vol3 52428800     83590  49073642  1%   /mnt

# /opt/VRTS/bin/fsadm -S shared /mnt
Mountpoint     Size(KB)   Available(KB)    Used(KB)    Logical_Size(KB) Space_Saved(KB)
/mnt           52428800   49073642         83590       83590            0

# du -sk /mnt
0       /mnt
```

```
# dd if=/dev/zero of=/mnt/foo bs=1024 count=10
10+0 records in
10+0 records out
10240 bytes (10 kB) copied, 0.018901 seconds, 542 kB/s

# vxfilesnap /mnt/foo /mnt/foo.snap

# df -k /mnt
Filesystem           1K-blocks      Used   Available Use% Mounted on
/dev/vx/dsk/dg1/vol3 52428800       83600  49073632  1%   /mnt

# /opt/VRTS/bin/fsadm -S shared /mnt
Mountpoint    Size(KB)   Available(KB)   Used(KB)   Logical_Size(KB) Space_Saved(KB)
/mnt          52428800   49073632        83600      83610            10

# du -sk /mnt
20      /mnt
```

Administering snapshot file systems

This chapter includes the following topics:

Snapshot file system backups

After a snapshot file system is created, the snapshot maintains a consistent backup of data in the snapped file system.

Backup programs, such as `cpio`, that back up a standard file system tree can be used without modification on a snapshot file system because the snapshot presents the same data as the snapped file system. Backup programs, such as `vxdump`, that access the disk structures of a file system require some modifications to handle a snapshot file system.

VxFS utilities recognize snapshot file systems and modify their behavior so that they operate the same way on snapshots as they do on standard file systems. Other backup programs that typically read the raw disk image cannot work on snapshots without altering the backup procedure.

These other backup programs can use the `fscat` command to obtain a raw image of the entire file system that is identical to an image obtainable by running a `dd` command on the disk device containing the snapped file system at the exact moment the snapshot was created. The snapread ioctl takes arguments similar to those of the `read` system call and returns the same results that are obtainable by performing a read on the disk device containing the snapped file system at the exact time the snapshot was created. In both cases, however, the snapshot file system provides a consistent image of the snapped file system with all activity complete—it is an instantaneous read of the entire file system. This is much different than the results that would be obtained by a `dd` or `read` command on the disk device of an active file system.

Snapshot file system performance

Snapshot file systems maximize the performance of the snapshot at the expense of writes to the snapped file system. Reads from a snapshot file system typically perform at nearly the throughput rates of reads from a standard VxFS file system.

The performance of reads from the snapped file system are generally not affected. However, writes to the snapped file system, typically average two to three times as long as without a snapshot. This is because the initial write to a data block requires reading the old data, writing the data to the snapshot, and then writing the new data to the snapped file system. If there are multiple snapshots of the same snapped file system, writes are even slower. Only the initial write to a block experiences this delay, so operations such as writes to the intent log or inode updates proceed at normal speed after the initial write.

Reads from the snapshot file system are impacted if the snapped file system is busy because the snapshot reads are slowed by the disk I/O associated with the snapped file system.

The overall impact of the snapshot is dependent on the read to write ratio of an application and the mixing of the I/O operations. For example, a database application running an online transaction processing (OLTP) workload on a snapped file system was measured at about 15 to 20 percent slower than a file system that was not snapped.

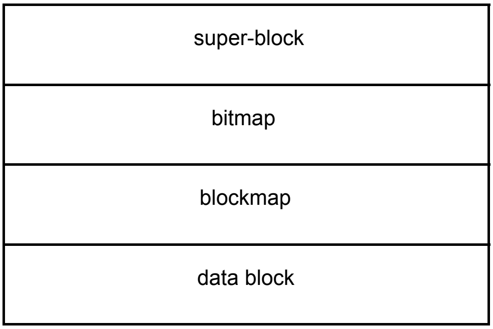
About snapshot file system disk structure

A snapshot file system consists of:

- A super-block
- A bitmap
- A blockmap
- Data blocks copied from the snapped file system

The following figure shows the disk structure of a snapshot file system.

Figure 20-1 The Snapshot Disk Structure



The super-block is similar to the super-block of a standard VxFS file system, but the magic number is different and many of the fields are not applicable.

The bitmap contains one bit for every block on the snapped file system. Initially, all bitmap entries are zero. A set bit indicates that the appropriate block was copied from the snapped file system to the snapshot. In this case, the appropriate position in the blockmap references the copied block.

The blockmap contains one entry for each block on the snapped file system. Initially, all entries are zero. When a block is copied from the snapped file system to the snapshot, the appropriate entry in the blockmap is changed to contain the block number on the snapshot file system that holds the data from the snapped file system.

The data blocks are filled by data copied from the snapped file system, starting from the beginning of the data block area.

Differences between snapshots and Storage Checkpoints

While snapshots and Storage Checkpoints both create a point-in-time image of a file system and only the changed data blocks are updated, there are significant differences between the two technologies:

Table 20-1 Differences between snapshots and Storage Checkpoints

Snapshots	Storage Checkpoints
Require a separate device for storage	Reside on the same device as the original file system

Table 20-1 Differences between snapshots and Storage Checkpoints *(continued)*

Snapshots	Storage Checkpoints
Are read-only	Can be read-only or read-write
Are transient	Are persistent
Cease to exist after being unmounted	Can exist and be mounted on their own
Track changed blocks on the file system level	Track changed blocks on each file in the file system

Storage Checkpoints also serve as the enabling technology for two other Veritas features: Block-Level Incremental Backups and Storage Rollback, which are used extensively for backing up databases.

See "About Storage Checkpoints" on page 435.

Creating a snapshot file system

You create a snapshot file system by using the `-o snapof=` option of the `mount` command. The `-o snapsize=` option may also be required if the device you are mounting does not identify the device size in its disk label, or if you want a size smaller than the entire device.

You must make the snapshot file system large enough to hold any blocks on the snapped file system that may be written to while the snapshot file system exists. If a snapshot runs out of blocks to hold copied data, the snapshot is disabled and further attempts to access the snapshot file system fail.

During periods of low activity (such as nights and weekends), a snapshot typically requires about two to six percent of the blocks of the snapped file system. During a period of high activity, the snapshot of a typical file system may require 15 percent of the blocks of the snapped file system. Most file systems do not turn over 15 percent of data in a single day. These approximate percentages tend to be lower for larger file systems and higher for smaller file systems. You can allocate blocks to a snapshot based on characteristics such as file system usage and duration of backups.

Warning: Any existing data on the device used for the snapshot is overwritten.

To create a snapshot file system

- Mount the file system with the `-o snapof=` option:

  ```
  # mount -t vxfs -o ro,snapof=/ \
  snapped_mount_point_mnt, snapsize=snapshot_size \
  /dev/vx/dsk/diskgrp/volume snapshot_mount_point
  ```

In the following examples, the `vxdump` utility is used to ascertain whether `/dev/rdsk/fsvol/vol1` is a snapshot mounted as `/backup/home` and does the appropriate work to get the snapshot data through the mount point.

These are typical examples of making a backup of a 300,000 block file system named `/home` using a snapshot file system on a Volume Manager volume with a snapshot mount point of `/backup/home`.

Example of creating a backup using a snapshop file system

1. To back up files changed within the last week using `cpio`:

   ```
   # mount -t vxfs -o snapof=/home,snapsize=100000 \
   /dev/vx/dsk/fsvol/vol1 /backup/home
   # cd /backup
   # find home -ctime -7 -depth -print | cpio -oc > /dev/st1
   # umount /backup/home
   ```

2. To do a level 3 backup of `/dev/vx/rdsk/fsvol/vol1` and collect those files that have changed in the current directory:

   ```
   # vxdump 3f - /dev/vx/rdsk/fsvol/vol1 | vxrestore -xf -
   ```

3. To do a full backup of `/home`, which exists on disk `/dev/vx/rdsk/fsvol/vol1`, and use `dd` to control blocking of output onto tape device using `vxdump`:

   ```
   # mount -t vxfs -o snapof=/home,snapsize=100000 \
   /dev/vx/dsk/fsvol/vol1 /backup/home
   # vxdump f - /dev/vx/rdsk/fsvol/vol1 | dd bs=128k > /dev/st1
   ```

Section 7

Optimizing storage with Storage Foundation

Chapter 21

Understanding storage optimization solutions in Storage Foundation

This chapter includes the following topics:

About thin provisioning

Thin provisioning is a storage array feature that optimizes storage use by allocating and reclaiming the storage on demand. With thin provisioning, the array allocates storage to applications only when the storage is needed, from a pool of free storage. Thin provisioning solves the problem of under-utilization of available array capacity. Administrators do not have to estimate how much storage an application requires. Instead, thin provisioning lets administrators provision large thin or thin reclaim capable LUNs to a host. When the application writes data, the physical storage is allocated from the free pool on the array to the thin-provisioned LUNs.

The two types of thin provisioned LUNs are thin-capable or thin-reclaim capable. Both types of LUNs provide the capability to allocate storage as needed from the free pool. For example, storage is allocated when a file system creates or changes a file. However, this storage is not released to the free pool when files get deleted. Therefore, thin-provisioned LUNs can become 'thick' over time, as the file system starts to include unused free space where the data was deleted. Thin-reclaim capable LUNs address this problem with the ability to release the once-used storage to the pool of free storage. This operation is called thin storage reclamation.

The thin-reclaim capable LUNs do not perform the reclamation automatically. The server using the LUNs must initiate the reclamation. The administrator can initiate a reclamation manually, or with a scheduled reclamation operation.

Storage Foundation provides several features to support thin provisioning and thin reclamation, and to optimize storage use on thin provisioned arrays.

See “About SmartMove” on page 470.

About thin optimization solutions in Storage Foundation

Array-based options like Thin Storage and Thin Provisioning help storage administrators to meet the challenges in managing their storage. These challenges include provisioning the storage, migrating data to maximize storage utilization, and maintaining the optimum storage utilization. Several features of Storage Foundation work together with the array functionality to solve these challenges.

Table 21-1 lists the Storage Foundation features and benefits relating to thin storage.

Table 21-1 Thin storage solutions in Storage Foundation

Feature	Description	Benefits
SmartMove	The SmartMove feature moves or copies only blocks in use by the Veritas File System	Maximizes use of thin storage. See “About SmartMove” on page 470. Improves performance for copy operations. Enables migration from thick LUNs to thin provisioned LUNs. See “Migrating to thin provisioning” on page 475.

Table 21-1 Thin storage solutions in Storage Foundation *(continued)*

Feature	Description	Benefits
Thin disk discovery	Storage Foundation provides discovery for thin storage devices.	Recognizes and displays thin attributes for thin disks.
Thin Reclamation	Thin reclamation commands enable you to reclaim space on a file system, disk, disk group, or enclosure level.	Improves storage utilization and savings. See "About the Thin Reclamation feature" on page 471.

About SmartMove

Storage Foundation provides the SmartMove utility to optimize move and copy operations. The SmartMove utility leverages the knowledge that Veritas File System (VxFS) has of the Veritas Volume Manager (VxVM) storage. VxFS lets VxVM know which blocks have data. When VxVM performs an operation that copies or moves data, SmartMove enables the operation to only copy or move the blocks used by the file system. This capability improves performance for synchronization, mirroring, and copying operations because it reduces the number of blocks that are copied. SmartMove only works with VxFS file systems that are mounted on VxVM volumes. If a file system is not mounted, the utility has no visibility into the usage on the file system.

SmartMove is not used for volumes that have instant snapshots.

The SmartMove operation also can be used to migrate data from thick storage to thin-provisioned storage. Because SmartMove copies only blocks that are in use by the file system, the migration process results in a thin-provisioned LUN.

SmartMove for thin provisioning

Storage Foundation uses the SmartMove feature for thin provisioning. SmartMove enables you to migrate storage from thick storage to thin storage. SmartMove provides the ability to maintain the intent of thin provisioning.

Without SmartMove, synchronization between disks copies the entire storage that is allocated to Veritas File System (VxFS) and Veritas Volume Manager (VxVM). Synchronizing or resynchronizing a volume, plex, or subdisk can lead to unused space being allocated on the thin disk. Over time, normal operations cause the storage to become thick. With SmartMove, the disk synchronization copies only blocks that are actually in use at the file system level. This behavior prevents unused space from being allocated when a disk is synchronized or resynchronized. The disks stay thin.

The SmartMove feature is enabled for all disks by default. To take advantage of thin provisioning, SmartMove must be enabled at least for thin disks.

About the Thin Reclamation feature

Storage Foundation supports reclamation of the unused storage on thin-reclamation capable arrays. Storage Foundation automatically discovers LUNs that support thin reclamation.

A Veritas File System (VxFS) file system can be mounted on a Veritas Volume Manager (VxVM) volume that is backed by a thin-capable array. The size of the VxVM volume is a virtual size, that is backed by the free storage pool. When files are created or changed, storage is physically allocated to the file system from the array. If the files on the file system are deleted or shrunk in size, the space is freed from the file system usage. However, the space is not removed from the physical allocation. Over time, the physical space allocated to the file system is greater than the actual space used by the file system. The thin LUN eventually becomes 'thick', as the physical space allocated nears the size of the LUN.

The Thin Reclamation feature provides the ability to release this unused space back to the free pool. Storage Foundation uses the VxFS allocation tables to identify unused blocks. VxVM maps this information about unused blocks down to the disk, enabling VxVM to return those blocks to the free pool. If the VxFS file system is not mounted, VxVM has no visibility into the file system usage. Therefore, it is critical that the file system is mounted when you perform a reclamation. The operation of reclamation can be done on a disk group, LUN, enclosure, or file system.

VxVM reclaims space automatically when you delete a volume or remove a plex. The automatic reclamation is asynchronous, so that the space is not reclaimed at the array level immediately. The disk is marked as pending reclamation. You cannot remove a disk from VxVM until the reclamation completes. You can control the timing and frequency of the automatic reclamation.

About reclaiming space on Solid State Devices (SSDs) with the TRIM operation

File systems that create and remove files often reuse storage blocks by overwriting a storage block with new contents. A Solid State Drive (SSD) device cannot overwrite a block of storage without erasing it first. This behavior causes a performance cost for writes to the previously used blocks, when compared to writes to unused or erased blocks. To avoid this cost, the TRIM operation informs the SSD which blocks of data are no longer in use and can be erased. The SSDs erase the unused blocks before the blocks are required for reuse, which improves the performance of the future write I/Os to the SSD. The TRIM operation also reduces wear leveling and fragmentation, because unused blocks are erased. The unused data does not get moved during a garbage collection or a cleaning cycle.

SF provides the TRIM operation only for supported devices. For more information, see the hardware compatibility list (HCL).

The SF components, Veritas File System (VxFS) and Veritas Volume Manager (VxVM), use the TRIM operation to free up the blocks that do not contain valid data. The TRIM capability is similar to thin reclamation, and is performed with the same commands. The default SF reclamation commands perform TRIM for SSDs and thin reclamation for Thin Reclaimable LUNs. For file systems and volumes that use both SSDs and Thin Reclaimable LUNs, you can choose whether SF performs only a TRIM operation, only a thin reclamation, or both.

See “Reclaiming space on a disk, disk group, or enclosure” on page 488.

See “Reclaiming space on a file system” on page 486.

To display information about SSDs, use the `vxdisk -o ssd list` command. SF can also discover and display the disk space usage for Veritas File System (VxFS) file systems on SSDs. The VxFS file systems must be mounted on Veritas Volume Manager (VxVM) volumes. Use the `vxdisk -o ssd -o fssize list` command.

See the `vxdisk`(1M) manual page.

Determining when to reclaim space on a thin reclamation LUN

When a thin LUN is used as a Veritas Volume Manager disk, the space is allocated only on an application write. Storage space is allocated from the free pool when files are created and written to in the file system. However, this storage is not automatically released to the free pool when data is deleted from a file system. As a result, all thin LUNs have a tendency to become thicker over time, with increased amounts of wasted storage (storage that is allocated but does not support application data).

As a storage administrator, you need to determine when to trigger the thin reclamation. The thin reclamation process can be time consuming, depending on various factors such as the size and fragmentation of the file system. The decision is a balance between how much space can be reclaimed, and how much time the reclaim operation will take.

The following considerations may apply:

- For a VxFS file system mounted on a VxVM volume, compare the file system usage to the actual physical allocation size to determine if a reclamation is desirable. If the file system usage is much smaller than the physical allocation size, it indicates that a lot of space can potentially be reclaimed. You may want to trigger a file system reclamation. If the file system usage is close to the physical allocation size, it indicates that the physical allocation is being used well. You may not want to trigger a reclamation.
  See "Displaying VxFS file system usage on thin reclamation LUNs" on page 484.
- The array may provide notification when the storage pool usage has reached a certain threshold. You can evaluate whether you can reclaim space with Storage Foundation to free more space in the storage pool.
- Deleted volumes are reclaimed automatically. You can customize the schedule for automatic reclamation.
  See "Configuring automatic reclamation" on page 492.

How automatic reclamation works

On thin-reclamable arrays, storage that is no longer in use needs to be reclaimed by the array. Storage Foundation automatically reclaims the space on the array for certain admimistrative operations, as follows:

- Deleting a volume.
- Removing a mirror.
- Shrinking a volume.
- Removing a log.
- Creating or growing a volume with the `init=zero` option.

The process of reclaiming storage on an array can be intense on the array. To avoid any effect on regular I/O's to the array, Storage Foundation performs the reclaim operation asynchronously. The disk is flagged as pending reclamation. The `vxrelocd` (or recovery) daemon asynchronously reclaims the disks marked for reclamation at a future time. By default, the `vxrelocd` daemon runs every day at 22:10 hours, and reclaims storage on the deleted volumes or plexes that are one day old.

To display the disks that are pending reclamation, use the following command:

```
# vxprint -z
```

You can configure the automatic reclamation to reclaim immediately, or to schedule the asynchronous reclamation.

See "Configuring automatic reclamation" on page 492.

You can also trigger a reclamation manually for a disk, disk group or enclosure. This operation also reclaims any disks flagged as pending reclamation.

See “Reclaiming space on a disk, disk group, or enclosure” on page 488.

Migrating data from thick storage to thin storage

This chapter includes the following topics:

About using SmartMove to migrate to Thin Storage

If you have existing data on a thick LUN, the SmartMove feature enables you to migrate the data to a thin LUN. The migration process copies only the blocks in use by the Veritas File System (VxFS) to the thin LUN. The SmartMove feature leverages the Veritas File System (VxFS) information about which blocks in a Veritas Volume Manager (VxVM) volume contain data. Therefore, the migration functionality is available only when a VxVM volume is on a mounted VxFS file system.

To migrate the data to the thin LUN, follow the recommended procedure.

See “Migrating to thin provisioning” on page 475.

Migrating to thin provisioning

The SmartMove™ feature enables migration from traditional LUNs to thinly provisioned LUNs, removing unused space in the process.

To migrate to thin provisioning

1. Check if the SmartMove feature is enabled.

   ```
   # vxdefault list
   KEYWORD                              CURRENT-VALUE    DEFAULT-VALUE
   usefssmartmove                       all              all
   ...
   ```

   If the output shows that the current value is none, configure SmartMove for all disks or thin disks.

   See "Configuring SmartMove " on page 669.

2. Add the new, thin LUNs to the existing disk group. Enter the following commands:

   ```
   # vxdisksetup -i da_name
   # vxdg -g datadg adddisk da_name
   ```

   where `da_name` is the disk access name in VxVM.

3. To identify LUNs with the thinonly or thinrclm attributes, enter:

   ```
   # vxdisk -o thin list
   ```

4. Add the new, thin LUNs as a new plex to the volume. On a thin LUN, when you create a mirrored volume or add a mirror to an existing LUN, VxVM creates a Data Change Object (DCO) by default. The DCO helps prevent the thin LUN from becoming thick, by eliminating the need for full resynchronization of the mirror.

   NOTE: The VxFS file system must be mounted to get the benefits of the SmartMove feature.

   The following methods are available to add the LUNs:

   - Use the default settings for the `vxassist` command:

     ```
     # vxassist -g datadg mirror datavol da_name
     ```

   - Specify the `vxassist` command options for faster completion. The -b option copies blocks in the background. The following command improves I/O throughput:

     ```
     # vxassist -b -oiosize=1m -t thinmig -g datadg mirror \
       datavol da_name
     ```

     To view the status of the command, use the `vxtask` command:

```
# vxtask list
TASKID PTID TYPE/STATE    PCT   PROGRESS
211         ATCOPY/R 10.64% 0/20971520/2232320 PLXATT vol1 vol1-02 xivdg smartmove
212         ATCOPY/R 09.88% 0/20971520/2072576 PLXATT vol1 vol1-03 xivdg smartmove
219         ATCOPY/R 00.27% 0/20971520/57344 PLXATT vol1 vol1-04 xivdg smartmove
```

```
# vxtask monitor 211
TASKID PTID TYPE/STATE    PCT   PROGRESS
211         ATCOPY/R 50.00% 0/20971520/10485760 PLXATT vol1 vol1-02 xivdg smartmove
211         ATCOPY/R 50.02% 0/20971520/10489856 PLXATT vol1 vol1-02 xivdg smartmove
211         ATCOPY/R 50.04% 0/20971520/10493952 PLXATT vol1 vol1-02 xivdg smartmove
211         ATCOPY/R 50.06% 0/20971520/10498048 PLXATT vol1 vol1-02 xivdg smartmove
211         ATCOPY/R 50.08% 0/20971520/10502144 PLXATT vol1 vol1-02 xivdg smartmove
211         ATCOPY/R 50.10% 0/20971520/10506240 PLXATT vol1 vol1-02 xivdg smartmove
```

- Specify the `vxassist` command options to reduce the effect on system performance. The following command takes longer to complete:

  ```
  # vxassist -oslow -g datadg mirror datavol da_name
  ```

5. Optionally, test the performance of the new LUNs before removing the old LUNs.

   To test the performance, use the following steps:

   - Determine which plex corresponds to the thin LUNs:

     ```
     # vxprint -g datadg
     ```

```
TY NAME               ASSOC           KSTATE   LENGTH     PLOFFS  STATE     TUTIL0 PUTIL0
dg datadg             datadg          -        -          -       -         -       -
dm THINARRAY0_02      THINARRAY0_02   -        83886080   -       -         -       -
dm STDARRAY1_01       STDARRAY1_01    -        41943040   -       -OHOTUSE  -       -

v  datavol            fsgen           ENABLED  41943040   -       ACTIVE    -       -
pl datavol-01         datavol         ENABLED  41943040   -       ACTIVE    -       -
sd STDARRAY1_01-01    datavol-01      ENABLED  41943040   0       -         -       -
pl datavol-02         datavol         ENABLED  41943040   -       ACTIVE    -       -
sd THINARRAY0_02-01   datavol-02      ENABLED  41943040   0       -         -       -
```

   The example output indicates that the thin LUN corresponds to plex datavol-02.

   - Direct all reads to come from those LUNs:

     ```
     # vxvol -g datadg rdpol prefer datavol datavol-02
     ```

6 Remove the original non-thin LUNs.

Note: The ! character is a special character in some shells. This example shows how to escape it in a bash shell.

```
# vxassist -g datadg remove mirror datavol \!STDARRAY1_01
# vxdg -g datadg rmdisk STDARRAY1_01
# vxdisk rm STDARRAY1_01
```

7 Grow the file system and volume to use all of the larger thin LUN:

```
# vxresize -g datadg -x datavol 40g da_name
```

Chapter 23

Maintaining Thin Storage with Thin Reclamation

This chapter includes the following topics:

Reclamation of storage on thin reclamation arrays

Storage Foundation supports reclamation of the unused storage on thin-reclamation capable arrays and LUNs. Storage Foundation can reclaim blocks in a Veritas File System (VxFS) file system that is mounted on a Veritas Volume Manager (VxVM) volume.

The thin reclamation feature is supported only for LUNs that have the `thinrclm` attribute. VxVM automatically discovers LUNs that support thin reclamation from thin capable storage arrays. You can list devices that are known to have the `thin` or `thinrclm` attributes on the host.

See "Identifying thin and thin reclamation LUNs " on page 481.

For a list of the storage arrays that support thin reclamation, refer to the hardware compatibility list (HCL).

Thin reclamation is not supported for boot devices.

You can use the thin reclamation feature in the following ways:

- Space is reclaimed automatically when a volume is deleted. Because it is asynchronous, you may not see the reclaimed space immediately.
- Perform the reclamation operation on a disk group, LUN, or enclosure using the `vxdisk` command.
  See "Reclaiming space on a disk, disk group, or enclosure" on page 488.
- Perform the reclamation operation on a Veritas File System (VxFS) file system using the `fsadm` command.
  See "Reclaiming space on a file system" on page 486.

About Thin Reclamation of a disk, a disk group, or an enclosure

Storage Foundation provides the ability to reclaim unused space on thin-provisioned arrays, without needing to stop application I/O. The Veritas File System (VxFS) file system must be mounted.

You can trigger thin reclamation on one or more disks, disk groups, or enclosures. The reclamation process scans the specified storage for the VxVM volumes that have a mounted VxFS file system. Each volume is analyzed for any previously allocated space that the VxFS file system no longer uses. The unused space is released to the free storage pool on the thin array. The reclamation skips any volumes that do not have a mounted VxFS file system. The reclamation process also releases the space for any volumes or plexes that are marked as pending reclamation.

By default, the reclamation command also performs the TRIM operation if the specified storage is on Solid State Devices (SSDs).

See "About reclaiming space on Solid State Devices (SSDs) with the TRIM operation" on page 471.

A full reclamation process also scans the specified storage for free space that is outside of the VxVM volumes.

Thin reclamation does not reclaim space on a volume that is part of an instant snapshot hierarchy.

Thin Reclamation takes a considerable amount of time when you reclaim thin storage on a large number of LUNs or an enclosure or disk group. As with other long-running operations, VxVM creates a task for a reclaim operation. You can monitor the reclaim operation with the `vxtask` command.

See “Monitoring Thin Reclamation using the vxtask command” on page 491.

About Thin Reclamation of a file system

Veritas File System (VxFS) supports reclamation of free storage on a Thin Storage LUN. Free storage is reclaimed using the `fsadm` command. You can perform the default reclamation or aggressive reclamation. If you used a file system for a long time and must perform reclamation on the file system, Veritas recommends that you run aggressive reclamation. Aggressive reclamation compacts the allocated blocks, which creates larger free blocks that can potentially be reclaimed.

See the `fsadm_vxfs`(1M) manual page.

Thin Reclamation is only supported on file systems mounted on a VxVM volume.

Thin Reclamation is not supported for file systems mounted on RAID5 volumes.

Veritas File System also supports reclamation of a portion of the file system using the `vxfs_ts_reclaim`() API.

See the `vxfs_ts_reclaim`(3) manual page and the *Veritas File System Programmer's Reference Guide*.

Note: Thin Reclamation is a slow process and may take several hours to complete, depending on the file system size. Thin Reclamation is not guaranteed to reclaim 100% of the free space.

You can track the progress of the Thin Reclamation process by using the `vxtask list` command when using the Veritas Volume Manager (VxVM) command `vxdisk reclaim`.

See the `vxtask`(1M) and `vxdisk`(1M) manual pages.

You can administer Thin Reclamation using VxVM commands.

Identifying thin and thin reclamation LUNs

Using Dynamic Multi-Pathing (DMP), Storage Foundation automatically discovers thin devices that have been recognized on the host as `thin` or `thinrclm`. DMP uses the Veritas array support libraries (ASLs) to recognize vendor-specific thin attributes and claim devices accordingly as `thin` or `thinrclm`.

Thin devices that are classified as `thin` are capable of thin provisioning. Veritas Thin Reclamation only works on devices with the `thinrclm` attribute set. Before performing thin reclamation, determine whether the system recognizes the LUN as a `thinrclm` LUN.

To identify devices on a host that are known to have the `thin` or `thinrclm` attributes, use the `vxdisk -o thin list` command. The `vxdisk -o thin list` command also reports on the size of the disk, and the physical space that is allocated on the array.

To identify thin and thinrclm LUNs

- To identify all of the `thin` or `thinrclm` LUNs that are locally known to a host, use the following command:

  ```
  # vxdisk -o thin list

  DEVICE          SIZE(MB) PHYS_ALLOC(MB) GROUP  TYPE     RECLAIM_CMD
  xiv0_6695       16384    30             dg1    thinrclm WRITE_SAME
  xiv0_6696       16384    30             dg1    thinrclm WRITE_SAME
  xiv0_6697       16384    30             dg1    thinrclm WRITE_SAME
  xiv0_6698       16384    30             dg1    thinrclm WRITE_SAME
  xiv0_6699       16384    30             dg1    thinrclm WRITE_SAME
  3pardata0_5074  2048     2043           vvrdg  thinrclm WRITE_SAME
  3pardata0_5075  2048     2043           vvrdg  thinrclm WRITE_SAME
  3pardata0_5076  2048     1166           vvrdg  thinrclm WRITE_SAME
  3pardata0_5077  2048     2043           vvrdg  thinrclm WRITE_SAME
  3pardata0_5081  2048     1092           vvrdg  thinrclm WRITE_SAME
  ```

  In the output, the SIZE column shows the size of the disk. The PHYS_ALLOC column shows the physical allocation on the array side. The TYPE indicates whether the array is thin or thinrclm. The RECLAIM_CMD column displays which reclamation method that DMP uses.

  See the `vxdisk`(1m) manual page.

Displaying detailed information about reclamation commands

Dynamic Multi-Pathing (DMP) supports several array-level reclamation commands: UNMAP, WRITE_SAME, TRIM, and PTRIM. The Array Support Library (ASL) for each array uses the most suitable reclamation method from those that are supported for the array. DMP uses the preferred reclamation method that the vendor suggests or selects a reclamation method based on performance analysis.

You cannot change the reclamation method that DMP uses. However, you can view the information about the reclamation command that DMP has selected.

You can view information about other reclamation attributes that DMP uses to create reclamation requests. The reclamation attributes are vendor-specific.

To display information about the thin reclamation methods

- To display detailed information about the thin reclamation methods for a device, use the following command:

```
# vxdisk -p list xiv0_6699

DISK                  : xiv0_6699
VID                   : IBM
UDID                  : IBM%5F2810XIV%5F0E95%5F1A2B
TP_PREF_RCLMCMD       : write_same
TP_RECLM_CMDS         : write_same, unmap
TP_ALLOC_UNIT         : 1048576
TP_MAX_REC_SIZE       : 268435456
TP_LUN_SHIFT_OF       : 0
SCSI_VERSION          : 5
SCSI3_VPD_ID          : 001738000E951A2B
REVISION              : 10.2
.
.
.
LUN_SIZE              : 33554432
NUM_PATHS             : 4
STATE                 : online
```

The following fields show the information about the reclamation attributes:

TP_PREF_RCLMCMD	The preferred reclaim method for this array.
TP_RECLM_CMDS	The reclamation methods that the underlying device supports.
TP_ALLOC_UNIT	The size in bytes of the allocation unit for thin provisioning on the device.
TP_MAX_REC_SIZE	The maximum size in bytes of the reclamation I/O on the device.
TP_LUN_SHIFT_OF	The value in bytes by which the vendor shifts the initial offset for the LUN. Set this value to align the reclaim request to the TP_ALLOC_UNIT.

Displaying VxFS file system usage on thin reclamation LUNs

Storage Foundation can discover and display the disk space usage for Veritas File System (VxFS) file systems on `thin` or `thinrclm` devices. The VxFS file systems must be mounted on Veritas Volume Manager (VxVM) volumes. The usage information can help you decide when to perform thin reclamation of a file system.

See "Determining when to reclaim space on a thin reclamation LUN" on page 472.

To report the per-LUN disk space usage for currently mounted VxFS file systems on VxVM volumes, use the `vxdisk -o thin -o fssize list` command. The command displays the amount of disk space that currently contains files and is actively in use by the VxFS file system. The usage does not include any space that is allocated to the file system but was freed by deleting files. If more than one mounted VxFS file system uses the device, the file system usage column displays the consolidated space usage. The `-o fssize` option maps the file system space usage to the underlying LUNs. The disk space usage statistics may differ slightly from the usage reported by other utilities because of the base unit used for the calculation.

The following limitations apply to the command to display file system usage:

- The `-o fssize` option does not display the space used by cache objects or instant snapshots.
- RAID5 format is not supported.
- If the VxFS file system is not mounted, or if the device has both mounted and unmounted VxFS file systems, no information is displayed. The file system usage (FS_SIZE) column displays a dash (-).

You can display the size and usage for all `thin` or `thinrclm` LUNs, or specify an enclosure name or a device name. If you specify one or more devices or enclosures, the command displays only the space usage on the specified devices. If the specified device is not a `thin` device or `thinrclm` device, the device is listed but the FS_SIZE column displays a dash (-).

If a VxFS file system spans multiple devices, you must specify all of the devices to display the entire file system usage. If you specify only some of the devices, the file system usage is incomplete. The command ignores the file system usage on any devices that are not specified.

Note: The command can potentially take a long time to complete depending on the file system size, the level of fragmentation, and other factors. The command creates a task that you can monitor with the `vxtask` command.

The command output displays the following information.

DEVICE

The name of the VxVM disk, in either Enclosure-based naming (EBN) or OS-based naming (OSN).

SIZE

The size of the disk; that is, the size that is presented to the file system. This size represents the virtual size rather than the actual physical space used on the device.

PHYS_ALLOC

The physical allocation on the array side. This size represents the physical space that is allocated as the application writes to the file system. When the files are deleted or changed, the physical space remains allocated until a reclamation is performed. In this case, the physical size includes some unused space.

FS_SIZE

The physical space Veritas File System (VxFS) file systems are using. The VxFS file systems must be mounted on VxVM volumes. The information is displayed only for thin provisioning capable (thin) or thin reclamation capable (thinrclm) LUNs.

GROUP

The disk group that contains the disk.

TYPE

The type of thin devices – thin provisioning capable (thin) or thin reclamation capable (thinrclm). The `vxdisk -o thin list` command displays thick disks only if you explicitly specify the disk name on the command line.

RECLAIM_CMD

The reclamation method that DMP uses.

To display file system usage on all thin LUNs

- To display the file system usage on all `thin` or `thinrclm` LUNs known locally to the system, use the following command:

  ```
  $ vxdisk -o thin,fssize [-u unit] list
  ```

  Where *unit* is a size unit for the display. For example:

  ```
  $ vxdisk -o thin,fssize -u m list
  ```

```
DEVICE    SIZE      PHYS_ALLOC FS_SIZE GROUP  TYPE     RECLAIM_CMD
emc0_428a 16384.00m 6335.00m   610.00m  mydg  thinrclm  WRITE_SAME
emc0_428b 16384.00m 3200.00m    22.00m  mydg  thinrclm  WRITE_SAME
emc0_4287 16384.00m 6233.00m   617.00m  mydg  thinrclm  WRITE_SAME
emc0_4288 16384.00m 1584.00m  1417.00m  mydg  thinrclm  WRITE_SAME
emc0_4289 16384.00m 2844.00m  1187.00m  mydg  thinrclm  WRITE_SAME
xiv0_030f 16384.00m 2839.00m  1223.00m  xivdg thinrclm  WRITE_SAME
xiv0_0307 16384.00m  666.00m   146.00m  xivdg thinrclm  WRITE_SAME
xiv0_0308 16384.00m  667.00m   147.00m  xivdg thinrclm  WRITE_SAME
xiv0_0309 16384.00m    3.00m  -         -     thinrclm  WRITE_SAME
xiv0_0310 16384.00m   30.00m  -         -     thinrclm  WRITE_SAME
```

Or, to display the file system usage on a specific LUN or enclosure, use the following form of the command:

```
$ vxdisk -o thin,fssize list [-u unit] disk|enclosure
```

For example:

```
$ vxdisk -o thin,fssize list emc0
```

```
DEVICE    SIZE(MB) PHYS_ALLOC(MB) FS_SIZE(MB) GROUP TYPE     RECLAIM_CMD
emc0_428a 16384    6335           610         mydg  thinrclm WRITE_SAME
emc0_428b 16384    6335           624         mydg  thinrclm WRITE_SAME
emc0_4287 16384    6335           617         mydg  thinrclm WRITE_SAME
emc0_4288 16384    1584           617         mydg  thinrclm WRITE_SAME
emc0_4289 16384    2844           1187        mydg  thinrclm WRITE_SAME
```

Reclaiming space on a file system

Table 23-1 lists the `fsadm` command options that administer thin reclamation.

Table 23-1 fsadm options for administering thin reclamation

Option	Description
`-o aggressive|-A`	Initiates Thin Storage aggressive reclamation. Aggressive reclamation is not supported on SSD devices.
`-o analyse|analyze`	Initiates the analyze reclaim option.
`-o auto`	Initiates the auto reclaim option.
`-o ssd`	Initiates the TRIM command on an underlying SSD trim-capable device.
`-o thin`	Initiates thin reclamation on the underlying Thin Reclaim-capable device.
`-P`	Performs multi-threaded Thin Storage reclamation. By default, the `fsadm` command performs single-threaded Thin Storage reclamation. To use multi-threaded Thin Storage Reclamation, the array must support multiple concurrent reclaim operations.
`-R`	Performs reclamation of free storage to the Thin Storage LUN on a VxFS file system .

See the `fsadm_vxfs`(1M) manual page.

To perform aggressive space reclamation

1 Ensure you mounted the VxFS file system.

See the `mount`(1M) manual page.

If you must mount the VxFS file system, see the `mount_vxfs`(1M) manual page.

2 Perform aggressive reclamation of free storage to the Thin Storage LUN on the VxFS file system that is mounted at `/mnt1`:

```
# /opt/VRTS/bin/fsadm -R -o aggressive /mnt1
```

To perform space reclamation

1. Ensure you mounted the VxFS file system.

   See the mount(1M) manual page.

   If you must mount the VxFS file system, see the mount_vxfs(1M) manual page.

2. Perform space reclamation on the VxFS file system that is mounted at /mnt1:

```
# /opt/VRTS/bin/fsadm -R /mnt1
```

Reclaiming space on a disk, disk group, or enclosure

Use the vxdisk reclaim command to trigger online Thin Reclamation on one or more disks, disk groups, or enclosures. By default, the vxdisk reclaim command performs Thin Reclamation on the disks where the VxVM volume is on a “mounted” VxFS file system. The reclamation skips disks that do not have a VxFS file system mounted. Thin reclamation is not supported for RAID-5 volumes, or for instant snapshots.

Storage Foundation logs the statistics for reclamation events in the /etc/vx/log/reclaim_log file.

See “About the reclamation log file” on page 490.

By default, the commands below also perform TRIM reclamation if the specified disks are supported Solid State Devices (SSDs).

Reclaiming space on a disk

- Use the following command to trigger reclamation:

```
# vxdisk reclaim [disk...]
```

For example, to trigger reclamation on LUNs hitachi_usp0_065a and hitachi_usp0_065b:

```
# vxdisk reclaim hitachi_usp0_065a hitachi_usp0_065b
```

In the above example, suppose the hitachi_usp0_065a contains a VxVM volume vol1 with a VxFS file system. If the VxFS file system is not mounted, the command skips reclamation for hitachi_usp0_065a. The command scans hitachi_usp0_065b, and reclaims any unused space.

Performing an aggressive space reclamation on a disk

- Use the following command to trigger reclamation:

  ```
  # vxdisk -o full reclaim [disk...]
  ```

  For example, to trigger reclamation on LUNs hitachi_usp0_065a:

  ```
  # vxdisk -o full reclaim hitachi_usp0_065a
  ```

  In the above example, suppose the hitachi_usp0_065a contains a VxVM volume vol1 with a VxFS file system mounted. With the -o full option, the above command scans hitachi_usp0_065a for unused space outside of the vol1, and reclaims any unused space found. For example, if there is space between subdisks, it is reclaimed.

Reclaiming space on an SSD disk

- Use the following command to trigger TRIM operation:

  ```
  # vxdisk [-o ssd] reclaim [disk...]
  ```

  For example, to trigger TRIM on fiodrive0_0 and fiodrive0_1:

  ```
  # vxdisk reclaim fiodrive0_0 fiodrive0_1
  ```

Reclaiming space on a disk group

- Use the following command to trigger reclamation:

  ```
  # vxdisk [-o ssd | -o thin] reclaim diskgroup
  ```

  For example, to trigger reclamation on the disk group oradg:

  ```
  # vxdisk reclaim oradg
  ```

  If the disk group contains both SSDs and Thin Reclamation LUNs, you can use the `-o ssd` option to perform only the TRIM operation. Use the `-o thin` option to perform only the thin reclamation.

Reclaiming space on an enclosure

- Use the following command to trigger reclamation:

  ```
  # vxdisk reclaim enclosure
  ```

  For example, to trigger reclamation on the enclosure=EMC_CLARiiON0:

  ```
  # vxdisk reclaim EMC_CLARiiON0
  ```

You can turn off TRIM functionalty or thin reclamation for a specific device with the following command:

```
# vxdisk set reclaim=off disk
```

See the vxdisk(1M) manual page.

About the reclamation log file

Storage Foundation logs the statistics for reclamation events in the /etc/vx/log/reclaim_log file. Table 23-2 describes the fields in the reclamation log file.

For Veritas Volume Replicator (VVR), reclamation logging only happens for the local node.

Table 23-2 The reclamation log file fields

LOG fields	Description
START_TIME	The start time of the reclamation task.
DURATION	The time taken to complete the reclamation task.
DISKGROUP	The disk group name associated with the subdisk. For TYPE=GAP, the disk group value may be NULL value.
VOLUME	The volume associated with the subdisk. If a volume is not associated with the subdisk, the value is NULL.
DISK	The disk associated with the subdisk.
SUBDISK	The subdisk name for which the reclamation operation is performed.
OFFSET	The starting offset of the subdisk.
LEN	The total length of the subdisk.
PA_BEFORE	The physical allocation before the reclamation task.
PA_AFTER	The physical allocation after the reclamation task.
TYPE	The type for the reclamation operation. The value is one of the following: ■ GAP: reclaim the gap between the subdisks ■ SD: reclaim the subdisk ■ FULL: reclaim the full LUN on disk with no DG present ■ VXFS: reclaim a mounted VxFS file system.

Table 23-2 The reclamation log file fields *(continued)*

LOG fields	Description
STATUS	Whether the reclamation operation succeeded or not. In case of failure, the STATUS also displays the error code. When an object such as a volume or plex is removed, the status is logged as "Pending."

Monitoring Thin Reclamation using the vxtask command

The thin reclamation can be an intensive operation that may be time consuming, depending on the size of the disk and the amount of space to be reclaimed. As with other long-running tasks, you can monitor the operation with the `vxtask` command.

To monitor thin reclamation

1 Initiate the thin reclamation as usual, for a disk, disk group, or enclosure.

```
# vxdisk reclaim diskgroup| disk| enclosure
```

For example:

```
# vxdisk reclaim dg100
```

2 To monitor the reclamation status, run the following command in another session:

```
# vxtask monitor
```

```
TASKID PTID TYPE/STATE    PCT   PROGRESS
1258   -    RECLAIM/R 17.28% 65792/33447328/5834752 RECLAIM vol4 dg100
1259   -    RECLAIM/R 25.98% 0/20971520/5447680     RECLAIM vol2 dg100
1263   -    RECLAIM/R 25.21% 0/20971520/5287936     RECLAIM vol3 dg100
1258   -    RECLAIM/R 25.49% 0/20971520/3248128     RECLAIM vol4 dg100
1258   -    RECLAIM/R 27.51% 0/20971520/3252224     RECLAIM vol4 dg100
1263   -    RECLAIM/R 25.23% 0/20971520/5292032     RECLAIM vol3 dg100
1259   -    RECLAIM/R 26.00% 0/20971520/5451776     RECLAIM vol2 dg100
```

3 If you have multiple tasks, you can use the following command to display the tasks.

```
# vxtask list

TASKID PTID TYPE/STATE    PCT   PROGRESS
1258   -    RECLAIM/R 17.28% 65792/33447328/5834752 RECLAIM vol4 dg100
1259   -    RECLAIM/R 25.98% 0/20971520/5447680     RECLAIM vol2 dg100
1263   -    RECLAIM/R 25.21% 0/20971520/5287936     RECLAIM vol3 dg100
```

4 Use the task id from the above output to monitor the task:

```
# vxtask monitor 1258
```

```
TASKID  PTID TYPE/STATE    PCT   PROGRESS
1258     -   RECLAIM/R 17.28% 65792/33447328/5834752 RECLAIM vol4 dg100
1258     -   RECLAIM/R 32.99% 65792/33447328/11077632 RECLAIM vol4 dg100
1258     -   RECLAIM/R 45.55% 65792/33447328/15271936 RECLAIM vol4 dg100
1258     -   RECLAIM/R 50.00% 0/20971520/10485760     RECLAIM vol4 dg100
.
.
.
```

The `vxdisk reclaim` command runs in another session while you run the `vxtask list` command.

See the `vxtask`(1m) manual page.

Configuring automatic reclamation

The `vxrelocd` daemon tracks the disks that require reclamation. By default, the `vxrelocd` daemon runs everyday at 22:10 hours and reclaims storage on the deleted volumes that are one day old.

To control the schedule for reclamation, use the following tunable parameters:

`reclaim_on_delete_wait_period`

Specifies the number of days after a volume or plex is deleted when VxVM reclaims the storage space. The value is an integer between `-1` and `367`.

The default value is `1`, which means the space is reclaimed the next day.

A value of `-1` indicates that the storage is reclaimed immediately.

A value of `367` indicates that the storage space is not reclaimed automatically. Storage space can only be reclaimed manually using the `vxdisk reclaim` command.

`reclaim_on_delete_start_time`

The time of day when VxVM starts the reclamation for deleted volumes. The value is any time of day in 24 hour format. (hh:mm)

The default time is `22:10`.

Change the tunables using the `vxdefault` command. See the `vxdefault`(1m) manual page.

Veritas InfoScale 4k sector device support solution

This chapter includes the following topics:

About 4K sector size technology

Over the years, the data that is stored on the storage devices such as the hard disk drives (HDD) and Solid State Devices (SSD) has been formatted into a small logical block which is referred to as **Sector**. Despite of increase in storage densities over a period of time, the storage device sector size has remained consistent - 512 bytes. But, this device sector size proves to be inefficient for Solid State Devices (SSD).

Benefits of transition from 512 bytes to 4096 bytes or 4K sector

The 4K sector disks are the first advanced generation format devices. They help with the optimum use of the storage surface area by reducing the amount of space that is allocated for headers and error correction code for sectors. They are considered to be more efficient for larger files as compared to smaller files.

The advanced format devices with 4K sector size are considered to be beneficial over 512-bytes sector size for following reasons:

1. Improves the format efficiency
2. Provides a more robust error correction

Considering the benefits, many storage device manufacturers such as Hitachi, NEC, Fujitsu have started shipping 4K sector devices.

However, many aspects of modern computing still assume that the sectors are always 512-bytes. The alternative is to implement 4K sector transition that is combined with the 512-bytes sector emulation method. The disadvantage of 512-bytes sector emulation method is that it reduces the efficiency of the device.

Veritas InfoScale uses the Veritas Volume Manager and Veritas File System storage components to provide a solution that supports 4K sector devices (formatted with 4KB) in storage environment. Earlier, you were required to format 4K devices with 512-bytes. You can now directly use the 4K sector devices with Veritas InfoScale without any additional formatting.

Supported operating systems

You can use 4k sector devices with Veritas InfoScale 7.2 or later versions only on Linux (RHEL, SLES, and supported RHEL compatible distributions) and Solaris 11 operating systems.

See "Veritas InfoScale unsupported configurations" on page 495.

See "Migrating VxFS file system from 512-bytes sector size devices to 4K sector size devices" on page 496.

Veritas InfoScale unsupported configurations

This section lists the various Veritas InfoScale features that are not supported with 4K sector devices.

- **Volume Layout**: RAID-5 is not supported. All other volume layouts are supported
- **VxVM Disk Group support**: Only cross Platform Data Sharing (CDS) disk group format is supported. A disk group with a combination or a mix of 512-byte sector disks and 4K sector disks is not supported. Two different disk groups, one with 4K disks and other with 512-byte disks can co-exist
- **VxVM SmartIO configuration support**: If the sector size of the disk which hosts the application volume and the disk which hosts the cache differ, then caching is not enabled on that application volume.
- Storage area network (SAN) boot
- Root disk encapsulation
- Snapshot across disk groups with different sector size disks
- **VxFS File System support**: The file system **block size** and **logiosize** less than 4 KB is not supported on a 4K sector device

Migrating VxFS file system from 512-bytes sector size devices to 4K sector size devices

This section describes the procedure to migrate VxFS file system from 512 bytes to 4K sector size devices.

VxFS file systems on the existing 512-bytes sector devices might have been created with a file system block size of 1 KB or 2 KB, which is not supported on a 4K sector device. Hence, the traditional storage migration solutions, such as array level or volume level migration or replication may not work properly.

Starting With Veritas InfoScale 7.2 and later versions, you can migrate VxFS file system from 512-bytes sector size devices to 4K sector size devices using the standard file copy mechanism.

Note: The standard file copy mechanism may not preserve certain file level attributes and allocation geometry.

Note: Migration of VxFS file system from 512-bytes sector size to 4K sector size is supported only on Linux (RHEL, SLES, and supported RHEL compatible distributions) and Solaris 11 operating systems.

To migrate VxFS file system from 512-bytes sector size devices to 4K sector size devices:

1. Mount 512 bytes and 4K VxFS file system

```
# mount -t vxfs /dev/vx/dsk/diskgroup/volume_512B /mnt1
```

```
# mount -t vxfs /dev/vx/dsk/diskgroup/volume_4K /mnt2
```

2. Copy all the files from /mnt1 to /mnt2 manually

```
# cp -r /mnt1 /mnt2
```

3. Unmount both the VxFS file system - 512 bytes and 4K

```
# umount  /mnt1
```

```
# umount  /mnt2
```

See “About 4K sector size technology” on page 494.

See “Veritas InfoScale unsupported configurations” on page 495.

Section 8

Maximizing storage utilization

Chapter 25

Understanding storage tiering with SmartTier

This chapter includes the following topics:

About SmartTier

SmartTier matches data storage with data usage requirements. After data matching, the data can then be relocated based upon data usage and other requirements determined by the storage or database administrator (DBA).

As more and more data is retained over a period of time, eventually, some of that data is needed less frequently. The data that is needed less frequently still requires a large amount of disk space. SmartTier enables the database administrator to manage data so that less frequently used data can be moved to slower, less expensive disks. This also permits the frequently accessed data to be stored on faster disks for quicker retrieval.

Tiered storage is the assignment of different types of data to different storage types to improve performance and reduce costs. With SmartTier, storage classes are used to designate which disks make up a particular tier. There are two common ways of defining storage classes:

- Performance, or storage, cost class: The most-used class consists of fast, expensive disks. When data is no longer needed on a regular basis, the data can be moved to a different class that is made up of slower, less expensive disks.

- Resilience class: Each class consists of non-mirrored volumes, mirrored volumes, and n-way mirrored volumes.

  For example, a database is usually made up of data, an index, and logs. The data could be set up with a three-way mirror because data is critical. The index could be set up with a two-way mirror because the index is important, but can be recreated. The redo and archive logs are not required on a daily basis but are vital to database recovery and should also be mirrored.

SmartTier is a VxFS feature that enables you to allocate file storage space from different storage tiers according to rules you create. SmartTier provides a more flexible alternative compared to current approaches for tiered storage. Static storage tiering involves a manual one- time assignment of application files to a storage class, which is inflexible over a long term. Hierarchical Storage Management solutions typically require files to be migrated back into a file system name space before an application access request can be fulfilled, leading to latency and run-time overhead. In contrast, SmartTier allows organizations to:

- Optimize storage assets by dynamically moving a file to its optimal storage tier as the value of the file changes over time
- Automate the movement of data between storage tiers without changing the way users or applications access the files
- Migrate data automatically based on policies set up by administrators, eliminating operational requirements for tiered storage and downtime commonly associated with data movement

Note: SmartTier is the expanded and renamed feature previously known as Dynamic Storage Tiering (DST).

SmartTier policies control initial file location and the circumstances under which existing files are relocated. These policies cause the files to which they apply to be created and extended on specific subsets of a file systems's volume set, known as placement classes. The files are relocated to volumes in other placement classes when they meet specified naming, timing, access rate, and storage capacity-related conditions.

In addition to preset policies, you can manually move files to faster or slower storage with SmartTier, when necessary. You can also run reports that list active policies, display file activity, display volume usage, or show file statistics.

SmartTier leverages two key technologies included with Storage Foundation: support for multi-volume file systems and automatic policy-based placement of files within the storage managed by a file system. A multi-volume file system occupies two or more virtual storage volumes and thereby enables a single file system to span across multiple, possibly heterogeneous, physical storage devices. For example the first volume could reside on EMC Symmetrix DMX spindles, and the second volume could reside on EMC CLARiiON spindles. By presenting a single name space, multi-volumes are transparent to users and applications. This multi-volume file system remains aware of each volume's identity, making it possible to control the locations at which individual files are stored. When combined with the automatic policy-based placement of files, the multi-volume file system provides an ideal storage tiering facility, which moves data automatically without any downtime requirements for applications and users alike.

In a database environment, the access age rule can be applied to some files. However, some data files, for instance are updated every time they are accessed and hence access age rules cannot be used. SmartTier provides mechanisms to relocate portions of files as well as entire files to a secondary tier.

To use SmartTier, your storage must be managed using the following features:

- VxFS multi-volume file system
- VxVM volume set
- Volume tags
- SmartTier management at the file level
- SmartTier management at the sub-file level

About VxFS multi-volume file systems

Multi-volume file systems are file systems that occupy two or more virtual volumes. The collection of volumes is known as a volume set, and is made up of disks or disk array LUNs belonging to a single Veritas Volume Manager (VxVM) disk group. A multi-volume file system presents a single name space, making the existence of multiple volumes transparent to users and applications. Each volume retains a separate identity for administrative purposes, making it possible to control the locations to which individual files are directed.

See "About multi-volume file systems" on page 512.

This feature is available only on file systems meeting the following requirements:

- The minimum disk group version is 140.
  See "Disk group versions" on page 672.
- The minimum file system layout version is 7 for file level SmartTier.
- The minimum file system layout version is 8 for sub-file level SmartTier.

To convert your existing VxFS system to a VxFS multi-volume file system, you must convert a single volume to a volume set.

See "Converting a single volume file system to a multi-volume file system" on page 517.

The VxFS volume administration utility (fsvoladm utility) can be used to administer VxFS volumes. The fsvoladm utility performs administrative tasks, such as adding, removing, resizing, encapsulating volumes, and setting, clearing, or querying flags on volumes in a specified Veritas File System.

See the `fsvoladm` (1M) manual page for additional information about using this utility.

About VxVM volume sets

Volume sets allow several volumes to be represented by a single logical object. Volume sets cannot be empty. All I/O from and to the underlying volumes is directed via the I/O interfaces of the volume set. The volume set feature supports the multi-volume enhancement to Veritas File System (VxFS). This feature allows file systems to make best use of the different performance and availability characteristics of the underlying volumes. For example, file system metadata could be stored on volumes with higher redundancy, and user data on volumes with better performance.

About volume tags

You make a VxVM volume part of a placement class by associating a volume tag with it. For file placement purposes, VxFS treats all of the volumes in a placement class as equivalent, and balances space allocation across them. A volume may have more than one tag associated with it. If a volume has multiple tags, the volume belongs to multiple placement classes and is subject to allocation and relocation policies that relate to any of the placement classes.

Warning: Multiple tagging should be used carefully.

A placement class is a SmartTier attribute of a given volume in a volume set of a multi-volume file system. This attribute is a character string, and is known as a volume tag.

SmartTier file management

SmartTier enables administrators of multi-volume VxFS file systems to manage the placement of files on individual volumes in a volume set by defining placement policies that control both initial file location and the circumstances under which existing files are relocated. These placement policies cause the files to which they apply to be created and extended on specific subsets of a file system's volume set, known as placement classes. The files are relocated to volumes in other placement classes when they meet the specified naming, timing, access rate, and storage capacity-related conditions.

File-based movement:

- The administrator can create a file allocation policy based on filename extension before new files are created, which will create the datafiles on the appropriate tier during database creation.
- The administrator can also create a file relocation policy for database files or any types of files, which would relocate files based on how frequently a file is used.

SmartTier sub-file object management

SmartTier enables administrators of multi-volume VxFS file systems to manage the placement of file objects as well as entire files on individual volumes.

Using sub-file based movement you can:

- Move a set of ranges of a specified set of files of a specified set of mounts to a desired set of tiers on command.
- Move segments of files using automation to:
    - Monitor a set of files for collecting I/O statistics
    - Periodically collect and persist the statistics, cluster-wide if applicable
    - Periodically enforce the ranges of the registered sets of files based on their relative frequency of access to a desired set of tiers
    - Track the historical movements of those ranges

How the SmartTier policy works with the shared extents

The SmartTier enforcement operation ignores moving the shared extents. For example, consider a file A that contains some shared and private extents that belong to device 1. If the user sets a policy that states that all the extents of the file A must be allocated to device 2, the SmartTier enforcement operation moves all the non-shared extents from device 1 to device 2. However, the SmartTier enforcement operation ignores moving the shared extents. As a result, the file A still contains shared extents that belong to device 1. This occurs even after the successful execution of the SmartTier enforcement operation.

On the other hand, any subsequent new allocation on behalf of the file A adheres to the preset SmartTier policy. Since the copy-on-write or unshare operation requires a new allocation, the SmartTier enforcement operation complies with the preset policy. If a write operation on the file A writes to shared extents, new allocations as part of copy-on-write operation is done from device 2. This behaviour adheres to the preset SmartTier policy.

SmartTier in a High Availability (HA) environment

Cluster Server does not provide a bundled agent for volume sets. If issues arise with volumes or volume sets, the issues can only be detected at the DiskGroup and Mount resource levels.

The DiskGroup agent brings online, takes offline, and monitors a Veritas Volume Manager (VxVM) disk group. This agent uses VxVM commands. When the value of the StartVolumes and StopVolumes attributes are both 1, the DiskGroup agent onlines and offlines the volumes during the import and deport operations of the disk group. When using volume sets, set StartVolumes and StopVolumes attributes of the DiskGroup resource that contains the volume are set to 1. If a file system is created on the volume set, use a Mount resource to mount the volume set.

The Mount agent brings online, takes offline, and monitors a file system or NFS client mount point.

For additional information, see the *Cluster Server Bundled Agents Reference Guide*.

Chapter 26

Creating and administering volume sets

This chapter includes the following topics:

About volume sets

Veritas File System (VxFS) uses volume sets to implement its Multi-Volume Support and SmartTier features.

See "About SmartTier" on page 498.

Veritas Volume Manager (VxVM) provides the `vxvset` command to create and administer volume sets.

See the vxvset(1M) manual page.

Volume sets have the following limitations:

- A maximum of 2048 volumes can be configured in a volume set.
- Only a Veritas File System is supported on a volume set.

- The first volume (index 0) in a volume set must be larger than the sum of the total volume size divided by 4000, the size of the VxFS intent log, and 1MB. Volumes 258 MB or larger should always suffice.
- Raw I/O from and to a volume set is not supported.
- Raw I/O from and to the component volumes of a volume set is supported under certain conditions.
  See "Managing raw device nodes of component volumes" on page 508.
- Volume sets can be used in place of volumes with the following `vxsnap` operations on instant snapshots: `addmir`, `dis`, `make`, `prepare`, `reattach`, `refresh`, `restore`, `rmmir`, `split`, `syncpause`, `syncresume`, `syncstart`, `syncstop`, `syncwait`, and `unprepare`. The third-mirror break-off usage model for full-sized instant snapshots is supported for volume sets provided that sufficient plexes exist for each volume in the volume set.
  For more information about snapshots, see the *Veritas InfoScale Solutions Guide*.
- A full-sized snapshot of a volume set must itself be a volume set with the same number of volumes and the same volume index numbers as the parent. The corresponding volumes in the parent and snapshot volume sets are also subject to the same restrictions as apply between standalone volumes and their snapshots.

Creating a volume set

To create a volume set for use by Veritas File System (VxFS), use the following command:

```
# vxvset [-g diskgroup] -t vxfs make volset
    volume
```

Here *volset* is the name of the volume set, and *volume* is the name of the first volume in the volume set. The `-t vxfs` option creates the volume set configured for use by VxFS. You must create the volume before running the command. `vxvset` will not automatically create the volume.

For example, to create a volume set named `myvset` that contains the volume `vol1`, in the disk group `mydg`, you would use the following command:

```
# vxvset -g mydg -t vxfs make myvset vol1
```

Adding a volume to a volume set

Having created a volume set containing a single volume, you can use the following command to add further volumes to the volume set:

```
# vxvset [-g diskgroup] [-f] addvol volset
    volume
```

For example, to add the volume `vol2`, to the volume set `myvset`, use the following command:

```
# vxvset -g mydg addvol myvset vol2
```

Warning: The `-f` (force) option must be specified if the volume being added, or any volume in the volume set, is either a snapshot or the parent of a snapshot. Using this option can potentially cause inconsistencies in a snapshot hierarchy if any of the volumes involved in the operation is already in a snapshot chain.

Removing a volume from a volume set

To remove a component volume from a volume set, use the following command:

```
# vxvset [-g diskgroup] [-f] rmvol volset
    volume
```

For example, the following commands remove the volumes, `vol1` and `vol2`, from the volume set `myvset`:

```
# vxvset -g mydg -f rmvol myvset vol1
# vxvset -g mydg -f rmvol myvset vol2
```

Removing the final volume deletes the volume set.

Warning: The `-f` (force) option must be specified if the volume being removed, or any volume in the volume set, is either a snapshot or the parent of a snapshot. Using this option can potentially cause inconsistencies in a snapshot hierarchy if any of the volumes involved in the operation is already in a snapshot chain.

Listing details of volume sets

To list the details of the component volumes of a volume set, use the following command:

```
# vxvset [-g diskgroup] list [volset]
```

If the name of a volume set is not specified, the command lists the details of all volume sets in a disk group, as shown in the following example:

```
# vxvset -g mydg list

NAME          GROUP      NVOLS      CONTEXT
set1          mydg           3      -
set2          mydg           2      -
```

To list the details of each volume in a volume set, specify the name of the volume set as an argument to the command:

```
# vxvset -g mydg list set1

VOLUME            INDEX          LENGTH         KSTATE        CONTEXT
vol1                  0        12582912        ENABLED        -
vol2                  1        12582912        ENABLED        -
vol3                  2        12582912        ENABLED        -
```

The context field contains details of any string that the application has set up for the volume or volume set to tag its purpose.

Stopping and starting volume sets

Under some circumstances, you may need to stop and restart a volume set. For example, a volume within the set may have become detached, as shown here:

```
# vxvset -g mydg list set1

VOLUME            INDEX          LENGTH         KSTATE        CONTEXT
vol1                  0        12582912       DETACHED        -
vol2                  1        12582912        ENABLED        -
vol3                  2        12582912        ENABLED        -
```

To stop and restart one or more volume sets, use the following commands:

```
# vxvset [-g diskgroup] stop volset ...
# vxvset [-g diskgroup] start volset ...
```

For the example given previously, the effect of running these commands on the component volumes is shown below:

```
# vxvset -g mydg stop set1

# vxvset -g mydg list set1

VOLUME          INDEX         LENGTH        KSTATE       CONTEXT
vol1                0       12582912      DISABLED       -
vol2                1       12582912      DISABLED       -
vol3                2       12582912      DISABLED       -

# vxvset -g mydg start set1

# vxvset -g mydg list set1

VOLUME          INDEX         LENGTH        KSTATE       CONTEXT
vol1                0       12582912       ENABLED       -
vol2                1       12582912       ENABLED       -
vol3                2       12582912       ENABLED       -
```

Managing raw device nodes of component volumes

To guard against accidental file system and data corruption, the device nodes of the component volumes are configured by default not to have raw and block entries in the `/dev/vx/rdsk/`*`diskgroup`* and `/dev/vx/dsk/`*`diskgroup`* directories. As a result, applications are prevented from directly reading from or writing to the component volumes of a volume set.

If some applications, such as the raw volume backup and restore feature of the Veritas NetBackup™ software, need to read from or write to the component volumes by accessing raw device nodes in the `/dev/vx/rdsk/`*`diskgroup`* directory, this is supported by specifying additional command-line options to the `vxvset` command. Access to the block device nodes of the component volumes of a volume set is unsupported.

Warning: Writing directly to or reading from the raw device node of a component volume of a volume set should only be performed if it is known that the volume's data will not otherwise change during the period of access.

All of the raw device nodes for the component volumes of a volume set can be created or removed in a single operation. Raw device nodes for any volumes added to a volume set are created automatically as required, and inherit the access mode of the existing device nodes.

Access to the raw device nodes for the component volumes can be configured to be read-only or read-write. This mode is shared by all the raw device nodes for the component volumes of a volume set. The read-only access mode implies that any writes to the raw device will fail, however writes using the `ioctl` interface or by VxFS to update metadata are not prevented. The read-write access mode allows direct writes via the raw device. The access mode to the raw device nodes of a volume set can be changed as required.

The presence of raw device nodes and their access mode is persistent across system reboots.

Note the following limitations of this feature:

- The disk group version must be 140 or greater.
- Access to the raw device nodes of the component volumes of a volume set is only supported for private disk groups; it is not supported for shared disk groups in a cluster.

Enabling raw device access when creating a volume set

To enable raw device access when creating a volume set, use the following form of the `vxvset make` command:

```
# vxvset [-g diskgroup] -o makedev=on \
  [-o compvol_access={read-only|read-write}] \
  [-o index] [-c "ch_addopt"] make vset
    vol [index]
```

The `-o makedev=on` option enables the creation of raw device nodes for the component volumes at the same time that the volume set is created. The default setting is `off`.

If the `-o compvol_access=read-write` option is specified, direct writes are allowed to the raw device of each component volume. If the value is set to `read-only`, only reads are allowed from the raw device of each component volume.

If the `-o makedev=on` option is specified, but `-o compvol_access` is not specified, the default access mode is `read-only`.

If the `vxvset addvol` command is subsequently used to add a volume to a volume set, a new raw device node is created in */dev/vx/rdsk/diskgroup* if the value of the `makedev` attribute is currently set to `on`. The access mode is determined by the current setting of the `compvol_access` attribute.

The following example creates a volume set, `myvset1`, containing the volume, `myvol1`, in the disk group, `mydg`, with raw device access enabled in read-write mode:

```
# vxvset -g mydg -o makedev=on -o compvol_access=read-write \
  make myvset1 myvol1
```

Displaying the raw device access settings for a volume set

You can use the `vxprint -m` command to display the current settings for a volume set. If the `makedev` attribute is set to `on`, one of the following strings is displayed in the output:

`vset_devinfo=on:read-only`	Raw device nodes in read-only mode.
`vset_devinfo=on:read-write`	Raw device nodes in read-write mode.

A string is not displayed if `makedev` is set to `off`.

If the output from the `vxprint -m` command is fed to the `vxmake` command to recreate a volume set, the `vset_devinfo` attribute must set to `off`. Use the vxvset set command to re-enable raw device access with the desired access mode.

See "Controlling raw device access for an existing volume set" on page 510.

Controlling raw device access for an existing volume set

To enable or disable raw device node access for an existing volume set, use the following command:

```
# vxvset [-g diskgroup] [-f] set makedev={on|off} vset
```

The `makedev` attribute can be specified to the `vxvset set` command to create (`makedev=on`) or remove (`makedev=off`) the raw device nodes for the component volumes of a volume set. If any of the component volumes are open, the `-f` (force) option must be specified to set the attribute to `off`.

Specifying `makedev=off` removes the existing raw device nodes from the *`/dev/vx/rdsk/diskgroup`* directory.

If the `makedev` attribute is set to `off`, and you use the `mknod` command to create the raw device nodes, you cannot read from or write to those nodes unless you set the value of `makedev` to `on`.

The syntax for setting the `compvol_access` attribute on a volume set is:

```
# vxvset [-g diskgroup] [-f] set \
  compvol_access={read-only|read-write} vset
```

The compvol_access attribute can be specified to the vxvset set command to change the access mode to the component volumes of a volume set. If any of the component volumes are open, the -f (force) option must be specified to set the attribute to read-only.

The following example sets the makedev=on and compvol_access=read-only attributes on a volume set, myvset2, in the disk group, mydg:

```
# vxvset -g mydg set makedev=on myvset2
```

The next example sets the compvol_access=read-write attribute on the volume set, myvset2:

```
# vxvset -g mydg set compvol_access=read-write myvset2
```

The final example removes raw device node access for the volume set, myvset2:

```
# vxvset -g mydg set makedev=off myvset2
```

Chapter 27

Multi-volume file systems

This chapter includes the following topics:

About multi-volume file systems

Veritas File System (VxFS) provides multi-volume file system (MVFS) support when used in conjunction with Veritas Volume Manager. Using MVFS, a single file system can be created over multiple volumes, each volume having its own properties. For example, it is possible to place metadata on mirrored storage while placing file data on better-performing volume types such as RAID-1+0 (striped and mirrored). The volume must be in the same disk group as the volume set, and it cannot already be a member of another volume set.

The MVFS feature also allows file systems to reside on different classes of devices, so that a file system can be supported from both inexpensive disks and from expensive arrays. Using the MVFS administrative interface, you can control which data goes on which volume types.

Note: Multi-volume file system support is available only on file systems using disk layout Version 7 or later.To determine the disk layout version of the file system see the vxupgrade(1M) manual page.

About volume types

Veritas File System (VxFS) utilizes two types of volumes, one of which contains only data, referred to as `dataonly`, and the other of which can contain metadata or data, referred to as `metadataok`.

Data refers to direct extents, which contain user data, of regular files and named data streams in a file system.

Metadata refers to all extents that are not regular file or named data stream extents. This includes certain files that appear to be regular files, but are not, such as the File Change Log file.

A volume availability flag is set to specify if a volume is `dataonly` or `metadataok`. The volume availability flag can be set, cleared, and listed with the `fsvoladm` command.

See the `fsvoladm`(1M) manual page.

Features implemented using multi-volume file system (MVFS) support

The following features can be implemented using MVFS:

- Controlling where files are stored can be selected at multiple levels so that specific files or file hierarchies can be stored on different volumes. This functionality is available in the Veritas File System SmartTier feature.See “About SmartTier” on page 528.
- Placing the VxFS intent log on a separate volume to minimize disk head movement and thereby increase performance.
- Separating Storage Checkpoints so that data allocated to a Storage Checkpoint is isolated from the rest of the file system.
- Separating metadata from file data.

- Encapsulating volumes so that a volume appears in the file system as a file. This is particularly useful for databases that are running on raw volumes.
- Guaranteeing that a dataonly volume being unavailable does not cause a metadataok volume to be unavailable.
  See "Volume availability" on page 514.

To use the multi-volume file system features, Veritas Volume Manager must be installed and the volume set feature must be accessible. The volume set feature is separately licensed.

Volume availability

MVFS guarantees that a dataonly volume being unavailable does not cause a metadataok volume to be unavailable. This allows you to mount a multi-volume file system even if one or more component dataonly volumes are missing.

The volumes are separated by whether metadata is allowed on the volume. An I/O error on a dataonly volume does not affect access to any other volumes. All VxFS operations that do not access the missing `dataonly` volume function normally.

Some VxFS operations that do not access the missing `dataonly` volume and function normally include the following:

- Mounting the multi-volume file system, regardless if the file system is read-only or read/write.
- Kernel operations.
- Performing a `fsck` replay. Logged writes are converted to normal writes if the corresponding volume is `dataonly`.
- Performing a full `fsck`.
- Using all other commands that do not access data on a missing volume.

Some operations that could fail if a `dataonly` volume is missing include:

- Reading or writing file data if the file's data extents were allocated from the missing `dataonly` volume.
- Using the `vxdump` command.

Volume availability is supported only on a file system with disk layout Version 7 or later.

Note: Do not mount a multi-volume system with the `ioerror=disable` or `ioerror=wdisable` mount options if the volumes have different availability properties. Veritas recommends the `ioerror=mdisable` mount option both for cluster mounts and for local mounts.

Creating multi-volume file systems

When a multi-volume file system is created, all volumes are `dataonly`, except volume zero, which is used to store the file system's metadata. The volume availability flag of volume zero cannot be set to `dataonly`.

As metadata cannot be allocated from `dataonly` volumes, enough metadata space should be allocated using `metadataok` volumes. The "file system out of space" error occurs if there is insufficient metadata space available, even if the `df` command shows that there is free space in the file system. The `fsvoladm` command can be used to see the free space in each volume and set the availability flag of the volume.

Unless otherwise specified, VxFS commands function the same on multi-volume file systems as the commands do on single-volume file systems.

The following procedure is an example of creating a multi-volume file system.

Example of creating a multi-volume file system

1. After a volume set is created, create a VxFS file system by specifying the volume set name as an argument to `mkfs`:

   See "Creating a volume set" on page 505.

   ```
   # mkfs -t vxfs /dev/vx/rdsk/dg1/myvset
     version 16 layout
     12288 sectors, 6144 blocks of size 1024, log size 256 blocks
     rcq size 1024 blocks
     largefiles supported
     maxlink supported
     WORM not supported
   ```

   After the file system is created, VxFS allocates space from the different volumes within the volume set.

2. List the component volumes of the volume set using of the `fsvoladm` command:

   ```
   # mount -t vxfs /dev/vx/dsk/dg1/myvset /mnt1
   # fsvoladm -H list /mnt1
   devid    size     used     avail    name
   0        20 GB    10 GB    10 GB    vol1
   1        30 TB    10 TB    20 TB    vol2
   ```

3. Add a new volume by adding the volume to the volume set, then adding the volume to the file system:

   ```
   # vxassist -g dg1 make vol5 50m
   # vxvset -g dg1 addvol myvset vol5
   # fsvoladm add /mnt1 vol5 50m
   # fsvoladm -H list /mnt1
   ddevid    size         used        avail     name
    0       10 GB       74.6 MB     9.93 GB    vol1
    1       20 GB        16 KB      20.0 GB    vol2
    2       50 MB        16 KB      50.0 MB    vol5
   ```

4. List the volume availability flags using the `fsvoladm` command:

```
  # fsvoladm queryflags /mnt1
volname       flags
vol1          metadataok
vol2          dataonly
vol5          dataonly
```

5. Increase the metadata space in the file system using the `fsvoladm` command:

```
  # fsvoladm clearflags dataonly /mnt1 vol2
  # fsvoladm queryflags /mnt1
volname       flags
vol1          metadataok
vol2          metadataok
vol5          dataonly
```

Converting a single volume file system to a multi-volume file system

The following procedure converts a traditional, single volume file system, `/mnt1`, on a single volume `vol1` in the disk group `dg1` to a multi-volume file system.

To convert a single volume file system to a multi-volume file system

1. Determine the version of the volume's disk group:

```
# vxdg list dg1 | grep version: | awk '{ print $2 }'
90
```

2. If the version is less than 110, upgrade the disk group:

```
# vxdg upgrade dg1
```

3. Determine the disk layout version of the file system:

```
# vxupgrade /mnt1
/mnt1: vxfs file system version 17 layout
```

4. If the disk layout version is less than 13, upgrade to a disk layout version that is greater than or equal to 13, as per the requirement.

   Currently we can upgrade up to the disk layout version 17.

   ```
   # vxupgrade -n 13 /mnt1
   # vxupgrade -n 14 /mnt1
   # vxupgrade -n 15 /mnt1
   # vxupgrade -n 16 /mnt1
   # vxupgrade -n 17 /mnt1
   ```

5. Unmount the file system:

   ```
   # umount /mnt1
   ```

6. Convert the volume into a volume set:

   ```
   # vxvset -g dg1 make vset1 vol1
   ```

7. Edit the `/etc/fstab` file to replace the volume device name, `vol1`, with the volume set name, `vset1`.

8. Mount the file system:

   ```
   # mount -t vxfs /dev/vx/dsk/dg1/vset1 /mnt1
   ```

9. As necessary, create and add volumes to the volume set:

   ```
   # vxassist -g dg1 make vol2 256M
   # vxvset -g dg1 addvol vset1 vol2
   ```

10. Set the placement class tags on all volumes that do not have a tag:

    ```
    # vxassist -g dg1 settag vol1 vxfs.placement_class.tier1
    # vxassist -g dg1 settag vol2 vxfs.placement_class.tier2
    ```

Adding a volume to and removing a volume from a multi-volume file system

Use the `fsvoladm` command to perform the following functions:

- Adding a volume to a multi-volume file system
- Removing a volume from a multi-volume file system

Use the `fsck` command to perform the following function:

- Forcibly removing a volume in a multi-volume file system

Use the `vxassist` command to perform the following function:

- Moving volume 0 in a multi-volume file system

Adding a volume to a multi-volume file system

Use the `fsvoladm add` command to add a volume to a multi-volume file system.

To add a volume to a multi-volume file system

- ◆ Add a new volume to a multi-volume file system:

```
# fsvoladm add /mnt1 vol2 256m
```

Removing a volume from a multi-volume file system

Use the `fsvoladm remove` command to remove a volume from a multi-volume file system. The `fsvoladm remove` command fails if the volume being removed is the first volume with metadataok flag in any allocation policy.

To remove a volume from a multi-volume file system

- ◆ Remove a volume from a multi-volume file system:

```
# fsvoladm remove /mnt1 vol2
```

Forcibly removing a volume in a multi-volume file system

If you must forcibly remove a volume from a file system, such as if a volume is permanently destroyed and you want to clean up the dangling pointers to the lost volume, use the `fsck -o zapvol=volname` command. The `zapvol` option performs a full file system check and zaps all inodes that refer to the specified volume. The `fsck` command prints the inode numbers of all files that the command destroys; the file names are not printed. The `zapvol` option only affects regular files if used on a dataonly volume. However, it could destroy structural files if used on a `metadataok` volume, which can make the file system unrecoverable. Therefore, the `zapvol` option should be used with caution on `metadataok` volumes.

Moving volume 0 in a multi-volume file system

Volume 0 in a multi-volume file system cannot be removed from the file system, but you can move volume 0 to different storage using the `vxassist move` command.

The vxassist command creates any necessary temporary mirrors and cleans up the mirrors at the end of the operation.

To move volume 0

- Move volume 0. For example, to move volume vol1 of the disk group mydg from disk1 to disk4:

```
# vxassist -g mydg move vol1 \!disk1 disk4
```

Volume encapsulation

Note: Root Disk Encapsulation (RDE) is not supported on Linux from 7.3.1 onwards.

Multi-volume file system support enables the ability to encapsulate an existing raw volume and make the volume contents appear as a file in the file system.

Encapsulating a volume involves the following actions:

- Adding the volume to an existing volume set.
- Adding the volume to the file system using fsvoladm.

Encapsulating a volume

Note: Root Disk Encapsulation (RDE) is not supported in Linux from 7.3.1 onwards.

The following example illustrates how to encapsulate a volume.

To encapsulate a volume

1 List the volumes:

```
  # vxvset -g dg1 list myvset

VOLUME     INDEX  LENGTH       KSTATE      CONTEXT
vol3        0     104857600    ENABLED       -
vol2        1     104857600    ENABLED       -
```

The volume set has two volumes.

2 Create a third volume and copy the passwd file to the third volume:

```
  # vxassist -g dg1 make dbvol 100m
  # dd if=/etc/passwd of=/dev/vx/rdsk/dg1/dbvol count=1
  1+0 records in
  1+0 records out
```

The third volume will be used to demonstrate how the volume can be accessed as a file, as shown later.

3 Create a file system on the volume set:

```
# /opt/VRTS/bin/mkfs /dev/vx/rdsk/dg1/myvset
  version 16 layout
  134217728 sectors, 67108864 blocks of size 1024, log size 65536 blocks
  rcq size 4096 blocks
  largefiles supported
  maxlink  supported
```

4 Mount the volume set:

```
  # /opt/VRTS/bin/mount /dev/vx/dsk/dg1/myvset /mnt1
```

5 Add the new volume to the volume set:

```
  # vxvset -g dg1 addvol myvset dbvol
```

6 Encapsulate `dbvol`:

```
# fsvoladm encapsulate /mnt1/dbfile dbvol 100m
# ls -l /mnt1/dbfile
-rw------- 1 root other 104857600 May 22 11:30 /mnt1/dbfile
```

7 Examine the contents of `dbfile` to see that it can be accessed as a file:

```
# head -2 /mnt1/dbfile
root:x:0:1:Super-User:/:/sbin/sh
daemon:x:1:1::/:
```

The passwd file that was written to the raw volume is now visible in the new file.

Note: If the encapsulated file is changed in any way, such as if the file is extended, truncated, or moved with an allocation policy or resized volume, or the volume is encapsulated with a bias, the file cannot be de-encapsulated.

Deencapsulating a volume

Note: Root Disk Encapsulation (RDE) is not supported on Linux from 7.3.1 onwards.

The following example illustrates how to deencapsulate a volume.

To deencapsulate a volume

1 List the volumes:

```
# vxvset -g dg1 list myvset
VOLUME   INDEX   LENGTH   KSTATE   CONTEXT
vol1     0       102400   ACTIVE  -
vol2     1       102400   ACTIVE  -
dbvol    2       102400   ACTIVE  -
```

The volume set has three volumes.

2 Deencapsulate `dbvol`:

```
# fsvoladm deencapsulate /mnt1/dbfile
```

Reporting file extents

Multi-volume file system feature provides the capability for file-to-volume mapping and volume-to-file mapping via the fsmap and fsvmap commands. The fsmap command reports the volume name, logical offset, and size of data extents, or the volume name and size of indirect extents associated with a file on a multi-volume file system. The fsvmap command maps volumes to the files that have extents on those volumes.

See the fsmap(1M) and fsvmap(1M) manual pages.

The fsmap command requires open() permission for each file or directory specified. Root permission is required to report the list of files with extents on a particular volume.

The following examples show typical uses of the fsmap and fsvmap commands.

Example of using the fsmap command

- Use the find command to descend directories recursively and run fsmap on the list of files:

```
# find . | fsmap -
Volume  Extent Type    File
vol2    Data           ./file1
vol1    Data           ./file2
```

Example of using the fsvmap command

1 Report the extents of files on multiple volumes:

```
# fsvmap /dev/vx/rdsk/fstest/testvset vol1 vol2
vol1  /.
vol1  /ns2
vol1  /ns3
vol1  /file1
vol2  /file1
vol2  /file2
```

2 Report the extents of files that have either data or metadata on a single volume in all Storage Checkpoints, and indicate if the volume has file system metadata:

```
# fsvmap -mvC /dev/vx/rdsk/fstest/testvset vol1
Meta  Structural   vol1  //volume has filesystem metadata//
Data  UNNAMED      vol1  /.
Data  UNNAMED      vol1  /ns2
Data  UNNAMED      vol1  /ns3
Data  UNNAMED      vol1  /file1
Meta  UNNAMED      vol1  /file1
```

Load balancing

An allocation policy with the balance allocation order can be defined and assigned to files that must have their allocations distributed at random between a set of specified volumes. Each extent associated with these files are limited to a maximum size that is defined as the required chunk size in the allocation policy. The distribution of the extents is mostly equal if none of the volumes are full or disabled.

Load balancing allocation policies can be assigned to individual files or for all files in the file system. Although intended for balancing data extents across volumes, a load balancing policy can be assigned as a metadata policy if desired, without any restrictions.

Note: If a file has both a fixed extent size set and an allocation policy for load balancing, certain behavior can be expected. If the chunk size in the allocation policy is greater than the fixed extent size, all extents for the file are limited by the chunk size. For example, if the chunk size is 16 MB and the fixed extent size is 3 MB, then the largest extent that satisfies both the conditions is 15 MB. If the fixed extent size is larger than the chunk size, all extents are limited to the fixed extent size. For example, if the chunk size is 2 MB and the fixed extent size is 3 MB, then all extents for the file are limited to 3 MB.

Defining and assigning a load balancing allocation policy

The following example defines a load balancing policy and assigns the policy to the file, `/mnt/file.db`.

To define and assign the policy

1. Define the policy by specifying the `-o balance` and `-c` options:

   ```
   # fsapadm define -o balance -c 2m /mnt loadbal vol1 vol2 vol3 vol4
   ```

   ```
   # fsapadm define /mnt meta vol1
   ```

2. Assign the policy:

   ```
   # fsapadm assignfile /mnt/file.db loadbal meta
   ```

3. Enforce the file:

   ```
   # fsapadm enforcefile /mnt/file.db
   ```

Rebalancing extents

Extents can be rebalanced by strictly enforcing the allocation policy. Rebalancing is generally required when volumes are added or removed from the policy or when the chunk size is modified. When volumes are removed from the volume set, any extents on the volumes being removed are automatically relocated to other volumes within the policy.

The following example redefines a policy that has four volumes by adding two new volumes, removing an existing volume, and enforcing the policy for rebalancing.

To rebalance extents

1. Define the policy by specifying the `-o balance` and `-c` options:

   ```
   # fsapadm define -o balance -c 2m /mnt loadbal vol1 vol2 vol4 \
   vol5 vol6
   ```

2. Enforce the policy:

   ```
   # fsapadm enforcefile -f strict /mnt/filedb
   ```

Converting a multi-volume file system to a single volume file system

Because data can be relocated among volumes in a multi-volume file system, you can convert a multi-volume file system to a traditional, single volume file system by moving all file system data onto a single volume. Such a conversion is useful to users who would like to try using a multi-volume file system or SmartTier, but are not committed to using a multi-volume file system permanently.

See "About SmartTier" on page 528.

There are three restrictions to this operation:

- The single volume must be the first volume in the volume set
- The first volume must have sufficient space to hold all of the data and file system metadata
- The volume cannot have any allocation policies that restrict the movement of data

The following procedure converts an existing multi-volume file system, `/mnt1`, of the volume set `vset1`, to a single volume file system, `/mnt1`, on volume `vol1` in disk group `dg1`.

Note: Steps 5, 6, 7, and 8 are optional, and can be performed if you prefer to remove the wrapper of the volume set object.

Converting to a single volume file system

1 Determine if the first volume in the volume set, which is identified as device number 0, has the capacity to receive the data from the other volumes that will be removed:

```
# df /mnt1
/mnt1  (/dev/vx/dsk/dg1/vol1):16777216 blocks  3443528 files
```

2 If the first volume does not have sufficient capacity, grow the volume to a sufficient size:

```
# fsvoladm resize /mnt1 vol1 150g
```

3 Remove all existing allocation policies:

```
# fsppadm unassign /mnt1
```

4 Remove all volumes except the first volume in the volume set:

```
# fsvoladm remove /mnt1 vol2
# vxvset -g dg1 rmvol vset1 vol2
# fsvoladm remove /mnt1 vol3
# vxvset -g dg1 rmvol vset1 vol3
```

Before removing a volume, the file system attempts to relocate the files on that volume. Successful relocation requires space on another volume, and no allocation policies can be enforced that pin files to that volume. The time for the command to complete is proportional to the amount of data that must be relocated.

5 Unmount the file system:

```
# umount /mnt1
```

6 Remove the volume from the volume set:

```
# vxvset -g dg1 rmvol vset1 vol1
```

7 Edit the `/etc/fstab` file to replace the volume set name, `vset1`, with the volume device name, `vol1`.

8 Mount the file system:

```
# mount -t vxfs /dev/vx/dsk/dg1/vol1 /mnt1
```

Chapter 28

Administering SmartTier

This chapter includes the following topics:

About SmartTier

Veritas File System (VxFS) uses multi-tier online storage by way of the SmartTier feature, which functions on top of multi-volume file systems. Multi-volume file systems are file systems that occupy two or more virtual volumes. The collection of volumes is known as a volume set. A volume set is made up of disks or disk array LUNs belonging to a single Veritas Volume Manager (VxVM) disk group. A multi-volume file system presents a single name space, making the existence of multiple volumes transparent to users and applications. Each volume retains a separate identity for administrative purposes, making it possible to control the locations to which individual files are directed.

See "About multi-volume file systems" on page 512.

Note: Some of the commands have changed or been removed between the 4.1 release and the current release to make placement policy management more user-friendly. The following commands have been removed: `fsrpadm`, `fsmove`, and `fssweep`. The output of the `queryfile`, `queryfs`, and `list` options of the `fsapadm` command now print the allocation order by name instead of number.

In the previous VxFS 5.x releases, SmartTier was known as Dynamic Storage Tiering.

SmartTier allows administrators of multi-volume VxFS file systems to manage the placement of files and the placement of portions of files on individual volumes in a volume set by defining placement policies. Placement policies control both initial file location and the circumstances under which existing files are relocated. These placement policies cause the files to which they apply to be created and extended on specific subsets of a file system's volume set, known as placement classes. The files are relocated to volumes in other placement classes when they meet the specified naming, timing, access rate, and storage capacity-related conditions.

You make a VxVM volume part of a placement class by associating a volume tag with it. For file placement purposes, VxFS treats all of the volumes in a placement class as equivalent, and balances space allocation across them. A volume may have more than one tag associated with it. If a volume has multiple tags, the volume belongs to multiple placement classes and is subject to allocation and relocation policies that relate to any of the placement classes. Multiple tagging should be used carefully.

See "Placement classes" on page 531.

VxFS imposes no capacity, performance, availability, or other constraints on placement classes. Any volume may be added to any placement class, no matter what type the volume has nor what types other volumes in the class have. However, a good practice is to place volumes of similar I/O performance and availability in the same placement class.

About compressing files with SmartTier

You can use the SmartTier feature to compress and uncompress files automatically based on the rules defined in a placement policy. SmartTier performs the allocation for compressed or uncompressed extents of the selected files directly from the tier that is specified in the policy. The selected files get compressed or uncompressed while relocating to the specified tier of storage.

You can perform in-place compressing of an entire tier, which compresses all of the uncompressed extents of all of the files on the tier. This operation is useful if a write or append was performed on a file on this tier, which results in the file having some uncompressed extents.

SmartTier uses `gzip` as the default compression algorithm, and 1 MB is the default block size for compression. These default values are not configurable through an XML policy file.

SmartTier can compress and uncompress files as specified by a placement policy in the following ways:

- Compress while relocating files from one tier to another in a multi-volume file system
- Uncompress while relocating files from one tier to another in a multi-volume file system
- Compress in-place in a multi-volume file system
- Uncompress in-place in multi-volume file system
- Compress in-place in single volume file system
- Uncompress in-place in single volume file system
- Compress an entire tier in multi-volume file system
- Uncompress an entire tier in multi-volume file system

See “About compressing files” on page 626.

Supported SmartTier document type definitions

Table 28-1 describes which releases of Veritas File System (VxFS) support specific SmartTier document type definitions (DTDs).

Table 28-1 Supported SmartTier document type definitions

	DTD Version	
VxFS Version	**1.0**	**1.1**
5.0	Supported	Not supported
5.1	Supported	Supported
5.1 SP1	Supported	Supported

Table 28-1 Supported SmartTier document type definitions *(continued)*

VxFS Version	DTD Version	
	1.0	1.1
6.0	Supported	Supported
6.0.1	Supported	Supported

Placement classes

A placement class is a SmartTier attribute of a given volume in a volume set of a multi-volume file system. This attribute is a character string, and is known as a volume tag. A volume can have different tags, one of which can be the placement class. The placement class tag makes a volume distinguishable by SmartTier.

Volume tags are organized as hierarchical name spaces in which periods separate the levels of the hierarchy . By convention, the uppermost level in the volume tag hierarchy denotes the Storage Foundation component or application that uses a tag, and the second level denotes the tag's purpose. SmartTier recognizes volume tags of the form `vxfs.placement_class.`*class_name*. The prefix `vxfs` identifies a tag as being associated with VxFS. The `placement_class` string identifies the tag as a file placement class that SmartTier uses. The *class_name* string represents the name of the file placement class to which the tagged volume belongs. For example, a volume with the tag **vxfs.placement_class.tier1** belongs to placement class `tier1`. Administrators use the `vxassist` command to associate tags with volumes.

See the `vxassist`(1M) manual page.

SmartTier policy rules specify file placement in terms of placement classes rather than in terms of individual volumes. All volumes that belong to a particular placement class are interchangeable with respect to file creation and relocation operations. Specifying file placement in terms of placement classes rather than in terms of specific volumes simplifies the administration of multi-tier storage.

The administration of multi-tier storage is simplified in the following ways:

- Adding or removing volumes does not require a file placement policy change. If a volume with a tag value of `vxfs.placement_class.tier2` is added to a file system's volume set, all policies that refer to `tier2` immediately apply to the newly added volume with no administrative action. Similarly, volumes can be evacuated, that is, have data removed from them, and be removed from a file system without a policy change. The active policy continues to apply to the file system's remaining volumes.

- File placement policies are not specific to individual file systems. A file placement policy can be assigned to any file system whose volume set includes volumes tagged with the tag values (placement classes) named in the policy. This property makes it possible for data centers with large numbers of servers to define standard placement policies and apply them uniformly to all servers with a single administrative action.

Tagging volumes as placement classes

The following example tags the `vsavola` volume as placement class `tier1`, `vsavolb` as placement class `tier2`, `vsavolc` as placement class `tier3`, and `vsavold` as placement class `tier4` using the `vxassist settag` command.

To tag volumes

◆ Tag the volumes as placement classes:

```
# vxassist -g cfsdg settag vsavola vxfs.placement_class.tier1
# vxassist -g cfsdg settag vsavolb vxfs.placement_class.tier2
# vxassist -g cfsdg settag vsavolc vxfs.placement_class.tier3
# vxassist -g cfsdg settag vsavold vxfs.placement_class.tier4
```

Listing placement classes

Placement classes are listed using the `vxassist listtag` command.

See the `vxassist`(1M) manual page.

The following example lists all volume tags, including placment classes, set on a volume `vsavola` in the diskgroup `cfsdg`.

To list placement classes

◆ List the volume tags, including placement classes:

```
# vxassist -g cfsdg listtag vsavola
```

Administering placement policies

A VxFS file placement policy document contains rules by which VxFS creates, relocates, and deletes files, but the placement policy does not refer to specific file systems or volumes. You can create a file system's active file placement policy by assigning a placement policy document to the file system via the `fsppadm` command or the GUI.

See the `fsppadm`(1M) manual page.

Note: Do not run the `fsppadm` command simultaneously from different terminals.

The `lost+found` must exist before you can use the `fsppadm` command.

At most, one file placement policy can be assigned to a VxFS file system at any time. A file system may have no file placement policy assigned to it, in which case VxFS allocates space for new files according to its own internal algorithms.

In systems with Storage Foundation Management Server (SFMS) software installed, file placement policy information is stored in the SFMS database. The SFMS database contains both XML policy documents and lists of hosts and file systems for which each document is the current active policy. When a policy document is updated, SFMS can assign the updated document to all file systems whose current active policies are based on that document. By default, SFMS does not update file system active policies that have been created or modified locally, that is by the hosts that control the placement policies' file systems. If a SFMS administrator forces assignment of a placement policy to a file system, the file system's active placement policy is overwritten and any local changes that had been made to the placement policy are lost.

You can view sample placement policies in the `/opt/VRTSvxfs/etc` directory. These sample placement policies are installed as part of the VxFS rpm installation.

Assigning a placement policy

The following example uses the `fsppadm assign` command to assign the file placement policy represented in the XML policy document `/tmp/policy1.xml` for the file system at mount point `/mnt1`.

To assign a placement policy

◆ Assign a placement policy to a file system:

```
# fsppadm assign /mnt1 /tmp/policy1.xml
```

Unassigning a placement policy

The following example uses the `fsppadm unassign` command to unassign the active file placement policy from the file system at mount point `/mnt1`.

To unassign a placement policy

- Unassign the placement policy from a file system:

```
# fsppadm unassign /mnt1
```

Analyzing the space impact of enforcing a placement policy

The following example uses the `fsppadm analyze` command to analyze the impact if the enforce operation is performed on the file placement policy represented in the XML policy document `/tmp/policy1.xml` for the mount point `/mnt1`. The command builds the I/O temperature database if necessary.

To analyze the space impact of enforcing a placement policy

- Analyze the impact of enforcing the file placement policy represented in the XML policy document `/tmp/policy1.xml` for the mount point `/mnt1`:

```
# fsppadm analyze -F /tmp/policy1.xml -i /mnt1
```

Querying which files will be affected by enforcing a placement policy

The following example uses the `fsppadm query` command to generate a list of files that will be affected by enforcing a placement policy. The command provides details about where the files currently reside, to where the files will be relocated, and which rule in the placement policy applies to the files.

To query which files will be affected by enforcing a placement policy

- Query the files that will be affected:

```
# fsppadm query /mnt1/dir1/dir2 /mnt2 /mnt1/dir3
```

Enforcing a placement policy

Enforcing a placement policy for a file system requires that the policy be assigned to the file system. You must assign a placement policy before it can be enforced.

See "Assigning a placement policy" on page 533.

Enforce operations are logged in a hidden file, `.__fsppadm_enforce.log`, in the `lost+found` directory of the mount point. This log file contains details such as files' previous locations, the files' new locations, and the reasons for the files' relocations. The enforce operation creates the `.__fsppadm_enforce.log` file if the file does not exist. The enforce operation appends the file if the file already exists. The `.__fsppadm_enforce.log` file can be backed up or removed as with a normal file.

You can specify the -F option to specify a placement policy other than the existing active placement policy. This option can be used to enforce the rules given in the specified placement policy for maintenance purposes, such as for reclaiming a LUN from the file system.

You can specify the -p option to specify the number of concurrent threads to be used to perform the fsppadm operation. You specify the *io_nice* parameter as an integer between 1 and 100, with 50 being the default value. A value of 1 specifies 1 slave and 1 master thread per mount. A value of 50 specifies 16 slaves and 1 master thread per mount. A value of 100 specifies 32 slaves and 1 master thread per mount.

You can specify the -C option so that the fsppadm command processes only those files that have some activity stats logged in the File Change Log (FCL) file during the period specified in the placement policy. You can use the -C option only if the policy's ACCESSTEMP or IOTEMP elements use the Prefer criteria.

You can specify the -T option to specify the placement classes that contain files for the fsppadm command to sweep and relocate selectively. You can specify the -T option only if the policy uses the Prefer criteria forIOTEMP.

See the fsppadm(1M) manual page.

The following example uses the fsppadm enforce command to enforce the file placement policy for the file system at mount point /mnt1, and includes the access time, modification time, and file size of the specified paths in the report, /tmp/report.

To enforce a placement policy

- Enforce a placement policy for a file system:

```
# fsppadm enforce -a -r /tmp/report /mnt1
Current Current Relocated Relocated
Class   Volume  Class     Volume    Rule   File
tier3   vole    tier3     vole      a_to_z /mnt1/mds1/d1/file1
tier3   vole    tier3     vole      a_to_z /mnt1/mds1/d1/file2
tier3   vole    tier3     vole      a_to_z /mnt1/mds1/d1/d2/file3
tier3   volf    tier3     volf      a_to_z /mnt1/mds1/d1/d2/file4
.
.
.
Sweep path    : /mnt1
Files moved   : 42
KB moved      : 1267


Tier Name       Size (KB)   Free Before (KB)   Free After (KB)
tier4              524288             524256            524256
tier3              524288             522968            522968
tier2              524288             524256            524256
tier1              524288             502188            501227
```

Validating a placement policy

The following example uses the `fsppadm validate` command to validate the placement policy `policy.xml` against all mounted file systems.

To validate a placement policy against all mounted file systems

- Validate the placement policy:

```
  # fsppadm validate /tmp/policy.xml
```

File placement policy grammar

VxFS allocates and relocates files within a multi-volume file system based on properties in the file system metadata that pertains to the files. Placement decisions may be based on file name, directory of residence, time of last access, access frequency, file size, and ownership. An individual file system's criteria for allocating and relocating files are expressed in the file system's file placement policy.

A VxFS file placement policy defines the desired placement of sets of files on the volumes of a VxFS multi-volume file system. A file placement policy specifies the placement classes of volumes on which files should be created, and where and under what conditions the files should be relocated to volumes in alternate placement classes or deleted. You can create file placement policy documents, which are XML text files, using an XML editor, a text editor, or Veritas Operations Manager (VOM).

See the /opt/VRTSvxfs/etc/placement_policy.dtd file for the overall structure of a placement policy.

File placement policy rules

A VxFS file placement policy consists of one or more rules. Each rule applies to one or more files. The files to which a rule applies are designated in one or more SELECT statements. A SELECT statement designates files according to one or more of four properties: their names or naming patterns, the directories in which they reside, their owners' user names, and their owners' group names.

A file may be designated by more than one rule. For example, if one rule designates files in directory /dir, and another designates files owned by user1, a file in /dir that is owned by user1 is designated by both rules. Only the rule that appears first in the placement policy applies to the file; subsequent rules are ignored.

You can define placement policies that do not encompass the entire file system name space. When a file that is not designated by any rule in its file system's active placement policy is created, VxFS places the file according to its own internal algorithms. To maintain full control over file placement, include a catchall rule at the end of each placement policy document with a SELECT statement that designates files by the naming pattern *. Such a rule designates all files that have not been designated by the rules appearing earlier in the placement policy document.

Two types of rules exist: data and ckpt. The data rule type allows SmartTier to relocate normal data files. The ckpt rule type allows SmartTier to relocate Storage Checkpoints. You specify the rule type by setting the Flags attribute for the rule.

SELECT statement

The VxFS placement policy rule SELECT statement designates the collection of files to which a rule applies.

The following XML snippet illustrates the general form of the SELECT statement:

```
<SELECT>
  <DIRECTORY Flags="directory_flag_value"> value
  </DIRECTORY>
```

```
    <PATTERN Flags="pattern_flag_value"> value </PATTERN>
    <USER> value </USER>
    <GROUP> value </GROUP>
  </SELECT>
```

A SELECT statement may designate files by using the following selection criteria:

<DIRECTORY>

A full path name relative to the file system mount point. The Flags="directory_flag_value" XML attribute must have a value of nonrecursive, denoting that only files in the specified directory are designated, or a value of recursive, denoting that files in all subdirectories of the specified directory are designated. The Flags attribute is mandatory.

The <DIRECTORY> criterion is optional, and may be specified more than once.

<PATTERN>

Either an exact file name or a pattern using a single wildcard character (*). For example, the pattern “abc*" denotes all files whose names begin with “abc". The pattern “abc.*" denotes all files whose names are exactly "abc" followed by a period and any extension. The pattern “*abc" denotes all files whose names end in “abc", even if the name is all or part of an extension. The pattern “*.abc" denotes files of any name whose name extension (following the period) is “abc". The pattern “ab*c" denotes all files whose names start with “ab" and end with “c". The first "*" character is treated as a wildcard, while any subsequent "*" characters are treated as literal text. The pattern cannot contain "/".

The wildcard character matches any character, including ".", "?", and "[", unlike using the wildcard in a shell.

The Flags="pattern_flag_value" XML attribute is optional, and if specified can only have a value of recursive. Specify Flags="recursive" only if the pattern is a directory. If Flags is not specified, the default attribute value is nonrecursive. If Flags="recursive" is specified, the enclosing selection criteria selects all files in any component directory that is anywhere below the directory specified by <DIRECTORY> if the component directory matches the pattern and either of the following is true:

- <DIRECTORY> is specified and has the recursive flag.
- <DIRECTORY> is not specified and the directory is anywhere in the file system.

If the pattern contains the wildcard character (*), wildcard character matching is performed.

The <PATTERN> criterion is optional, and may be specified more than once. Only one value can be specified per <PATTERN> element.

`<USER>`	User name of the file's owner. The user number cannot be specified in place of the name. The `<USER>` criterion is optional, and may be specified more than once.
`<GROUP>`	Group name of the file's owner. The group number cannot be specified in place of the group name. The `<GROUP>` criterion is optional, and may be specified more than once.

One or more instances of any or all of the file selection criteria may be specified within a single `SELECT` statement. If two or more selection criteria of different types are specified in a single statement, a file must satisfy one criterion of each type to be selected.

In the following example, only files that reside in either the `ora/db` or the `crash/dump` directory, and whose owner is either `user1` or `user2` are selected for possible action:

```
<SELECT>
  <DIRECTORY Flags="nonrecursive">ora/db</DIRECTORY>
  <DIRECTORY Flags="nonrecursive">crash/dump</DIRECTORY>
  <USER>user1</USER>
  <USER>user2</USER>
</SELECT>
```

A rule may include multiple `SELECT` statements. If a file satisfies the selection criteria of one of the `SELECT` statements, it is eligible for action.

In the following example, any files owned by either `user1` or `user2`, no matter in which directories they reside, as well as all files in the `ora/db` or `crash/dump` directories, no matter which users own them, are eligible for action:

```
<SELECT>
  <DIRECTORY Flags="nonrecursive">ora/db</DIRECTORY>
  <DIRECTORY Flags="nonrecursive">crash/dump</DIRECTORY>
</SELECT>
<SELECT>
  <USER>user1</USER>
  <USER>user2</USER>
</SELECT>
```

When VxFS creates new files, VxFS applies active placement policy rules in the order of appearance in the active placement policy's XML source file. The first rule in which a `SELECT` statement designates the file to be created determines the file's placement; no later rules apply. Similarly, VxFS scans the active policy rules on behalf of each file when relocating files, stopping the rules scan when it reaches the first rule containing a SELECT statement that designates the file. This behavior holds true even if the applicable rule results in no action. Take for example a policy rule that indicates that .dat files inactive for 30 days should be relocated, and a later rule indicates that .dat files larger than 10 megabytes should be relocated. A 20 megabyte .dat file that has been inactive for 10 days will not be relocated because the earlier rule applied. The later rule is never scanned.

A placement policy rule's action statements apply to all files designated by any of the rule's SELECT statements. If an existing file is not designated by a SELECT statement in any rule of a file system's active placement policy, then SmartTier does not relocate or delete the file. If an application creates a file that is not designated by a SELECT statement in a rule of the file system's active policy, then VxFS places the file according to its own internal algorithms. If this behavior is inappropriate, the last rule in the policy document on which the file system's active placement policy is based should specify <PATTERN>*</PATTERN> as the only selection criterion in its SELECT statement, and a CREATE statement naming the desired placement class for files not selected by other rules.

CREATE statement

A CREATE statement in a file placement policy rule specifies one or more placement classes of volumes on which VxFS should allocate space for new files to which the rule applies at the time the files are created. You can specify only placement classes, not individual volume names, in a CREATE statement.

A file placement policy rule may contain at most one CREATE statement. If a rule does not contain a CREATE statement, VxFS places files designated by the rule's SELECT statements according to its internal algorithms. However, rules without CREATE statements can be used to relocate or delete existing files that the rules' SELECT statements designate.

The following XML snippet illustrates the general form of the CREATE statement:

```
<CREATE>
  <ON Flags="flag_value">
    <DESTINATION>
      <CLASS> placement_class_name </CLASS>
      <BALANCE_SIZE Units="units_specifier"> chunk_size
      </BALANCE_SIZE>
    </DESTINATION>
    <DESTINATION> additional_placement_class_specifications
    </DESTINATION>
  </ON>
</CREATE>
```

A CREATE statement includes a single <ON> clause, in which one or more <DESTINATION> XML elements specify placement classes for initial file allocation in order of decreasing preference. VxFS allocates space for new files to which a rule applies on a volume in the first class specified, if available space permits. If space cannot be allocated on any volume in the first class, VxFS allocates space on a volume in the second class specified if available space permits, and so forth.

If space cannot be allocated on any volume in any of the placement classes specified, file creation fails with an ENOSPC error, even if adequate space is available elsewhere in the file system's volume set. This situation can be circumvented by specifying a Flags attribute with a value of "any" in the <ON> clause. If <ON Flags="any"> is specified in a CREATE statement, VxFS first attempts to allocate space for new files to which the rule applies on the specified placement classes. Failing that, VxFS resorts to its internal space allocation algorithms, so file allocation does not fail unless there is no available space any-where in the file system's volume set.

The Flags="any" attribute differs from the catchall rule in that this attribute applies only to files designated by the SELECT statement in the rule, which may be less inclusive than the <PATTERN>*</PATTERN> file selection specification of the catchall rule.

In addition to the placement class name specified in the <CLASS> sub-element, a <DESTINATION> XML element may contain a <BALANCE_SIZE> sub-element. Presence of a <BALANCE_SIZE> element indicates that space allocation should be distributed across the volumes of the placement class in chunks of the indicated size. For example, if a balance size of one megabyte is specified for a placement class containing three volumes, VxFS allocates the first megabyte of space for a new or extending file on the first (lowest indexed) volume in the class, the second megabyte on the second volume, the third megabyte on the third volume, the fourth megabyte on the first volume, and so forth. Using the Units attribute in the <BALANCE_SIZE> XML tag, the balance size value may be specified in the following units:

bytes	Bytes
KB	Kilobytes
MB	Megabytes
GB	Gigabytes

The <BALANCE_SIZE> element distributes the allocation of database files across the volumes in a placement class. In principle, distributing the data in each file across multiple volumes distributes the I/O load across the volumes as well.

The CREATE statement in the following example specifies that files to which the rule applies should be created on the tier1 volume if space is available, and on one of the tier2 volumes if not. If space allocation on tier1 and tier2 volumes is not possible, file creation fails, even if space is available on tier3 volumes.

```
<CREATE>
  <ON>
    <DESTINATION>
      <CLASS>tier1</CLASS>
    </DESTINATION>
    <DESTINATION>
      <CLASS>tier2</CLASS>
      <BALANCE_SIZE Units="MB">1</BALANCE_SIZE>
    </DESTINATION>
  </ON>
</CREATE>
```

The <BALANCE_SIZE> element with a value of one megabyte is specified for allocations on tier2 volumes. For files allocated on tier2 volumes, the first megabyte would be allocated on the first volume, the second on the second volume, and so forth.

RELOCATE statement

The RELOCATE action statement of file placement policy rules specifies an action that VxFS takes on designated files during periodic scans of the file system, and the circumstances under which the actions should be taken. The fsppadm enforce command is used to scan all or part of a file system for files that should be relocated based on rules in the active placement policy at the time of the scan.

See the fsppadm(1M) manual page.

The fsppadm enforce command scans file systems in path name order. For each file, VxFS identifies the first applicable rule in the active placement policy, as determined by the rules' SELECT statements. If the file resides on a volume specified in the <FROM> clause of one of the rule's RELOCATE statements, and if the file meets the criteria for relocation specified in the statement's <WHEN> clause, the file is scheduled for relocation to a volume in the first placement class listed in the <TO> clause that has space available for the file. The scan that results from issuing the fsppadm enforce command runs to completion before any files are relocated.

The following XML snippet illustrates the general form of the RELOCATE statement:

```
<RELOCATE>
  <FROM>
```

```
      <SOURCE>
        <CLASS> placement_class_name </CLASS>
      </SOURCE>
      <SOURCE> additional_placement_class_specifications
      </SOURCE>
    </FROM>
    <TO>
      <DESTINATION>
        <CLASS> placement_class_name </CLASS>
        <BALANCE_SIZE Units="units_specifier">
          chunk_size
        </BALANCE_SIZE>
      </DESTINATION>
      <DESTINATION>
        additional_placement_class_specifications
      </DESTINATION>
    </TO>
    <WHEN> relocation_conditions </WHEN>
  </RELOCATE>
```

A RELOCATE statement contains the following clauses:

- <FROM> – An optional clause that contains a list of placement classes from whose volumes designated files should be relocated if the files meet the conditions specified in the <WHEN> clause. No priority is associated with the ordering of placement classes listed in a <FROM> clause. If a file to which the rule applies is located on a volume in any specified placement class, the file is considered for relocation.

  If a RELOCATE statement contains a <FROM> clause, VxFS only considers files that reside on volumes in placement classes specified in the clause for relocation. If no <FROM> clause is present, qualifying files are relocated regardless of where the files reside.

- <TO> – Indicates the placement classes to which qualifying files should be relocated. Unlike the source placement class list in a FROM clause, placement classes in a <TO>clause are specified in priority order. Files are relocated to volumes in the first specified placement class if possible, to the second if not, and so forth.

  The <TO> clause of the RELOCATE statement contains a list of <DESTINATION> XML elements specifying placement classes to whose volumes VxFS relocates qualifying files. Placement classes are specified in priority order. VxFS relocates qualifying files to volumes in the first placement class specified as long as space is available. A <DESTINATION> element may contain an optional <BALANCE_SIZE> modifier sub-element. The <BALANCE_SIZE> modifier indicates that relocated files should be distributed across the volumes of the destination placement class in chunks of the indicated size. For example, if a balance size of one megabyte is specified for a placement class containing three volumes, VxFS relocates the first megabyte the file to the first (lowest indexed) volume in the class, the second megabyte to the second volume, the third megabyte to the third volume, the fourth megabyte to the first volume, and so forth. Using the Units attribute in the `<BALANCE_SIZE>` XML tag, the chunk value may be specified in the balance size value may be specified in bytes (`Units="bytes"`), kilobytes (`Units="KB"`), megabytes (`Units="MB"`), or gigabytes (`Units="GB"`).

The `<BALANCE_SIZE>` element distributes the allocation of database files across the volumes in a placement class. In principle, distributing the data in each file across multiple volumes distributes the I/O load across the volumes as well.

For a multi-volume file system, you can specify the `compress` flag or the `uncompress` flag with the `<TO>` clause. The `compress` flag causes SmartTier to compress a file's extents while relocating the file to the tier specified by the `<DESTINATION>` element. SmartTier compresses the entire file and relocates the file to the destination tier, even if the file spans multiple tiers. The `uncompress` flag causes SmartTier to uncompress a file's extents while relocating the file to the tier specified by the `<DESTINATION>` element.

The following XML snippet specifies the `compress` flag:

```
<TO Flags="compress">
   <DESTINATION>
      <CLASS> tier4 </CLASS>
   </DESTINATION>
</TO>
```

The following XML snippet specifies the `uncompress` flag:

```
<TO Flags="uncompress">
   <DESTINATION>
      <CLASS> tier4 </CLASS>
   </DESTINATION>
</TO>
```

- `<WHEN>` – An optional clause that indicates the conditions under which files to which the rule applies should be relocated. Files that have been unaccessed or unmodified for a specified period, reached a certain size, or reached a specific I/O temperature or access temperature level may be relocated. If a `RELOCATE` statement does not contain a `<WHEN>` clause, files to which the rule applies are relocated unconditionally.

A <WHEN>clause may be included in a RELOCATE statement to specify that files should be relocated only if any or all of four types of criteria are met. Files can be specified for relocation if they satisfy one or more criteria.

The following are the criteria that can be specified for the <WHEN> clause:

<ACCAGE>

This criterion is met when files are inactive for a designated period or during a designated period relative to the time at which the `fsppadm enforce` command was issued.

<MODAGE>

This criterion is met when files are unmodified for a designated period or during a designated period relative to the time at which the `fsppadm enforce` command was issued.

<SIZE>

This criterion is met when files exceed or drop below a designated size or fall within a designated size range.

<IOTEMP>

This criterion is met when files exceed or drop below a designated I/O temperature, or fall within a designated I/O temperature range. A file's I/O temperature is a measure of the I/O activity against it during the period designated by the <PERIOD>element prior to the time at which the `fsppadm enforce` command was issued.

See “Calculating I/O temperature and access temperature” on page 578.

<ACCESSTEMP>

This criterion is met when files exceed or drop below a specified average access temperature, or fall within a specified access temperature range. A file's access temperature is similar to its I/O temperature, except that access temperature is computed using the number of I/O requests to the file, rather than the number of bytes transferred.

Note: The use of <IOTEMP> and <ACCESSTEMP> for data placement on VxFS servers that are used as NFS servers may not be very effective due to NFS caching. NFS client side caching and the way that NFS works can result in I/O initiated from an NFS client not producing NFS server side I/O. As such, any temperature measurements in place on the server side will not correctly reflect the I/O behavior that is specified by the placement policy.

If the server is solely used as an NFS server, this problem can potentially be mitigated by suitably adjusting or lowering the temperature thresholds. However, adjusting the thresholds may not always create the desired effect. In addition, if the same mount point is used both as an NFS export as well as a local mount, the temperature-based placement decisions will not be very effective due to the NFS cache skew.

The following XML snippet illustrates the general form of the <WHEN> clause in a RELOCATE statement:

```
<WHEN>
  <ACCAGE Units="units_value">
    <MIN Flags="comparison_operator">
      min_access_age</MIN>
    <MAX Flags="comparison_operator">
      max_access_age</MAX>
  </ACCAGE>
  <MODAGE Units="units_value">
    <MIN Flags="comparison_operator">
      min_modification_age</MIN>
    <MAX Flags="comparison_operator">
      max_modification_age</MAX>
  </MODAGE>
  <SIZE " Units="units_value">
    <MIN Flags="comparison_operator">
      min_size</MIN>
    <MAX Flags="comparison_operator">
      max_size</MAX>
  </SIZE>
  <IOTEMP Type="read_write_preference" Prefer="temperature_preference">
    <MIN Flags="comparison_operator">
      min_I/O_temperature</MIN>
    <MAX Flags="comparison_operator">
      max_I/O_temperature</MAX>
    <PERIOD Units="days_or_hours"> days_or_hours_of_interest </PERIOD>
  </IOTEMP>
  <ACCESSTEMP Type="read_write_preference"
   Prefer="temperature_preference">
    <MIN Flags="comparison_operator">
      min_access_temperature</MIN>
    <MAX Flags="comparison_operator">
      max_access_temperature</MAX>
    <PERIOD Units="days_or_hours"> days_or_hours_of_interest </PERIOD>
  </ACCESSTEMP>
</WHEN>
```

The access age (<ACCAGE>) element refers to the amount of time since a file was last accessed. VxFS computes access age by subtracting a file's time of last access, atime, from the time when the fsppadm enforce command was issued. The <MIN> and <MAX> XML elements in an <ACCAGE> clause, denote the minimum and maximum access age thresholds for relocation, respectively. These elements are optional, but at least one must be included. Using the `Units` XML attribute, the `<MIN>` and `<MAX>` elements may be specified in the following units:

`hours`	Hours
`days`	Days. A day is considered to be 24 hours prior to the time that the `fsppadm enforce` command was issued.

Both the `<MIN>` and `<MAX>` elements require `Flags` attributes to direct their operation.

For `<MIN>`, the following `Flags` attributes values may be specified:

`gt`	The time of last access must be greater than the specified interval.
`eq`	The time of last access must be equal to the specified interval.
`gteq`	The time of last access must be greater than or equal to the specified interval.

For `<MAX>`, the following Flags attributes values may be specified.

`lt`	The time of last access must be less than the specified interval.
`lteq`	The time of last access must be less than or equal to the specified interval.

Including a `<MIN>` element in a `<WHEN>` clause causes VxFS to relocate files to which the rule applies that have been inactive for longer than the specified interval. Such a rule would typically be used to relocate inactive files to less expensive storage tiers. Conversely, including `<MAX>` causes files accessed within the specified interval to be relocated. It would typically be used to move inactive files against which activity had recommenced to higher performance or more reliable storage. Including both `<MIN>` and `<MAX>` causes VxFS to relocate files whose access age lies between the two.

The modification age relocation criterion, `<MODAGE>`, is similar to access age, except that files' POSIX mtime values are used in computations. You would typically specify the `<MODAGE>` criterion to cause relocation of recently modified files to higher performance or more reliable storage tiers in anticipation that the files would be accessed recurrently in the near future.

The file size relocation criterion, `<SIZE>`, causes files to be relocated if the files are larger or smaller than the values specified in the `<MIN>` and `<MAX>` relocation criteria, respectively, at the time that the `fsppadm enforce` command was issued. Specifying both criteria causes VxFS to schedule relocation for files whose sizes lie between the two. Using the Units attribute, threshold file sizes may be specified in the following units:

bytes	Bytes
KB	Kilobytes
MB	Megabytes
GB	Gigabytes

Specifying the I/O temperature relocation criterion

The I/O temperature relocation criterion, <IOTEMP>, causes files to be relocated if their I/O temperatures rise above or drop below specified values over a specified period immediately prior to the time at which the fsppadm enforce command was issued. A file's I/O temperature is a measure of the read, write, or total I/O activity against it normalized to the file's size. Higher I/O temperatures indicate higher levels of application activity; lower temperatures indicate lower levels. VxFS computes a file's I/O temperature by dividing the number of bytes transferred to or from it (read, written, or both) during the specified period by its size at the time that the fsppadm enforce command was issued.

See "Calculating I/O temperature and access temperature" on page 578.

As with the other file relocation criteria, <IOTEMP> may be specified with a lower threshold by using the <MIN> element, an upper threshold by using the <MAX> element, or as a range by using both. However, I/O temperature is dimensionless and therefore has no specification for units.

VxFS computes files' I/O temperatures over the period between the time when the fsppadm enforce command was issued and the number of days or hours in the past specified in the <PERIOD> element, where a day is a 24 hour period. The default unit of time is days. You can specify hours as the time unit by setting the Units attribute of the <PERIOD> element to hours. Veritas recommends that you specify hours only if you are using solid state disks (SSDs).

See "Frequent SmartTier scans with solid state disks" on page 590.

For example, if you issued the fsppadm enforce command at 2 PM on Wednesday and you want VxFS to look at file I/O activity for the period between 2 PM on Monday and 2 PM on Wednesday, which is a period of 2 days, you would specify the following <PERIOD> element:

```
<PERIOD> 2 </PERIOD>
```

If you instead want VxFS to look at file I/O activity between 3 hours prior to running the fsppadm enforce command and the time that you ran the command, you specify the following <PERIOD> element:

```
<PERIOD Units="hours"> 3 </PERIOD>
```

The amount of time specified in the <PERIOD> element should not exceed one or two weeks due to the disk space used by the File Change Log (FCL) file.

See "About the Veritas File System File Change Log file" on page 767.

I/O temperature is a softer measure of I/O activity than access age. With access age, a single access to a file resets the file's atime to the current time. In contrast, a file's I/O temperature decreases gradually as time passes without the file being accessed, and increases gradually as the file is accessed periodically. For example, if a new 10 megabyte file is read completely five times on Monday and fsppadm enforce runs at midnight, the file's two-day I/O temperature will be five and its access age in days will be zero. If the file is read once on Tuesday, the file's access age in days at midnight will be zero, and its two-day I/O temperature will have dropped to three. If the file is read once on Wednesday, the file's access age at midnight will still be zero, but its two-day I/O temperature will have dropped to one, as the influence of Monday's I/O will have disappeared.

If the intention of a file placement policy is to keep files in place, such as on top-tier storage devices, as long as the files are being accessed at all, then access age is the more appropriate relocation criterion. However, if the intention is to relocate files as the I/O load on them decreases, then I/O temperature is more appropriate.

The case for upward relocation is similar. If files that have been relocated to lower-tier storage devices due to infrequent access experience renewed application activity, then it may be appropriate to relocate those files to top-tier devices. A policy rule that uses access age with a low <MAX> value, that is, the interval between fsppadm enforce runs, as a relocation criterion will cause files to be relocated that have been accessed even once during the interval. Conversely, a policy that uses I/O temperature with a <MIN> value will only relocate files that have experienced a sustained level of activity over the period of interest.

Prefer attribute

You can specify a value for the Prefer attribute for the <IOTEMP> and <ACCESSTEMP> criteria, which gives preference to relocating files. The Prefer attribute can take two values: low or high. If you specify low, Veritas File System (VxFS) relocates the files with the lower I/O temperature before relocating the files with the higher I/O temperature. If you specify high, VxFS relocates the files with the higher I/O temperature before relocating the files with the lower I/O temperature. Veritas recommends that you specify a `Prefer` attribute value only if you are using solid state disks (SSDs).

See "Prefer mechanism with solid state disks" on page 589.

Different `<PERIOD>` elements may be used in the `<IOTEMP>` and `<ACCESSTEMP>` criteria of different `RELOCATE` statements within the same policy.

The following placement policy snippet gives an example of the `Prefer` criteria:

```
<RELOCATE>
   ...
   <WHEN>
      <IOTEMP Type="nrbytes" Prefer="high">
         <MIN Flags="gteq"> 3.4 </MIN>
         <PERIOD Units="hours"> 6 </PERIOD>
      </IOTEMP>
   </WHEN>
</RELOCATE>
```

If there are a number of files whose I/O temperature is greater than the given minimum value, the files with the higher temperature are first subject to the `RELOCATE` operation before the files with the lower temperature.

Average I/O activity criteria

The `Average` criteria allows you to specify the value of the I/O temperature as a ratio of per-file activity that occurs over the time specified by the `<PERIOD>` element compared to the overall file system activity that occurs over a longer period of time. The `<PERIOD>` element in the `RELOCATE` criteria specifies the a number of hours or days immediately before the time of the scan. During that time, the I/O statistics that are collected are used to process the files that are being scanned. Since I/O activity can change over time, collect the average I/O activity over a longer duration than the `<PERIOD>` value itself, which is by default 24 hours. Doing so lets you compute an average temperature of the whole file system. Veritas recommends that you specify an `Average` attribute value only if you are using solid state disks (SSDs).

See "Average I/O activity with solid state disks" on page 590.

The following placement policy snippet gives an example of the `Average` criteria:

```
<RELOCATE>
   ...
   <WHEN>
      <IOTEMP Type="nrbytes" Prefer="high" Average="*">
         <MIN Flags="gteq"> 1.5 </MIN>
```

```
            <PERIOD Units="hours"> 6 </PERIOD>
         </IOTEMP>
      </WHEN>
   </RELOCATE>
```

In the snippet, VxFS relocates any file whose read IOTEMP over the last 6 hours is 1.5 times that of all the active files in the whole file system over the last 24 hours. This `Average` criteria is more intuitive and easier to specify than the absolute values.

The following formula computes the read IOTEMP of a given file:

```
IOTEMP = (bytes of the file that are read in the PERIOD) /
         (PERIOD in hours * size of the file in bytes)
```

The write and read/write IOTEMP are also computed accordingly.

The following formula computes the average read IOTEMP:

```
Average IOTEMP = (bytes read of all active files in the last h hours) /
                 (h * size of all the active files in bytes)
```

h is 24 hours by default. The average write and read/write IOTEMP are also computed accordingly.

In the example snippet, the value 1.5 is the multiple of average read IOTEMP over the last 24 hours across the whole file system, or rather across all of the active inodes whose activity is still available in the File Change Log (FCL) file at the time of the scan. Thus, the files' read IOTEMP activity over the last 6 hours is compared against 1.5 times that of the last 24 hours average activity to make the relocation decision. Using this method eliminates the need to give a specific number for the `<IOTEMP>` or `<ACCESSTEMP>` criteria, and instead lets you specify a multiple of the Average temperature. Keeping this averaging period longer than the specified `<PERIOD>` value normalizes the effects of any spikes and lulls in the file activity.

You can also use the `Average` criteria with the `<ACCESSTEMP>` criteria. The purpose and usage are the same.

You determine the type of the average by whether you specify the `Average` criteria with the `<IOTEMP>` or with the `<ACCESSTEMP>` criteria. The `Average` criteria can be any of the following types, depending on the criteria used:

- read Average IOTEMP
- write Average IOTEMP
- rw Average IOTEMP
- read Average ACCESSTEMP
- write Average ACCESSTEMP

- rw Average ACCESSTEMP

The default Average is a 24 hour average temperature, which is the total of all of the temperatures available up to the last 24 hours in the FCL file, divided by the number of files for which such I/O statistics still exist in the FCL file. You can override the number of hours by specifying the AveragePeriod attribute in the <PLACEMENT_POLICY> element. Veritas recommends that you specify an AveragePeriod attribute value only if you are using solid state disks (SSDs).

The following example statement causes the average file system activity be collected and computed over a period of 30 hours instead of the default 24 hours:

```
<PLACEMENT_POLICY Name="Policy1" Version="5.1" AveragePeriod="30">
```

RELOCATE statement examples

The following example illustrates an unconditional relocation statement, which is the simplest form of the RELOCATE policy rule statement:

```
<RELOCATE>
  <FROM>
    <SOURCE>
      <CLASS>tier1</CLASS>
    </SOURCE>
  </FROM>
  <TO>
    <DESTINATION>
      <CLASS>tier2</CLASS>
    </DESTINATION>
  </TO>
</RELOCATE>
```

The files designated by the rule's SELECT statement that reside on volumes in placement class tier1 at the time the fsppadm enforce command executes would be unconditionally relocated to volumes in placement class tier2 as long as space permitted. This type of rule might be used, for example, with applications that create and access new files but seldom access existing files once they have been processed. A CREATE statement would specify creation on tier1 volumes, which are presumably high performance or high availability, or both. Each instantiation of fsppadm enforce would relocate files created since the last run to tier2 volumes.

The following example illustrates a more comprehensive form of the RELOCATE statement that uses access age as the criterion for relocating files from tier1 volumes to tier2 volumes. This rule is designed to maintain free space on tier1 volumes by relocating inactive files to tier2 volumes:

```
<RELOCATE>
  <FROM>
    <SOURCE>
      <CLASS>tier1</CLASS>
    </SOURCE>
  </FROM>
  <TO>
    <DESTINATION>
      <CLASS>tier2</CLASS>
     </DESTINATION>
  </TO>
  <WHEN>
    <SIZE Units="MB">
      <MIN Flags="gt">1</MIN>
      <MAX Flags="lt">1000</MAX>
    </SIZE>
    <ACCAGE Units="days">
      <MIN Flags="gt">30</MIN>
    </ACCAGE>
  </WHEN>
</RELOCATE>
```

Files designated by the rule's `SELECT` statement are relocated from `tier1` volumes to `tier2` volumes if they are between 1 MB and 1000 MB in size and have not been accessed for 30 days. VxFS relocates qualifying files in the order in which it encounters them as it scans the file system's directory tree. VxFS stops scheduling qualifying files for relocation when when it calculates that already-scheduled relocations would result in `tier2` volumes being fully occupied.

The following example illustrates a possible companion rule that relocates files from `tier2` volumes to `tier1` ones based on their I/O temperatures. This rule might be used to return files that had been relocated to `tier2` volumes due to inactivity to `tier1` volumes when application activity against them increases. Using I/O temperature rather than access age as the relocation criterion reduces the chance of relocating files that are not actually being used frequently by applications. This rule does not cause files to be relocated unless there is sustained activity against them over the most recent two-day period.

```
<RELOCATE>
  <FROM>
    <SOURCE>
      <CLASS>tier2</CLASS>
    </SOURCE>
  </FROM>
```

```
   <TO>
     <DESTINATION>
       <CLASS>tier1</CLASS>
     </DESTINATION>
   </TO>
   <WHEN>
     <IOTEMP Type="nrbytes">
       <MIN Flags="gt">5</MIN>
       <PERIOD>2</PERIOD>
     </IOTEMP>
   </WHEN>
 </RELOCATE>
```

This rule relocates files that reside on `tier2` volumes to `tier1` volumes if their I/O temperatures are above 5 for the two day period immediately preceding the issuing of the `fsppadm enforce` command. VxFS relocates qualifying files in the order in which it encounters them during its file system directory tree scan. When `tier1` volumes are fully occupied, VxFS stops scheduling qualifying files for relocation.

VxFS file placement policies are able to control file placement across any number of placement classes. The following example illustrates a rule for relocating files with low I/O temperatures from `tier1` volumes to `tier2` volumes, and to `tier3` volumes when `tier2` volumes are fully occupied:

```
 <RELOCATE>
   <FROM>
     <SOURCE>
       <CLASS>tier1</CLASS>
     </SOURCE>
   </FROM>
   <TO>
     <DESTINATION>
       <CLASS>tier2</CLASS>
     </DESTINATION>
     <DESTINATION>
       <CLASS>tier3</CLASS>
     </DESTINATION>
   </TO>
   <WHEN>
     <IOTEMP Type="nrbytes">
       <MAX Flags="lt">4</MAX>
       <PERIOD>3</PERIOD>
     </IOTEMP>
```

```
  </WHEN>
</RELOCATE>
```

This rule relocates files whose 3-day I/O temperatures are less than 4 and which reside on `tier1` volumes. When VxFS calculates that already-relocated files would result in `tier2` volumes being fully occupied, VxFS relocates qualifying files to `tier3` volumes instead. VxFS relocates qualifying files as it encounters them in its scan of the file system directory tree.

The `<FROM>` clause in the `RELOCATE` statement is optional. If the clause is not present, VxFS evaluates files designated by the rule's `SELECT` statement for relocation no matter which volumes they reside on when the `fsppadm enforce` command is issued. The following example illustrates a fragment of a policy rule that relocates files according to their sizes, no matter where they reside when the `fsppadm enforce` command is issued:

```
<RELOCATE>
  <TO>
    <DESTINATION>
      <CLASS>tier1</CLASS>
    </DESTINATION>
  </TO>
  <WHEN>
    <SIZE Units="MB">
      <MAX Flags="lt">10</MAX>
    </SIZE>
  </WHEN>
</RELOCATE>
<RELOCATE>
  <TO>
    <DESTINATION>
      <CLASS>tier2</CLASS>
    </DESTINATION>
  </TO>
  <WHEN>
    <SIZE Units="MB">
      <MIN Flags="gteq">10</MIN>
      <MAX Flags="lt">100</MAX>
    </SIZE>
  </WHEN>
</RELOCATE>
<RELOCATE>
  <TO>
    <DESTINATION>
```

```
      <CLASS>tier3</CLASS>
    </DESTINATION>
  </TO>
  <WHEN>
    <SIZE Units="MB">
      <MIN Flags="gteq">100</MIN>
    </SIZE>
  </WHEN>
</RELOCATE>
```

This rule relocates files smaller than 10 megabytes to `tier1` volumes, files between 10 and 100 megabytes to `tier2` volumes, and files larger than 100 megabytes to `tier3` volumes. VxFS relocates all qualifying files that do not already reside on volumes in their `DESTINATION` placement classes when the `fsppadm enforce` command is issued.

The following example compresses while relocating all of the files from `tier2` with the extension `dbf` to `tier4` if the file was accessed over 30 days ago:

```
<SELECT Flags="Data">
  <PATTERN> *.dbf </PATTERN>
</SELECT>

<RELOCATE>
  <FROM>
    <SOURCE>
      <CLASS> tier2 </CLASS>
    </SOURCE>
  </FROM>
  <TO Flags="compress">
    <DESTINATION>
      <CLASS> tier4 </CLASS>
    </DESTINATION>
  </TO>
  <WHEN>
    <ACCAGE Units="days">
      <MIN Flags="gt">30</MIN>
    </ACCAGE>
  </WHEN>
</RELOCATE>
```

The following example uncompresses while relocating all of the files from `tier3` with the extension `dbf` to `tier1` if the file was accessed over 1 hour ago:

```
<SELECT Flags="Data">
  <PATTERN> *.dbf </PATTERN>
</SELECT>

<RELOCATE>
  <FROM>
    <SOURCE>
      <CLASS> tier3 </CLASS>
    </SOURCE>
  </FROM>
  <TO Flags="uncompress">
    <DESTINATION>
      <CLASS> tier1 </CLASS>
    </DESTINATION>
  </TO>
  <WHEN>
    <ACCAGE Units="hours">
      <MIN Flags="gt">1</MIN>
    </ACCAGE>
  </WHEN>
</RELOCATE>
```

DELETE statement

The DELETE file placement policy rule statement is very similar to the RELOCATE statement in both form and function, lacking only the <TO> clause. File placement policy-based deletion may be thought of as relocation with a fixed destination.

Note: Use DELETE statements with caution.

The following XML snippet illustrates the general form of the DELETE statement:

```
<DELETE>
  <FROM>
    <SOURCE>
      <CLASS> placement_class_name </CLASS>
    </SOURCE>
    <SOURCE>
      additional_placement_class_specifications
    </SOURCE>
  </FROM>
  <WHEN> relocation_conditions </WHEN>
</DELETE>
```

A DELETE statement contains the following clauses:

<FROM>

An optional clause that contains a list of placement classes from whose volumes designated files should be deleted if the files meet the conditions specified in the <WHEN> clause. No priority is associated with the ordering of placement classes in a <FROM> clause. If a file to which the rule applies is located on a volume in any specified placement class, the file is deleted. If a DELETE statement does not contain a <FROM> clause, VxFS deletes qualifying files no matter on which of a file system's volumes the files reside.

<WHEN>

An optional clause specifying the conditions under which files to which the rule applies should be deleted. The form of the <WHEN> clause in a DELETE statement is identical to that of the <WHEN> clause in a RELOCATE statement. If a DELETE statement does not contain a <WHEN> clause, files designated by the rule's SELECT statement, and the <FROM> clause if it is present, are deleted unconditionally.

DELETE statement examples

The following example illustrates the use of the DELETE statement:

```
<DELETE>
  <FROM>
    <SOURCE>
      <CLASS>tier3</CLASS>
    </SOURCE>
  </FROM>
</DELETE>
<DELETE>
  <FROM>
    <SOURCE>
      <CLASS>tier2</CLASS>
    </SOURCE>
  </FROM>
  <WHEN>
    <ACCAGE Units="days">
      <MIN Flags="gt">120</MIN>
    </ACCAGE>
  </WHEN>
</DELETE>
```

The first DELETE statement unconditionally deletes files designated by the rule's SELECT statement that reside on tier3 volumes when the fsppadm enforce command is issued. The absence of a `<WHEN>` clause in the `DELETE` statement indicates that deletion of designated files is unconditional.

The second `DELETE` statement deletes files to which the rule applies that reside on `tier2` volumes when the `fsppadm enforce` command is issued and that have not been accessed for the past 120 days.

COMPRESS statement

The `COMPRESS` statement in a file placement policy rule specifies in-place file compression on multi-volume or single-volume file systems. The placement policy becomes assigned to the selected file, and allocation for the compressed extents is done from the same tier specified in the `<SOURCE>` element of the `<FROM>` clause. SmartTier performs in-place compression of the entire file, even if the file spans across multiple tiers.

Note: SmartTier does not schedule compression activity. If you did not integrate your Veritas InfoScale product with the Veritas Operations Manager (VOM), then you must automate compression activity by using techniques such as scheduling through cron jobs.

The following XML snippet illustrates the general form of the `COMPRESS` statement:

```
<COMPRESS>
  <FROM>
    <SOURCE>
      <CLASS> placement_class_name </CLASS>
    </SOURCE>
    <SOURCE> additional_placement_class_specifications
    </SOURCE>
  </FROM>
  <WHEN> compression_conditions </WHEN>
</COMPRESS>
```

A `COMPRESS` statement contains the following clauses:

`<FROM>`

An optional clause that contains a list of placement classes from whose volumes designated files should be compressed if the files meet the conditions specified in the `<WHEN>` clause. No priority is associated with the ordering of placement classes listed in a `<FROM>` clause. If a file to which the rule applies is located on a volume in any specified placement class, the file is considered for compression.

If a `COMPRESS` statement contains a `<FROM>` clause, VxFS only considers files that reside on volumes in placement classes specified in the clause for compression. If no `<FROM>` clause is present, qualifying files are compressed regardless of where the files reside.

`<WHEN>`

An optional clause that indicates the conditions under which files to which the rule applies should be compressed. Files that have been unaccessed or unmodified for a specified period, reached a certain size, or reached a specific I/O temperature or access temperature level may be compressed. If a `COMPRESS` statement does not contain a `<WHEN>` clause, files to which the rule applies are compressed unconditionally.

A `<WHEN>` clause may be included in a `COMPRESS` statement to specify that files should be compressed only if any or all of four types of criteria are met. Files can be specified for compression if they satisfy one or more criteria.

The following are the criteria that can be specified for the `<WHEN>` clause:

`<ACCAGE>`

This criterion is met when files are inactive for a designated period or during a designated period relative to the time at which the `fsppadm enforce` command was issued.

`<MODAGE>`

This criterion is met when files are unmodified for a designated period or during a designated period relative to the time at which the `fsppadm enforce` command was issued.

`<SIZE>`

This criterion is met when files exceed or drop below a designated size or fall within a designated size range.

`<IOTEMP>`

This criterion is met when files exceed or drop below a designated I/O temperature, or fall within a designated I/O temperature range. A file's I/O temperature is a measure of the I/O activity against it during the period designated by the `<PERIOD>`element prior to the time at which the `fsppadm enforce` command was issued.

See “Calculating I/O temperature and access temperature” on page 578.

`<ACCESSTEMP>` This criterion is met when files exceed or drop below a specified average access temperature, or fall within a specified access temperature range. A file's access temperature is similar to its I/O temperature, except that access temperature is computed using the number of I/O requests to the file, rather than the number of bytes transferred.

Note: The use of `<IOTEMP>` and `<ACCESSTEMP>` for data placement on VxFS servers that are used as NFS servers may not be very effective due to NFS caching. NFS client side caching and the way that NFS works can result in I/O initiated from an NFS client not producing NFS server side I/O. As such, any temperature measurements in place on the server side will not correctly reflect the I/O behavior that is specified by the placement policy.

If the server is solely used as an NFS server, this problem can potentially be mitigated by suitably adjusting or lowering the temperature thresholds. However, adjusting the thresholds may not always create the desired effect. In addition, if the same mount point is used both as an NFS export as well as a local mount, the temperature-based placement decisions will not be very effective due to the NFS cache skew.

The following XML snippet illustrates the general form of the `<WHEN>` clause in a `COMPRESS` statement:

```
<WHEN>
  <ACCAGE Units="units_value">
    <MIN Flags="comparison_operator">
      min_access_age</MIN>
    <MAX Flags="comparison_operator">
      max_access_age</MAX>
  </ACCAGE>
  <MODAGE Units="units_value">
    <MIN Flags="comparison_operator">
      min_modification_age</MIN>
    <MAX Flags="comparison_operator">
      max_modification_age</MAX>
  </MODAGE>
  <SIZE " Units="units_value">
    <MIN Flags="comparison_operator">
      min_size</MIN>
    <MAX Flags="comparison_operator">
      max_size</MAX>
  </SIZE>
  <IOTEMP Type="read_write_preference" Prefer="temperature_preference">
    <MIN Flags="comparison_operator">
      min_I/O_temperature</MIN>
    <MAX Flags="comparison_operator">
      max_I/O_temperature</MAX>
    <PERIOD Units="days_or_hours"> days_or_hours_of_interest </PERIOD>
  </IOTEMP>
  <ACCESSTEMP Type="read_write_preference"
   Prefer="temperature_preference">
    <MIN Flags="comparison_operator">
      min_access_temperature</MIN>
    <MAX Flags="comparison_operator">
      max_access_temperature</MAX>
    <PERIOD Units="days_or_hours"> days_or_hours_of_interest </PERIOD>
  </ACCESSTEMP>
</WHEN>
```

The access age (`<ACCAGE>`) element refers to the amount of time since a file was last accessed. VxFS computes access age by subtracting a file's time of last access, atime, from the time when the `fsppadm enforce` command was issued. The `<MIN>` and `<MAX>` XML elements in an `<ACCAGE>` clause, denote the minimum and maximum access age thresholds for compression, respectively. These elements are optional, but at least one must be included. Using the `Units` XML attribute, the `<MIN>` and `<MAX>` elements may be specified in the following units:

`hours` Hours

`days` Days. A day is considered to be 24 hours prior to the time that the `fsppadm enforce` command was issued.

Both the `<MIN>` and `<MAX>` elements require `Flags` attributes to direct their operation.

For `<MIN>`, the following `Flags` attributes values may be specified:

`gt` The time of last access must be greater than the specified interval.

`eq` The time of last access must be equal to the specified interval.

`gteq` The time of last access must be greater than or equal to the specified interval.

For `<MAX>`, the following Flags attributes values may be specified.

`lt` The time of last access must be less than the specified interval.

lteq	The time of last access must be less than or equal to the specified interval.

Including a <MIN> element in a <WHEN> clause causes VxFS to compress files to which the rule applies that have been inactive for longer than the specified interval. Such a rule would typically be used to compress inactive files to less expensive storage tiers. Conversely, including <MAX> causes files accessed within the specified interval to be compressed. It would typically be used to move inactive files against which activity had recommenced to higher performance or more reliable storage. Including both <MIN> and <MAX> causes VxFS to compress files whose access age lies between the two.

The modification age compression criterion, <MODAGE>, is similar to access age, except that files' POSIX mtime values are used in computations. You would typically specify the <MODAGE> criterion to cause compression of recently modified files to higher performance or more reliable storage tiers in anticipation that the files would be accessed recurrently in the near future.

The file size compression criterion, <SIZE>, causes files to be compressed if the files are larger or smaller than the values specified in the <MIN> and <MAX> compression criteria, respectively, at the time that the `fsppadm enforce` command was issued. Specifying both criteria causes VxFS to schedule compression for files whose sizes lie between the two. Using the Units attribute, threshold file sizes may be specified in the following units:

bytes	Bytes
KB	Kilobytes
MB	Megabytes
GB	Gigabytes

Specifying the I/O temperature compression criterion

The I/O temperature compression criterion, <IOTEMP>, causes files to be compressed if their I/O temperatures rise above or drop below specified values over a specified period immediately prior to the time at which the `fsppadm enforce` command was issued. A file's I/O temperature is a measure of the read, write, or total I/O activity against it normalized to the file's size. Higher I/O temperatures indicate higher levels of application activity; lower temperatures indicate lower levels. VxFS computes a file's I/O temperature by dividing the number of bytes transferred to or from it (read, written, or both) during the specified period by its size at the time that the `fsppadm enforce` command was issued.

See "Calculating I/O temperature and access temperature" on page 578.

As with the other file compression criteria, <IOTEMP> may be specified with a lower threshold by using the <MIN> element, an upper threshold by using the <MAX> element, or as a range by using both. However, I/O temperature is dimensionless and therefore has no specification for units.

VxFS computes files' I/O temperatures over the period between the time when the fsppadm enforce command was issued and the number of days or hours in the past specified in the <PERIOD> element, where a day is a 24 hour period. The default unit of time is days. You can specify hours as the time unit by setting the Units attribute of the <PERIOD> element to hours. Veritas recommends that you specify hours only if you are using solid state disks (SSDs).

See "Frequent SmartTier scans with solid state disks" on page 590.

For example, if you issued the fsppadm enforce command at 2 PM on Wednesday and you want VxFS to look at file I/O activity for the period between 2 PM on Monday and 2 PM on Wednesday, which is a period of 2 days, you would specify the following <PERIOD> element:

```
<PERIOD> 2 </PERIOD>
```

If you instead want VxFS to look at file I/O activity between 3 hours prior to running the fsppadm enforce command and the time that you ran the command, you specify the following <PERIOD> element:

```
<PERIOD Units="hours"> 3 </PERIOD>
```

The amount of time specified in the <PERIOD> element should not exceed one or two weeks due to the disk space used by the File Change Log (FCL) file.

See "About the Veritas File System File Change Log file" on page 767.

I/O temperature is a softer measure of I/O activity than access age. With access age, a single access to a file resets the file's atime to the current time. In contrast, a file's I/O temperature decreases gradually as time passes without the file being accessed, and increases gradually as the file is accessed periodically. For example, if a new 10 megabyte file is read completely five times on Monday and fsppadm enforce runs at midnight, the file's two-day I/O temperature will be five and its access age in days will be zero. If the file is read once on Tuesday, the file's access age in days at midnight will be zero, and its two-day I/O temperature will have dropped to three. If the file is read once on Wednesday, the file's access age at midnight will still be zero, but its two-day I/O temperature will have dropped to one, as the influence of Monday's I/O will have disappeared.

If the intention of a file placement policy is to keep files in place, such as on top-tier storage devices, as long as the files are being accessed at all, then access age is the more appropriate compression criterion. However, if the intention is to compress files as the I/O load on them decreases, then I/O temperature is more appropriate.

The case for upward compression is similar. If files that have been compressed to lower-tier storage devices due to infrequent access experience renewed application activity, then it may be appropriate to compress those files to top-tier devices. A policy rule that uses access age with a low <MAX> value, that is, the interval between fsppadm enforce runs, as a compression criterion will cause files to be compressed that have been accessed even once during the interval. Conversely, a policy that uses I/O temperature with a <MIN> value will only compress files that have experienced a sustained level of activity over the period of interest.

Prefer attribute

You can specify a value for the Prefer attribute for the <IOTEMP> and <ACCESSTEMP> criteria, which gives preference to compressing files. The Prefer attribute can take two values: low or high. If you specify low, Veritas File System (VxFS) compresss the files with the lower I/O temperature before compressing the files with the higher I/O temperature. If you specify high, VxFS compresss the files with the higher I/O temperature before compressing the files with the lower I/O temperature. Veritas recommends that you specify a Prefer attribute value only if you are using solid state disks (SSDs).

See "Prefer mechanism with solid state disks" on page 589.

Different <PERIOD> elements may be used in the <IOTEMP> and <ACCESSTEMP> criteria of different COMPRESS statements within the same policy.

The following placement policy snippet gives an example of the Prefer criteria:

```
<COMPRESS>
   ...
   <WHEN>
      <IOTEMP Type="nrbytes" Prefer="high">
         <MIN Flags="gteq"> 3.4 </MIN>
         <PERIOD Units="hours"> 6 </PERIOD>
      </IOTEMP>
   </WHEN>
</COMPRESS>
```

If there are a number of files whose I/O temperature is greater than the given minimum value, the files with the higher temperature are first subject to the COMPRESS operation before the files with the lower temperature.

Average I/O activity criteria

The `Average` criteria allows you to specify the value of the I/O temperature as a ratio of per-file activity that occurs over the time specified by the `<PERIOD>` element compared to the overall file system activity that occurs over a longer period of time. The `<PERIOD>` element in the `COMPRESS` criteria specifies the a number of hours or days immediately before the time of the scan. During that time, the I/O statistics that are collected are used to process the files that are being scanned. Since I/O activity can change over time, collect the average I/O activity over a longer duration than the `<PERIOD>` value itself, which is by default 24 hours. Doing so lets you compute an average temperature of the whole file system. Veritas recommends that you specify an `Average` attribute value only if you are using solid state disks (SSDs).

See "Average I/O activity with solid state disks" on page 590.

The following placement policy snippet gives an example of the `Average` criteria:

```
<COMPRESS>
   ...
   <WHEN>
      <IOTEMP Type="nrbytes" Prefer="high" Average="*">
         <MIN Flags="gteq"> 1.5 </MIN>
         <PERIOD Units="hours"> 6 </PERIOD>
      </IOTEMP>
   </WHEN>
</COMPRESS>
```

In the snippet, VxFS compresss any file whose read IOTEMP over the last 6 hours is 1.5 times that of all the active files in the whole file system over the last 24 hours. This `Average` criteria is more intuitive and easier to specify than the absolute values.

The following formula computes the read IOTEMP of a given file:

```
IOTEMP = (bytes of the file that are read in the PERIOD) /
         (PERIOD in hours * size of the file in bytes)
```

The write and read/write IOTEMP are also computed accordingly.

The following formula computes the average read IOTEMP:

```
Average IOTEMP = (bytes read of all active files in the last h hours) /
                 (h * size of all the active files in bytes)
```

h is 24 hours by default. The average write and read/write IOTEMP are also computed accordingly.

In the example snippet, the value 1.5 is the multiple of average read IOTEMP over the last 24 hours across the whole file system, or rather across all of the active inodes whose activity is still available in the File Change Log (FCL) file at the time of the scan. Thus, the files' read IOTEMP activity over the last 6 hours is compared against 1.5 times that of the last 24 hours average activity to make the compression decision. Using this method eliminates the need to give a specific number for the `<IOTEMP>` or `<ACCESSTEMP>` criteria, and instead lets you specify a multiple of the Average temperature. Keeping this averaging period longer than the specified `<PERIOD>` value normalizes the effects of any spikes and lulls in the file activity.

You can also use the `Average` criteria with the `<ACCESSTEMP>` criteria. The purpose and usage are the same.

You determine the type of the average by whether you specify the `Average` criteria with the `<IOTEMP>` or with the `<ACCESSTEMP>` criteria. The `Average` criteria can be any of the following types, depending on the criteria used:

- read Average IOTEMP
- write Average IOTEMP
- rw Average IOTEMP
- read Average ACCESSTEMP
- write Average ACCESSTEMP
- rw Average ACCESSTEMP

The default `Average` is a 24 hour average temperature, which is the total of all of the temperatures available up to the last 24 hours in the FCL file, divided by the number of files for which such I/O statistics still exist in the FCL file. You can override the number of hours by specifying the `AveragePeriod` attribute in the `<PLACEMENT_POLICY>` element. Veritas recommends that you specify an `AveragePeriod` attribute value only if you are using solid state disks (SSDs).

The following example statement causes the average file system activity be collected and computed over a period of 30 hours instead of the default 24 hours:

```
<PLACEMENT_POLICY Name="Policy1" Version="5.1" AveragePeriod="30">
```

COMPRESS statement examples

The following example compresses all of the files with the extension `dbf` on the multi-volume file system `tier2` that have not been accessed for last 30 days:

```
  <SELECT Flags="Data">
    <PATTERN> *.dbf </PATTERN>
  </SELECT>
```

```
<COMPRESS>
  <FROM>
    <SOURCE>
      <CLASS> tier2 </CLASS>
    </SOURCE>
  </FROM>
  <WHEN>
    <ACCAGE Units="days">
      <MIN Flags="gt">30</MIN>
    </ACCAGE>
  </WHEN>
</COMPRESS>
```

The files designated by the rule's `SELECT` statement that reside on volumes in placement class `tier2` at the time the `fsppadm enforce` command executes are compressed in place. Each instantiation of `fsppadm enforce` compresses files created since the last run on the `tier2` volumes.

The following example compresses all of the files with the extension `dbf` on a single volume if the file was not accessed for one minute.

```
<SELECT Flags="Data">
  <PATTERN> *.dbf </PATTERN>
</SELECT>

<COMPRESS>
  <WHEN>
    <ACCAGE Units="minutes">
      <MIN Flags="gt">1</MIN>
    </ACCAGE>
  </WHEN>
</COMPRESS>
```

No `<FROM>` clause is required for single volume. The files designated by the rule's `SELECT` statement at the time the `fsppadm enforce` command executes are compressed in place. Each instantiation of `fsppadm enforce` compresses files created since the last run on the volume.

The following example compresses all of the files on `tier3`:

```
<SELECT Flags="Data">
  <PATTERN> * </PATTERN>
</SELECT>
```

```
<COMPRESS>
  <FROM>
    <SOURCE>
      <CLASS> tier3 </CLASS>
    </SOURCE>
  </FROM>
</COMPRESS>
```

This rule compresses in place all files that reside on `tier3` at the time the `fsppadm enforce` command executes.

UNCOMPRESS statement

The `UNCOMPRESS` statement in a file placement policy rule specifies in-place file uncompression on multi-volume and single-volume file systems. The placement policy becomes assigned to the selected file, and allocation for the uncompressed extents is done from the tier specified in the `<SOURCE>` element of the `<FROM>` clause.

If a file is partially compressed, then the file can be picked only for in-place compression. After being compressed, the file will be uncompressed before being relocated in the next policy enforcement.

Note: SmartTier does not schedule uncompression activity. If you did not integrate your Veritas InfoScale product with the Veritas Operations Manager (VOM), then you must automate uncompression activity by using techniques such as scheduling through cron jobs.

The following XML snippet illustrates the general form of the `UNCOMPRESS` statement:

```
<UNCOMPRESS>
  <FROM>
    <SOURCE>
      <CLASS> placement_class_name </CLASS>
    </SOURCE>
    <SOURCE> additional_placement_class_specifications
    </SOURCE>
  </FROM>
  <WHEN> uncompression_conditions </WHEN>
</UNCOMPRESS>
```

A `UNCOMPRESS` statement contains the following clauses:

<FROM>

An optional clause that contains a list of placement classes from whose volumes designated files should be uncompressed if the files meet the conditions specified in the <WHEN> clause. No priority is associated with the ordering of placement classes listed in a <FROM> clause. If a file to which the rule applies is located on a volume in any specified placement class, the file is considered for uncompression.

If a UNCOMPRESS statement contains a <FROM> clause, VxFS only considers files that reside on volumes in placement classes specified in the clause for uncompression. If no <FROM> clause is present, qualifying files are uncompressed regardless of where the files reside.

<WHEN>

An optional clause that indicates the conditions under which files to which the rule applies should be uncompressed. Files that have been unaccessed or unmodified for a specified period, reached a certain size, or reached a specific I/O temperature or access temperature level may be uncompressed. If a UNCOMPRESS statement does not contain a <WHEN> clause, files to which the rule applies are uncompressed unconditionally.

A <WHEN> clause may be included in a UNCOMPRESS statement to specify that files should be uncompressed only if any or all of four types of criteria are met. Files can be specified for uncompression if they satisfy one or more criteria.

The following are the criteria that can be specified for the <WHEN> clause:

<ACCAGE>

This criterion is met when files are inactive for a designated period or during a designated period relative to the time at which the `fsppadm enforce` command was issued.

<MODAGE>

This criterion is met when files are unmodified for a designated period or during a designated period relative to the time at which the `fsppadm enforce` command was issued.

<SIZE>

This criterion is met when files exceed or drop below a designated size or fall within a designated size range.

<IOTEMP>

This criterion is met when files exceed or drop below a designated I/O temperature, or fall within a designated I/O temperature range. A file's I/O temperature is a measure of the I/O activity against it during the period designated by the <PERIOD>element prior to the time at which the `fsppadm enforce` command was issued.

See "Calculating I/O temperature and access temperature" on page 578.

`<ACCESSTEMP>`	This criterion is met when files exceed or drop below a specified average access temperature, or fall within a specified access temperature range. A file's access temperature is similar to its I/O temperature, except that access temperature is computed using the number of I/O requests to the file, rather than the number of bytes transferred.

Note: The use of `<IOTEMP>` and `<ACCESSTEMP>` for data placement on VxFS servers that are used as NFS servers may not be very effective due to NFS caching. NFS client side caching and the way that NFS works can result in I/O initiated from an NFS client not producing NFS server side I/O. As such, any temperature measurements in place on the server side will not correctly reflect the I/O behavior that is specified by the placement policy.

If the server is solely used as an NFS server, this problem can potentially be mitigated by suitably adjusting or lowering the temperature thresholds. However, adjusting the thresholds may not always create the desired effect. In addition, if the same mount point is used both as an NFS export as well as a local mount, the temperature-based placement decisions will not be very effective due to the NFS cache skew.

The following XML snippet illustrates the general form of the `<WHEN>` clause in a `UNCOMPRESS` statement:

```
<WHEN>
  <ACCAGE Units="units_value">
    <MIN Flags="comparison_operator">
      min_access_age</MIN>
    <MAX Flags="comparison_operator">
      max_access_age</MAX>
  </ACCAGE>
  <MODAGE Units="units_value">
    <MIN Flags="comparison_operator">
      min_modification_age</MIN>
    <MAX Flags="comparison_operator">
      max_modification_age</MAX>
  </MODAGE>
  <SIZE " Units="units_value">
    <MIN Flags="comparison_operator">
      min_size</MIN>
    <MAX Flags="comparison_operator">
      max_size</MAX>
  </SIZE>
  <IOTEMP Type="read_write_preference" Prefer="temperature_preference">
    <MIN Flags="comparison_operator">
      min_I/O_temperature</MIN>
    <MAX Flags="comparison_operator">
      max_I/O_temperature</MAX>
    <PERIOD Units="days_or_hours"> days_or_hours_of_interest </PERIOD>
  </IOTEMP>
  <ACCESSTEMP Type="read_write_preference"
   Prefer="temperature_preference">
    <MIN Flags="comparison_operator">
      min_access_temperature</MIN>
    <MAX Flags="comparison_operator">
      max_access_temperature</MAX>
    <PERIOD Units="days_or_hours"> days_or_hours_of_interest </PERIOD>
  </ACCESSTEMP>
</WHEN>
```

The access age (`<ACCAGE>`) element refers to the amount of time since a file was last accessed. VxFS computes access age by subtracting a file's time of last access, atime, from the time when the `fsppadm enforce` command was issued. The `<MIN>` and `<MAX>` XML elements in an `<ACCAGE>` clause, denote the minimum and maximum access age thresholds for uncompression, respectively. These elements are optional, but at least one must be included. Using the `Units` XML attribute, the `<MIN>` and `<MAX>` elements may be specified in the following units:

`hours` Hours

`days` Days. A day is considered to be 24 hours prior to the time that the `fsppadm enforce` command was issued.

Both the `<MIN>` and `<MAX>` elements require `Flags` attributes to direct their operation.

For `<MIN>`, the following `Flags` attributes values may be specified:

`gt` The time of last access must be greater than the specified interval.

`eq` The time of last access must be equal to the specified interval.

`gteq` The time of last access must be greater than or equal to the specified interval.

For `<MAX>`, the following Flags attributes values may be specified.

`lt` The time of last access must be less than the specified interval.

lteq	The time of last access must be less than or equal to the specified interval.

Including a <MIN> element in a <WHEN> clause causes VxFS to uncompress files to which the rule applies that have been inactive for longer than the specified interval. Such a rule would typically be used to uncompress inactive files to less expensive storage tiers. Conversely, including <MAX> causes files accessed within the specified interval to be uncompressed. It would typically be used to move inactive files against which activity had recommenced to higher performance or more reliable storage. Including both <MIN> and <MAX> causes VxFS to uncompress files whose access age lies between the two.

The modification age uncompression criterion, <MODAGE>, is similar to access age, except that files' POSIX mtime values are used in computations. You would typically specify the <MODAGE> criterion to cause uncompression of recently modified files to higher performance or more reliable storage tiers in anticipation that the files would be accessed recurrently in the near future.

The file size uncompression criterion, <SIZE>, causes files to be uncompressed if the files are larger or smaller than the values specified in the <MIN> and <MAX> uncompression criteria, respectively, at the time that the fsppadm enforce command was issued. Specifying both criteria causes VxFS to schedule uncompression for files whose sizes lie between the two. Using the Units attribute, threshold file sizes may be specified in the following units:

bytes	Bytes
KB	Kilobytes
MB	Megabytes
GB	Gigabytes

Specifying the I/O temperature uncompression criterion

The I/O temperature uncompression criterion, <IOTEMP>, causes files to be uncompressed if their I/O temperatures rise above or drop below specified values over a specified period immediately prior to the time at which the fsppadm enforce command was issued. A file's I/O temperature is a measure of the read, write, or total I/O activity against it normalized to the file's size. Higher I/O temperatures indicate higher levels of application activity; lower temperatures indicate lower levels. VxFS computes a file's I/O temperature by dividing the number of bytes transferred to or from it (read, written, or both) during the specified period by its size at the time that the fsppadm enforce command was issued.

See “Calculating I/O temperature and access temperature” on page 578.

As with the other file uncompression criteria, <IOTEMP> may be specified with a lower threshold by using the <MIN> element, an upper threshold by using the <MAX> element, or as a range by using both. However, I/O temperature is dimensionless and therefore has no specification for units.

VxFS computes files' I/O temperatures over the period between the time when the fsppadm enforce command was issued and the number of days or hours in the past specified in the <PERIOD> element, where a day is a 24 hour period. The default unit of time is days. You can specify hours as the time unit by setting the Units attribute of the <PERIOD> element to hours. Veritas recommends that you specify hours only if you are using solid state disks (SSDs).

See “Frequent SmartTier scans with solid state disks” on page 590.

For example, if you issued the fsppadm enforce command at 2 PM on Wednesday and you want VxFS to look at file I/O activity for the period between 2 PM on Monday and 2 PM on Wednesday, which is a period of 2 days, you would specify the following <PERIOD> element:

```
<PERIOD> 2 </PERIOD>
```

If you instead want VxFS to look at file I/O activity between 3 hours prior to running the fsppadm enforce command and the time that you ran the command, you specify the following <PERIOD> element:

```
<PERIOD Units="hours"> 3 </PERIOD>
```

The amount of time specified in the <PERIOD> element should not exceed one or two weeks due to the disk space used by the File Change Log (FCL) file.

See “About the Veritas File System File Change Log file” on page 767.

I/O temperature is a softer measure of I/O activity than access age. With access age, a single access to a file resets the file's atime to the current time. In contrast, a file's I/O temperature decreases gradually as time passes without the file being accessed, and increases gradually as the file is accessed periodically. For example, if a new 10 megabyte file is read completely five times on Monday and fsppadm enforce runs at midnight, the file's two-day I/O temperature will be five and its access age in days will be zero. If the file is read once on Tuesday, the file's access age in days at midnight will be zero, and its two-day I/O temperature will have dropped to three. If the file is read once on Wednesday, the file's access age at midnight will still be zero, but its two-day I/O temperature will have dropped to one, as the influence of Monday's I/O will have disappeared.

If the intention of a file placement policy is to keep files in place, such as on top-tier storage devices, as long as the files are being accessed at all, then access age is the more appropriate uncompression criterion. However, if the intention is to uncompress files as the I/O load on them decreases, then I/O temperature is more appropriate.

The case for upward uncompression is similar. If files that have been uncompressed to lower-tier storage devices due to infrequent access experience renewed application activity, then it may be appropriate to uncompress those files to top-tier devices. A policy rule that uses access age with a low <MAX> value, that is, the interval between fsppadm enforce runs, as a uncompression criterion will cause files to be uncompressed that have been accessed even once during the interval. Conversely, a policy that uses I/O temperature with a <MIN> value will only uncompress files that have experienced a sustained level of activity over the period of interest.

Prefer attribute

You can specify a value for the Prefer attribute for the <IOTEMP> and <ACCESSTEMP> criteria, which gives preference to uncompressing files. The Prefer attribute can take two values: low or high. If you specify low, Veritas File System (VxFS) uncompresss the files with the lower I/O temperature before uncompressing the files with the higher I/O temperature. If you specify high, VxFS uncompresss the files with the higher I/O temperature before uncompressing the files with the lower I/O temperature. Veritas recommends that you specify a Prefer attribute value only if you are using solid state disks (SSDs).

See “Prefer mechanism with solid state disks” on page 589.

Different <PERIOD> elements may be used in the <IOTEMP> and <ACCESSTEMP> criteria of different UNCOMPRESS statements within the same policy.

The following placement policy snippet gives an example of the Prefer criteria:

```
<UNCOMPRESS>
   ...
   <WHEN>
      <IOTEMP Type="nrbytes" Prefer="high">
         <MIN Flags="gteq"> 3.4 </MIN>
         <PERIOD Units="hours"> 6 </PERIOD>
      </IOTEMP>
   </WHEN>
</UNCOMPRESS>
```

If there are a number of files whose I/O temperature is greater than the given minimum value, the files with the higher temperature are first subject to the UNCOMPRESS operation before the files with the lower temperature.

Average I/O activity criteria

The `Average` criteria allows you to specify the value of the I/O temperature as a ratio of per-file activity that occurs over the time specified by the `<PERIOD>` element compared to the overall file system activity that occurs over a longer period of time. The `<PERIOD>` element in the `UNCOMPRESS` criteria specifies the a number of hours or days immediately before the time of the scan. During that time, the I/O statistics that are collected are used to process the files that are being scanned. Since I/O activity can change over time, collect the average I/O activity over a longer duration than the `<PERIOD>` value itself, which is by default 24 hours. Doing so lets you compute an average temperature of the whole file system. Veritas recommends that you specify an `Average` attribute value only if you are using solid state disks (SSDs).

See “Average I/O activity with solid state disks” on page 590.

The following placement policy snippet gives an example of the `Average` criteria:

```
<UNCOMPRESS>
   ...
   <WHEN>
      <IOTEMP Type="nrbytes" Prefer="high" Average="*">
         <MIN Flags="gteq"> 1.5 </MIN>
         <PERIOD Units="hours"> 6 </PERIOD>
      </IOTEMP>
   </WHEN>
</UNCOMPRESS>
```

In the snippet, VxFS uncompresss any file whose read IOTEMP over the last 6 hours is 1.5 times that of all the active files in the whole file system over the last 24 hours. This `Average` criteria is more intuitive and easier to specify than the absolute values.

The following formula computes the read IOTEMP of a given file:

```
IOTEMP = (bytes of the file that are read in the PERIOD) /
         (PERIOD in hours * size of the file in bytes)
```

The write and read/write IOTEMP are also computed accordingly.

The following formula computes the average read IOTEMP:

```
Average IOTEMP = (bytes read of all active files in the last h hours) /
                 (h * size of all the active files in bytes)
```

h is 24 hours by default. The average write and read/write IOTEMP are also computed accordingly.

In the example snippet, the value 1.5 is the multiple of average read IOTEMP over the last 24 hours across the whole file system, or rather across all of the active inodes whose activity is still available in the File Change Log (FCL) file at the time of the scan. Thus, the files' read IOTEMP activity over the last 6 hours is compared against 1.5 times that of the last 24 hours average activity to make the uncompression decision. Using this method eliminates the need to give a specific number for the `<IOTEMP>` or `<ACCESSTEMP>` criteria, and instead lets you specify a multiple of the Average temperature. Keeping this averaging period longer than the specified `<PERIOD>` value normalizes the effects of any spikes and lulls in the file activity.

You can also use the `Average` criteria with the `<ACCESSTEMP>` criteria. The purpose and usage are the same.

You determine the type of the average by whether you specify the `Average` criteria with the `<IOTEMP>` or with the `<ACCESSTEMP>` criteria. The `Average` criteria can be any of the following types, depending on the criteria used:

- read Average IOTEMP
- write Average IOTEMP
- rw Average IOTEMP
- read Average ACCESSTEMP
- write Average ACCESSTEMP
- rw Average ACCESSTEMP

The default `Average` is a 24 hour average temperature, which is the total of all of the temperatures available up to the last 24 hours in the FCL file, divided by the number of files for which such I/O statistics still exist in the FCL file. You can override the number of hours by specifying the `AveragePeriod` attribute in the `<PLACEMENT_POLICY>` element. Veritas recommends that you specify an `AveragePeriod` attribute value only if you are using solid state disks (SSDs).

The following example statement causes the average file system activity be collected and computed over a period of 30 hours instead of the default 24 hours:

```
<PLACEMENT_POLICY Name="Policy1" Version="5.1" AveragePeriod="30">
```

UNCOMPRESS statement examples

The following example uncompresses in place all of the files with the extension `dbf` on the multi-volume file system `tier3` that have been accessed over 60 minutes ago:

```
<SELECT Flags="Data">
  <PATTERN> *.dbf </PATTERN>
</SELECT>

<UNCOMPRESS>
  <FROM>
    <SOURCE>
      <CLASS> tier3 </CLASS>
    </SOURCE>
  </FROM>
  <WHEN>
    <ACCAGE Units="minutes">
      <MIN Flags="gt">60</MIN>
    </ACCAGE>
  </WHEN>
</UNCOMPRESS>
```

The following example uncompresses in place all of the files with the extension `dbf` on a single volume that have been accessed over 1 minute ago:

```
<SELECT Flags="Data">
  <PATTERN> *.dbf </PATTERN>
</SELECT>

<UNCOMPRESS>
  <WHEN>
    <ACCAGE Units="minutes">
      <MIN Flags="gt">1</MIN>
    </ACCAGE>
  </WHEN>
</UNCOMPRESS>
```

Calculating I/O temperature and access temperature

An important application of VxFS SmartTier is automating the relocation of inactive files to lower cost storage. If a file has not been accessed for the period of time specified in the `<ACCAGE>` element, a scan of the file system should schedule the file for relocation to a lower tier of storage. But, time since last access is inadequate as the only criterion for activity-based relocation.

Why time since last access is inadequate as the only criterion for activity-based relocation:

- Access age is a binary measure. The time since last access of a file is computed by subtracting the time at which the `fsppadm enforce` command is issued from the POSIX `atime` in the file's metadata. If a file is opened the day before the `fsppadm enforce` command, its time since last access is one day, even though it may have been inactive for the month preceding. If the intent of a policy rule is to relocate inactive files to lower tier volumes, it will perform badly against files that happen to be accessed, however casually, within the interval defined by the value of the `<ACCAGE>` pa-rameter.
- Access age is a poor indicator of resumption of significant activity. Using `ACCAGE`, the time since last access, as a criterion for relocating inactive files to lower tier volumes may fail to schedule some relocations that should be performed, but at least this method results in less relocation activity than necessary. Using `ACCAGE` as a criterion for relocating previously inactive files that have become active is worse, because this method is likely to schedule relocation activity that is not warranted. If a policy rule's intent is to cause files that have experienced I/O activity in the recent past to be relocated to higher performing, perhaps more failure tolerant storage, `ACCAGE` is too coarse a filter. For example, in a rule specifying that files on `tier2` volumes that have been accessed within the last three days should be relocated to `tier1` volumes, no distinction is made between a file that was browsed by a single user and a file that actually was used intensively by applications.

SmartTier implements the concept of I/O temperature and access temperature to overcome these deficiencies. A file's I/O temperature is equal to the number of bytes transferred to or from it over a specified period of time divided by the size of the file. For example, if a file occupies one megabyte of storage at the time of an `fsppadm enforce` operation and the data in the file has been completely read or written 15 times within the last three days, VxFS calculates its 3-day average I/O temperature to be 5 (15 MB of I/O ÷ 1 MB file size ÷ 3 days).

Similarly, a file's average access temperature is the number of read or write requests made to it over a specified number of 24-hour periods divided by the number of periods. Unlike I/O temperature, access temperature is unrelated to file size. A large file to which 20 I/O requests are made over a 2-day period has the same average access temperature as a small file accessed 20 times over a 2-day period.

If a file system's active placement policy includes any `<IOTEMP>` or `<ACCESSTEMP>` clauses, VxFS begins policy enforcement by using information in the file system's FCL file to calculate average I/O activity against all files in the file system during the longest `<PERIOD>` specified in the policy. Shorter specified periods are ignored. VxFS uses these calculations to qualify files for I/O temperature-based relocation and deletion.

See “About the Veritas File System File Change Log file” on page 767.

Note: If FCL is turned off, I/O temperature-based relocation will not be accurate. When you invoke the `fsppadm enforce` command, the command displays a warning if the FCL is turned off.

As its name implies, the File Change Log records information about changes made to files in a VxFS file system. In addition to recording creations, deletions, extensions, the FCL periodically captures the cumulative amount of I/O activity (number of bytes read and written) on a file-by-file basis. File I/O activity is recorded in the FCL each time a file is opened or closed, as well as at timed intervals to capture information about files that remain open for long periods.

If a file system's active file placement policy contains `<IOTEMP>` clauses, execution of the `fsppadm enforce` command begins with a scan of the FCL to extract I/O activity information over the period of interest for the policy. The period of interest is the interval between the time at which the `fsppadm enforce` command was issued and that time minus the largest interval value specified in any `<PERIOD>` element in the active policy.

For files with I/O activity during the largest interval, VxFS computes an approximation of the amount of read, write, and total data transfer (the sum of the two) activity by subtracting the I/O levels in the oldest FCL record that pertains to the file from those in the newest. It then computes each file's I/O temperature by dividing its I/O activity by its size at `Tscan`. Dividing by file size is an implicit acknowledgement that relocating larger files consumes more I/O resources than relocating smaller ones. Using this algorithm requires that larger files must have more activity against them in order to reach a given I/O temperature, and thereby justify the resource cost of relocation.

While this computation is an approximation in several ways, it represents an easy to compute, and more importantly, unbiased estimate of relative recent I/O activity upon which reasonable relocation decisions can be based.

File relocation and deletion decisions can be based on read, write, or total I/O activity.

The following XML snippet illustrates the use of `IOTEMP` in a policy rule to specify relocation of low activity files from `tier1` volumes to `tier2` volumes:

```
<RELOCATE>
  <FROM>
    <SOURCE>
      <CLASS>tier1</CLASS>
    </SOURCE>
  </FROM>
  <TO>
```

```
    <DESTINATION>
      <CLASS>tier2</CLASS>
    </DESTINATION>
  </TO>
  <WHEN>
    <IOTEMP Type="nrwbytes}">
      <MAX Flags="lt">3</MAX>
      <PERIOD Units="days">4</PERIOD>
    </IOTEMP>
  </WHEN>
</RELOCATE>
```

This snippet specifies that files to which the rule applies should be relocated from `tier1` volumes to `tier2` volumes if their I/O temperatures fall below 3 over a period of 4 days. The `Type="nrwbytes}"` XML attribute specifies that total data transfer activity, which is the the sum of bytes read and bytes written, should be used in the computation. For example, a 50 megabyte file that experienced less than 150 megabytes of data transfer over the 4-day period immediately preceding the `fsppadm enforce` scan would be a candidate for relocation. VxFS considers files that experience no activity over the period of interest to have an I/O temperature of zero. VxFS relocates qualifying files in the order in which it encounters the files in its scan of the file system directory tree.

Using I/O temperature or access temperature rather than a binary indication of activity, such as the POSIX `atime` or `mtime`, minimizes the chance of not relocating files that were only accessed occasionally during the period of interest. A large file that has had only a few bytes transferred to or from it would have a low I/O temperature, and would therefore be a candidate for relocation to `tier2` volumes, even if the activity was very recent.

But, the greater value of I/O temperature or access temperature as a file relocation criterion lies in upward relocation: detecting increasing levels of I/O activity against files that had previously been relocated to lower tiers in a storage hierarchy due to inactivity or low temperatures, and relocating them to higher tiers in the storage hierarchy.

The following XML snippet illustrates relocating files from `tier2` volumes to `tier1` when the activity level against them increases.

```
<RELOCATE>
  <FROM>
    <SOURCE>
      <CLASS>tier2</CLASS>
    </SOURCE>
  </FROM>
  <TO>
    <DESTINATION>
      <CLASS>tier1</CLASS>
    </DESTINATION>
  </TO>
  <WHEN>
    <IOTEMP Type="nrbytes">
      <MAX Flags="gt">5</MAX>
      <PERIOD Units="days">2</PERIOD>
    </IOTEMP>
  </WHEN>
</RELOCATE>
```

The `<RELOCATE>` statement specifies that files on `tier2` volumes whose I/O temperature as calculated using the number of bytes read is above 5 over a 2-day period are to be relocated to `tier1` volumes. Bytes written to the file during the period of interest are not part of this calculation.

Using I/O temperature rather than a binary indicator of activity as a criterion for file relocation gives administrators a granular level of control over automated file relocation that can be used to attune policies to application requirements. For example, specifying a large value in the `<PERIOD>` element of an upward relocation statement prevents files from being relocated unless I/O activity against them is sustained. Alternatively, specifying a high temperature and a short period tends to relocate files based on short-term intensity of I/O activity against them.

I/O temperature and access temperature utilize the `sqlite3` database for building a temporary table indexed on an inode. This temporary table is used to filter files based on I/O temperature and access temperature. The temporary table is stored in the database file `.__fsppadm_fcliotemp.db`, which resides in the `lost+found` directory of the mount point.

Multiple criteria in file placement policy rule statements

In certain cases, file placement policy rule statements may contain multiple clauses that affect their behavior. In general, when a rule statement contains multiple clauses of a given type, all clauses must be satisfied in order for the statement to be effective. There are four cases of note in which multiple clauses may be used.

Multiple file selection criteria in SELECT statement clauses

Within a single SELECT statement, all the selection criteria clauses of a single type are treated as a selection list. A file need only satisfy a single criterion of a given type to be designated.

In the following example, files in any of the db/datafiles, db/indexes, and db/logs directories, all relative to the file system mount point, would be selected:

```
<SELECT>
  <DIRECTORY Flags="nonrecursive">db/datafiles</DIRECTORY>
  <DIRECTORY Flags="nonrecursive">db/indexes</DIRECTORY>
  <DIRECTORY Flags="nonrecursive">db/logs</DIRECTORY>
</SELECT>
```

This example is in direct contrast to the treatment of selection criteria clauses of different types. When a SELECT statement includes multiple types of file selection criteria, a file must satisfy one criterion of each type in order for the rule's action statements to apply.

In the following example, a file must reside in one of db/datafiles, db/indexes, or db/logs and be owned by one of DBA_Manager, MFG_DBA, or HR_DBA to be designated for possible action:

```
<SELECT>
  <DIRECTORY Flags="nonrecursive">db/datafiles</DIRECTORY>
  <DIRECTORY Flags="nonrecursive">db/indexes</DIRECTORY>
  <DIRECTORY Flags="nonrecursive">db/logs</DIRECTORY>
  <USER>DBA_Manager</USER>
  <USER>MFG_DBA</USER>
  <USER>HR_DBA</USER>
</SELECT>
```

If a rule includes multiple SELECT statements, a file need only satisfy one of them to be selected for action. This property can be used to specify alternative conditions for file selection.

In the following example, a file need only reside in one of db/datafiles, db/indexes, or db/logs or be owned by one of DBA_Manager, MFG_DBA, or HR_DBA to be designated for possible action:

```
<SELECT>
  <DIRECTORY Flags="nonrecursive">db/datafiles</DIRECTORY>
  <DIRECTORY Flags="nonrecursive">db/indexes</DIRECTORY>
  <DIRECTORY Flags="nonrecursive">db/logs</DIRECTORY>
</SELECT>
```

```
<SELECT>
  <USER>DBA_Manager</USER>
  <USER>MFG_DBA</USER>
  <USER>HR_DBA</USER>
</SELECT>
```

Multiple placement classes in <ON> clauses of CREATE statements and in <TO> clauses of RELOCATE statements

Both the `<ON>` clause of the `CREATE` statement and the `<TO>` clause of the `RELOCATE` statement can specify priority ordered lists of placement classes using multiple `<DESTINATION>` XML elements. VxFS uses a volume in the first placement class in a list for the designated purpose of file creation or relocation, if possible. If no volume in the first listed class has sufficient free space or if the file system's volume set does not contain any volumes with that placement class, VxFS uses a volume in the second listed class if possible. If no volume in the second listed class can be used, a volume in the third listed class is used if possible, and so forth.

The following example illustrates of three placement classes specified in the `<ON>` clause of a `CREATE` statement:

```
<CREATE>
  <ON>
    <DESTINATION>
      <CLASS>tier1</CLASS>
    </DESTINATION>
    <DESTINATION>
      <CLASS>tier2</CLASS>
    </DESTINATION>
    <DESTINATION>
      <CLASS>tier3</CLASS>
    </DESTINATION>
  </ON>
</CREATE>
```

In this statement, VxFS would allocate space for newly created files designated by the rule's `SELECT` statement on `tier1` volumes if space was available. If no `tier1` volume had sufficient free space, VxFS would attempt to allocate space on a `tier2` volume. If no `tier2` volume had sufficient free space, VxFS would attempt allocation on a `tier3` volume. If sufficient space could not be allocated on a volume in any of the three specified placement classes, allocation would fail with an `ENOSPC` error, even if the file system's volume set included volumes in other placement classes that did have sufficient space.

The <TO> clause in the RELOCATE statement behaves similarly. VxFS relocates qualifying files to volumes in the first placement class specified if possible, to volumes in the second specified class if not, and so forth. If none of the destination criteria can be met, such as if all specified classes are fully occupied, qualifying files are not relocated, but no error is signaled in this case.

Multiple placement classes in <FROM> clauses of RELOCATE and DELETE statements

The <FROM> clause in RELOCATE and DELETE statements can include multiple source placement classes. However, unlike the <ON> and <TO> clauses, no order or priority is implied in <FROM> clauses. If a qualifying file resides on a volume in any of the placement classes specified in a <FROM> clause, it is relocated or deleted regardless of the position of its placement class in the <FROM> clause list of classes.

Multiple conditions in <WHEN> clauses of RELOCATE and DELETE statements

The <WHEN> clause in RELOCATE and DELETE statements may include multiple relocation criteria. Any or all of <ACCAGE>, <MODAGE>, <SIZE>, and <IOTEMP> can be specified. When multiple conditions are specified, all must be satisfied in order for a selected file to qualify for relocation or deletion.

In the following example, a selected file would have to be both inactive, that is, not accessed, for more than 30 days and larger than 100 megabytes to be eligible for relocation or deletion:

```
<WHEN>
  <ACCAGE Units="days">
    <MIN Flags="gt">30</MIN>
  </ACCAGE>
  <SIZE Units="MB">
    <MIN Flags="gt">100</MIN>
  </SIZE>
</WHEN>
```

You cannot write rules to relocate or delete a single designated set of files if the files meet one of two or more relocation or deletion criteria.

File placement policy rule and statement ordering

You can use the SmartTier graphical user interface (GUI) to create any of four types of file placement policy documents. Alternatively, you can use a text editor or XML editor to create XML policy documents directly. The GUI places policy rule statements in the correct order to achieve the desired behavior. If you use a text editor, it is your responsibility to order policy rules and the statements in them so that the desired behavior results.

The rules that comprise a placement policy may occur in any order, but during both file allocation and `fsppadm enforce` relocation scans, the first rule in which a file is designated by a `SELECT` statement is the only rule against which that file is evaluated. Thus, rules whose purpose is to supersede a generally applicable behavior for a special class of files should precede the general rules in a file placement policy document.

The following XML snippet illustrates faulty rule placement with potentially unintended consequences:

```
<?xml version="1.0"?>
<!DOCTYPE FILE_PLACEMENT_POLICY SYSTEM "placement.dtd">
<FILE_PLACEMENT_POLICY Version="5.0">
  <RULE Name="GeneralRule">
    <SELECT>
      <PATTERN>*</PATTERN>
    </SELECT>
    <CREATE>
      <ON>
        <DESTINATION>
          <CLASS>tier2</CLASS>
        </DESTINATION>
      </ON>
    </CREATE>
    other_statements
  </RULE>
  <RULE Name="DatabaseRule">
    <SELECT>
      <PATTERN>*.db</PATTERN>
    </SELECT>
    <CREATE>
      <ON>
        <DESTINATION>
          <CLASS>tier1</CLASS>
        </DESTINATION>
      </ON>
    </CREATE>
    other_statements
  </RULE>
</FILE_PLACEMENT_POLICY>
```

The GeneralRule rule specifies that all files created in the file system, designated by `<PATTERN>*</PATTERN>`, should be created on `tier2` volumes. The `DatabaseRule` rule specifies that files whose names include an extension of `.db` should be created on `tier1` volumes. The `GeneralRule` rule applies to any file created in the file system, including those with a naming pattern of `*.db`, so the `DatabaseRule` rule will never apply to any file. This fault can be remedied by exchanging the order of the two rules. If the `DatabaseRule` rule occurs first in the policy document, VxFS encounters it first when determining where to new place files whose names follow the pattern `*.db`, and correctly allocates space for them on `tier1` volumes. For files to which the `DatabaseRule` rule does not apply, VxFS continues scanning the policy and allocates space according to the specification in the `CREATE` statement of the `GeneralRule` rule.

A similar consideration applies to statements within a placement policy rule. VxFS processes these statements in order, and stops processing on behalf of a file when it encounters a statement that pertains to the file. This can result in unintended behavior.

The following XML snippet illustrates a `RELOCATE` statement and a `DELETE` statement in a rule that is intended to relocate if the files have not been accessed in 30 days, and delete the files if they have not been accessed in 90 days:

```
<RELOCATE>
  <TO>
    <DESTINATION>
      <CLASS>tier2</CLASS>
    </DESTINATION>
  </TO>
  <WHEN>
    <ACCAGE Units="days">
      <MIN Flags="gt">30</MIN>
    </ACCAGE>
  </WHEN>
</RELOCATE>
<DELETE>
  <WHEN>
    <ACCAGE Units="days">
      <MIN Flags="gt">90</MIN>
    </ACCAGE>
  </WHEN>
</DELETE>
```

As written with the RELOCATE statement preceding the DELETE statement, files will never be deleted, because the <WHEN> clause in the RELOCATE statement applies to all selected files that have not been accessed for at least 30 days. This includes those that have not been accessed for 90 days. VxFS ceases to process a file against a placement policy when it identifies a statement that applies to that file, so the DELETE statement would never occur. This example illustrates the general point that RELOCATE and DELETE statements that specify less inclusive criteria should precede statements that specify more inclusive criteria in a file placement policy document. The GUI automatically produce the correct statement order for the policies it creates.

File placement policies and extending files

In a VxFS file system with an active file placement policy, the placement class on whose volume a file resides is part of its metadata, and is attached when it is created and updated when it is relocated. When an application extends a file, VxFS allocates the incremental space on the volume occupied by the file if possible. If not possible, VxFS allocates the space on another volume in the same placement class. For example, if a file is created on a tier1 volume and later relocated to a tier2 volume, extensions to the file that occur before the relocation have space allocated on a tier1 volume, while those occurring after to the relocation have their space allocated on tier2 volumes. When a file is relocated, all of its allocated space, including the space acquired by extension, is relocated to tier2 volumes in this case.

Using SmartTier with solid state disks

The SmartTier placement policies support SSD-based tiers with the following features:

- Allowance of fine grained temperatures, such as allowing hours as units for the <IOTEMP> and <ACCESSTEMP> criteria
  See "Fine grain temperatures with solid state disks" on page 589.
- Support of the Prefer attribute for the <IOTEMP> and <ACCESSTEMP> criteria
  See "Prefer mechanism with solid state disks" on page 589.
- Provision of a mechanism to relocate based on average I/O activity
  See "Average I/O activity with solid state disks" on page 590.
- Reduction of the intensity and duration of scans to minimize the impact on resources, such as memory, CPU, and I/O bandwidth
  See "Frequent SmartTier scans with solid state disks" on page 590.
- Quick identification of cold files

See “Quick identification of cold files with solid state disks” on page 591.

To gain these benefits, you must modify the existing placement policy as per the latest version of the DTD and assign the policy again. However, existing placement policies continue to function as before. You do not need to update the placement policies if you do not use the new features.

Fine grain temperatures with solid state disks

Before the solid state disk (SSD) enhancements, the SmartTier feature computed temperature values on a day granularity. Day granularity is the I/O activity per day over at least one day. As such, the `<PERIOD>` element had to be in days for the `<IOTEMP>` and `<ACCESSTEMP>` criteria. With SSDs, relocation decisions might need to happen within the day itself, based on I/O activity that Veritas File System (VxFS) measured over a shorter duration. As such, you can now specify "hours" for the `Units` attribute value for the `<IOTEMP>` and `<ACCESSTEMP>` criteria.

See “Specifying the I/O temperature relocation criterion” on page 548.

The following placement policy snippet gives an example of specifying 4 hours as the period of time:

```
<RELOCATE>
   ...
   <WHEN>
      <IOTEMP Type="nwbytes">
         <MIN Flags="gteq"> 2 </MIN>
            <PERIOD Units="hours"> 4 </PERIOD>
      </IOTEMP>
   </WHEN>
</RELOCATE>
```

Prefer mechanism with solid state disks

You can now specify a value for the `Prefer` attribute for the `<IOTEMP>` and `<ACCESSTEMP>` criteria, which gives preference to relocating files.

See “Prefer attribute” on page 549.

In case of a solid state disk (SSD)-based tier, you might want to relocate a file to an SSD as soon as there is a marked increase in the I/O activity. However, once Veritas File System (VxFS) has relocated the file to an SSD, it may be beneficial to keep the file on the SSD as long as the activity remains high to avoid frequent thrashing. You want to watch the activity for some time longer than the time that you watched the activity when you relocated the file to the SSD before you decide to move the file off of the SSD.

The following placement policy snippet gives an example of the Prefer criteria:

```
<RELOCATE>
   ...
   <WHEN>
      <IOTEMP Type="nrbytes" Prefer="high">
         <MIN Flags="gteq"> 3.4 </MIN>
         <PERIOD Units="hours"> 6 </PERIOD>
      </IOTEMP>
   </WHEN>
</RELOCATE>
```

If there are a number of files whose I/O temperature is greater than the given minimum value, the files with the higher temperature are first subject to the RELOCATE operation before the files with the lower temperature. This is particularly useful in case of SSDs, which are limited in size and are expensive. As such, you generally want to use SSDs for the most active files.

Average I/O activity with solid state disks

Before the solid state disk (SSD) enhancements, you were required to specify an absolute value of the temperature when you used the ACCESSTEMP criteria and IOTEMP criteria in the SmartTier placement policies. However, arriving at such absolute numbers is difficult and requires you to experiment and observe data access patterns over a period of time. Moreover, over a period of time, you might have to change this value due to changing access patterns. As such, you might need to repeat the experiment. To ease constructing ACCESSTEMP and IOTEMP-based policies, a new criteria has been introduced: Average.

See “Average I/O activity criteria” on page 550.

Frequent SmartTier scans with solid state disks

You can specify "hours" for the Units attribute value, and as such the I/O stats collection PERIOD can be much shorter than in previous releases. When not using solid state disks (SSDs), you can only specify "days" for the Units attribute value, which might be sufficient for your needs. However, a PERIOD shorter than a day is required in the context of using SSDs since the candidate files and their activity levels can change during the day. As a result, SmartTier must scan more frequently, which leads to a higher scan load on the host systems.

You must satisfy the following conflicting requirements simultaneously:

- Bring down the temperature collection windows to hourly levels.

- Reduce the impact of more frequent scans on resources, such as CPU, I/O, and memory.

The following scheme is an example of one way to reduce the impact of frequent scans:

- Confine the scan to only active files during the PERIOD by focusing only on the files that showed any activity in the File Change Log (FCL) by running the `fsppadm` command with the `-C` option.
  See "Quick identification of cold files with solid state disks" on page 591.
- Scan frequently, such as every few hours. Frequent scans potentially reduce the number of inodes that VxFS touches and logs in the File Change Log (FCL) file, thereby limiting the duration of each scan. As such, the changes that VxFS collects in the FCL file since the last scan provide details on fewer active files.
- Use the `<IOTEMP>` and `<ACCESSTEMP>` criteria to promote files to SSDs more aggressively, which leaves cold files sitting in SSDs.

Quick identification of cold files with solid state disks

The placement mechanism generally leaves the cold files in solid state disks (SSDs) if the files continue to remain inactive. This results in a lack of room for active files if the active files need to be moved into SSDs, and thus results in ineffective use of storage. An SSD enhancement for identifying cold files quickly solves this problem.

The enhancement is a method for quickly identifying files on a particular tier of the SmartTier file system so that the files can be relocated if necessary. The method consists of a map that associates storage devices with the inodes of files residing on the storage devices.

Veritas File System (VxFS) updates the file location map during the following times:

- SmartTier's own file relocations
- On examination of the file system's File Change Log (FCL) for changes that are made outside of SmartTier's scope.

Both of these updates occur during SmartTier's relocation scans, which are typically scheduled to occur periodically. But, you can also update the file location map anytime by running the `fsppadm` command with the `-T` option.

The `-C` option is useful to process active files before any other files. For best results, specify the `-T` option in conjunction with the `-C` option. Specifying both the `-T` option and `-C` option causes the `fsppadm` command to evacuate any cold files first to create room in the SSD tier to accommodate any active files that will be moved into the SSD tier via the `-C` option. Specifying `-C` in conjunction with `-T` confines the scope of the scan, which consumes less time and resources, and thus allows frequent scans to meet the dynamic needs of data placement.

See "Enforcing a placement policy" on page 534.

See the fsppadm(1M) manual page.

With the help of the map, instead of scanning the full file system, you can confine the scan to only the files on the SSD tiers in addition to the active files that VxFS recorded in the FCL. This scheme potentially achieves the dual purpose of reducing the temperature time granularity and at the same time reducing the scan load.

Example placement policy when using solid state disks

The following snippet is one possible placement policy for use with solid state disk (SSD)-based tiers.

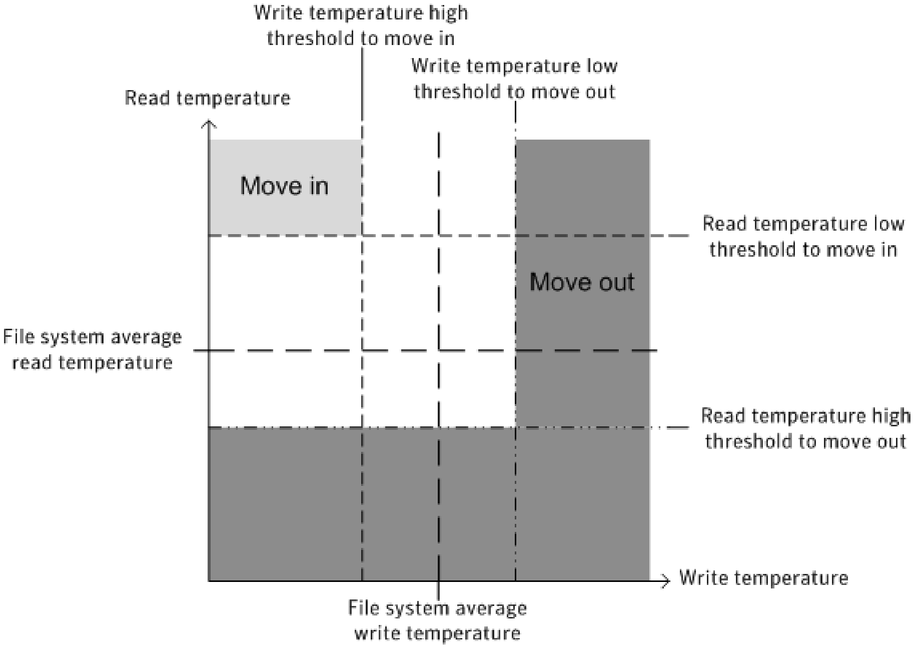
```
<?xml version="1.0"?>
<!DOCTYPE PLACEMENT_POLICY SYSTEM "/opt/VRTSvxfs/etc/placement_policy.dtd">
<PLACEMENT_POLICY Version="5.0" Name="SSD_policy">
  <RULE Flags="data" Name="all_files">
    <COMMENT>
      The first two RELOCATEs will do the evacuation
      out of SSDs to create room for any relocations
      into the SSDs by the third RELOCATE. The parameters
      that can be tuned are basically values for PERIOD and
      the values of MIN and/or MAX as the per the case.
      The values for MIN and MAX are treated as multiples of
      average activity over past 24 hour period.
    </COMMENT>
    <SELECT>
      <PATTERN> * </PATTERN>
    </SELECT>

    <CREATE>
      <COMMENT>
        create files on ssdtier, failing which
        create them on other tiers
      </COMMENT>
      <ON>
        <DESTINATION Flags="any">
          <CLASS> ssdtier </CLASS>
        </DESTINATION>
      </ON>
    </CREATE>
```

```
<RELOCATE>
  <COMMENT>
    Move the files out of SSD if their last 3 hour
    write IOTEMP is more than 1.5 times the last
    24 hour average write IOTEMP. The PERIOD is
    purposely shorter than the other RELOCATEs
    because we want to move it out as soon as
    write activity starts peaking. This criteria
    could be used to reduce SSD wear outs.
  </COMMENT>
  <FROM>
    <SOURCE>
      <CLASS> ssdtier </CLASS>
    </SOURCE>
  </FROM>
  <TO>
    <DESTINATION>
      <CLASS> nonssd_tier </CLASS>
    </DESTINATION>
  </TO>
  <WHEN>
    <IOTEMP Type="nwbytes" Average="*">
      <MIN Flags="gt"> 1.5 </MIN>
      <PERIOD Units="hours"> 3 </PERIOD>
    </IOTEMP>
  </WHEN>
</RELOCATE>

<RELOCATE>
  <COMMENT>
    OR move the files out of SSD if their last 6 hour
    read IOTEMP is less than half the last 24 hour
    average read IOTEMP. The PERIOD is longer,
    we may want to observe longer periods
    having brought the file in. This avoids quickly
    sending the file out of SSDs once in.
  </COMMENT>
  <FROM>
    <SOURCE>
      <CLASS> ssdtier </CLASS>
    </SOURCE>
  </FROM>
```

```
      <TO>
        <DESTINATION>
          <CLASS> nonssd_tier </CLASS>
        </DESTINATION>
      </TO>
      <WHEN>
        <IOTEMP Type="nrbytes" Average="*">
          <MAX Flags="lt"> 0.5 </MAX>
          <PERIOD Units="hours"> 6 </PERIOD>
        </IOTEMP>
      </WHEN>
    </RELOCATE>

    <RELOCATE>
      <COMMENT>
        OR move the files into SSD if their last 3 hour
        read IOTEMP is more than or equal to 1.5 times
        the last 24 hour average read IOTEMP AND
        their last 6 hour write IOTEMP is less than
        half of the last 24 hour average write IOTEMP
      </COMMENT>
      <TO>
        <DESTINATION>
          <CLASS> ssd_tier </CLASS>
        </DESTINATION>
      </TO>
      <WHEN>
        <IOTEMP Type="nrbytes" Prefer="high" Average="*">
          <MIN Flags="gteq"> 1.5 </MIN>
          <PERIOD Units="hours"> 3 </PERIOD>
        </IOTEMP>
        <IOTEMP Type="nwbytes" Average="*">
          <MAX Flags="lt"> 0.5 </MAX>
          <PERIOD Units="hours"> 3 </PERIOD>
        </IOTEMP>
      </WHEN>
    </RELOCATE>
  </RULE>
</PLACEMENT_POLICY>
```

In this placement policy, new files are created on the SSD tiers if space is available, or elsewhere if space is not available. When enforce is performed, the files that are currently in SSDs whose write activity is increased above a threshold or whose read activity fell below a threshold over a given period are moved out of the SSDs. The first two RELOCATEs capture this intent. However, the files whose read activity intensified above a threshold and whose write activity does not exceed a threshold over the given period are moved into SSDs, while giving preference to files with higher read activity.

The following figure illustrates the behavior of the example placement policy:


Write temperature high threshold to move in
Write temperature low threshold to move out
Read temperature
Move in
Read temperature low threshold to move in
Move out
File system average read temperature
Read temperature high threshold to move out
Write temperature
File system average write temperature

The files whose I/O activity falls in the light gray area are good candidates for moving in to SSD storage. These files have less write activity such that they have less impact on wear leveling, and the slower write times to SSDs is less of a factor. These files have intense read activity, which also makes the files ideal for placement on SSDs since read activity does not cause any wear leveling side effects, and reads are faster from SSDs. In contrast, the files whose I/O activity falls in the dark gray area are good candidates to be moved out of SSD storage, since they have more write activity or less read activity. Greater write activity leads to greater wear leveling of the SSDs, and your file system's performance suffers from the slower write times of SSDs. Lesser read activity means that you are not benefitting from the faster read times of SSDs with these files.

Sub-file relocation

The sub-file relocation functionality relocates the data ranges of the specified files to the specified target tier. Only one instance is allowed at a time on a given node for a given mount.

You can move sub-file data by using the `fsppadm subfilemove` command. The application using this framework calls the `fsppadm subfilemove` command periodically via some external scheduling mechanism at desired intervals, to effect relocations. The application might need to call `subfilemove` on each node of a cluster, in case of a cluster file system, if you want to distribute the load. The application also must arrange for initiating this relocation for new mounts and reboots, if the application needs sub−file relocations on those nodes or mounts.

In a cluster situation, since enforcement can happen from multiple nodes even if each node is scheduled to collect statistics at the same intervals, each node's persistence into the database can be slightly out of sync with each other on each node. Since enforcement should follow statistics collection, Veritas recommends that you schedule enforcements on each node with a few minutes of lag so that all nodes can complete the statistics synchronizing by that time. A lag time of 5 minutes suffices in most cases.

Note: You cannot use SmartTier to compress files while using the sub-file relocation functionality.

Moving sub-file data of files to specific target tiers

See the `fsppadm`(1M) manual page.

The following example moves a total of 32 MB in the file `test.dbf` from the offset 64 MB through 96 MB from its existing tier to `tier2`:

```
# cat /var/tmp/list
test.dbf 67108864 100663296 tier2
# fsppadm subfilemove -f /var/tmp/list /mount1
```

Chapter 29

Administering hot-relocation

This chapter includes the following topics:

About hot-relocation

If a volume has a disk I/O failure (for example, the disk has an uncorrectable error), Veritas Volume Manager (VxVM) can detach the plex involved in the failure. I/O stops on that plex but continues on the remaining plexes of the volume.

If a disk fails completely, VxVM can detach the disk from its disk group. All plexes on the disk are disabled. If there are any unmirrored volumes on a disk when it is detached, those volumes are also disabled.

Apparent disk failure may not be due to a fault in the physical disk media or the disk controller, but may instead be caused by a fault in an intermediate or ancillary component such as a cable, host bus adapter, or power supply.

The hot-relocation feature in VxVM automatically detects disk failures, and notifies the system administrator and other nominated users of the failures by electronic mail. Hot-relocation also attempts to use spare disks and free disk space to restore redundancy and to preserve access to mirrored and RAID-5 volumes.

See "How hot-relocation works" on page 598.

If hot-relocation is disabled or you miss the electronic mail, you can use the `vxprint` command or the graphical user interface to examine the status of the disks. You may also see driver error messages on the console or in the system messages file.

Failed disks must be removed and replaced manually.

See "Removing and replacing disks" on page 724.

For more information about recovering volumes and their data after hardware failure, see the *Veritas InfoScale Troubleshooting Guide*.

How hot-relocation works

Hot-relocation allows a system to react automatically to I/O failures on redundant (mirrored or RAID-5) VxVM objects, and to restore redundancy and access to those objects. VxVM detects I/O failures on objects and relocates the affected subdisks to disks designated as spare disks or to free space within the disk group. VxVM then reconstructs the objects that existed before the failure and makes them redundant and accessible again.

When a partial disk failure occurs (that is, a failure affecting only some subdisks on a disk), redundant data on the failed portion of the disk is relocated. Existing volumes on the unaffected portions of the disk remain accessible.

Hot-relocation is only performed for redundant (mirrored or RAID-5) subdisks on a failed disk. Non-redundant subdisks on a failed disk are not relocated, but the system administrator is notified of their failure.

Hot-relocation is enabled by default and takes effect without the intervention of the system administrator when a failure occurs.

The hot-relocation daemon, `vxrelocd`, detects and reacts to VxVM events that signify the following types of failures:

Disk failure

This is normally detected as a result of an I/O failure from a VxVM object. VxVM attempts to correct the error. If the error cannot be corrected, VxVM tries to access configuration information in the private region of the disk. If it cannot access the private region, it considers the disk failed.

Plex failure

This is normally detected as a result of an uncorrectable I/O error in the plex (which affects subdisks within the plex). For mirrored volumes, the plex is detached.

RAID-5 subdisk failure

This is normally detected as a result of an uncorrectable I/O error. The subdisk is detached.

When `vxrelocd` detects such a failure, it performs the following steps:

- `vxrelocd` informs the system administrator (and other nominated users) by electronic mail of the failure and which VxVM objects are affected.
  See “Partial disk failure mail messages” on page 601.
  See “Complete disk failure mail messages” on page 602.
  See “Modifying the behavior of hot-relocation” on page 612.
- `vxrelocd` next determines if any subdisks can be relocated. `vxrelocd` looks for suitable space on disks that have been reserved as hot-relocation spares (marked `spare`) in the disk group where the failure occurred. It then relocates the subdisks to use this space.
- If no spare disks are available or additional space is needed, `vxrelocd` uses free space on disks in the same disk group, except those disks that have been excluded for hot-relocation use (marked `nohotuse`). When `vxrelocd` has relocated the subdisks, it reattaches each relocated subdisk to its plex.
- Finally, `vxrelocd` initiates appropriate recovery procedures. For example, recovery includes mirror resynchronization for mirrored volumes or data recovery for RAID-5 volumes. It also notifies the system administrator of the hot-relocation and recovery actions that have been taken.

If relocation is not possible, `vxrelocd` notifies the system administrator and takes no further action.

Warning: Hot-relocation does not guarantee the same layout of data or the same performance after relocation. An administrator should check whether any configuration changes are required after hot-relocation occurs.

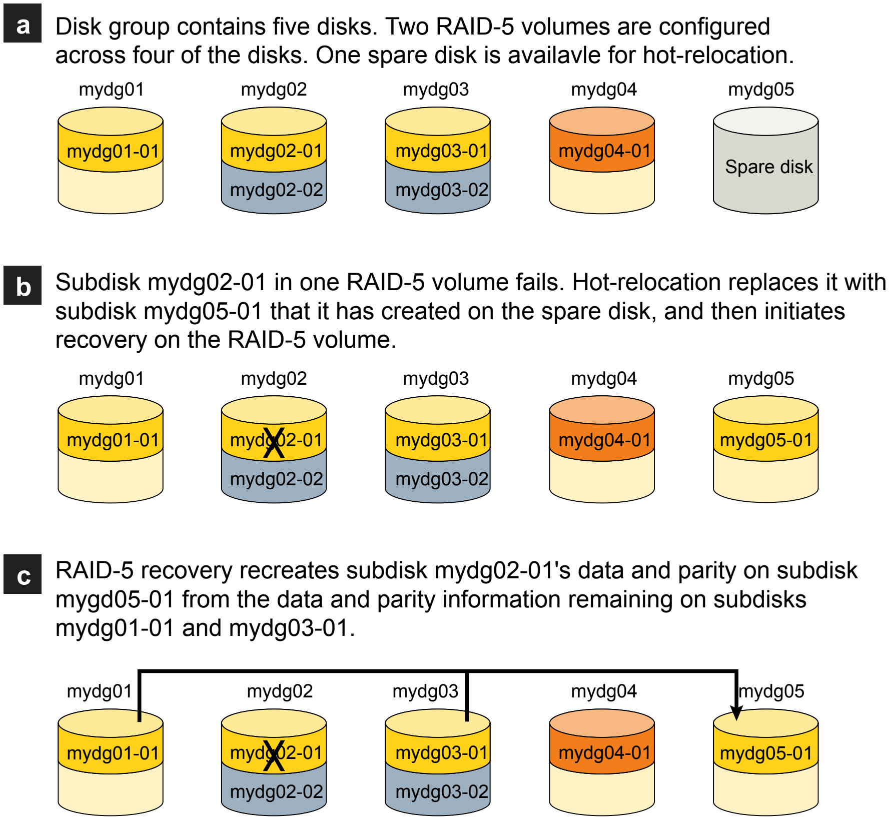
Relocation of failing subdisks is not possible in the following cases:

- The failing subdisks are on non-redundant volumes (that is, volumes of types other than mirrored or RAID-5).
- There are insufficient spare disks or free disk space in the disk group.
- The only available space is on a disk that already contains a mirror of the failing plex.
- The only available space is on a disk that already contains the RAID-5 log plex or one of its healthy subdisks. Failing subdisks in the RAID-5 plex cannot be relocated.
- If a mirrored volume has a dirty region logging (DRL) log subdisk as part of its data plex, failing subdisks belonging to that plex cannot be relocated.
- If a RAID-5 volume log plex or a mirrored volume DRL log plex fails, a new log plex is created elsewhere. There is no need to relocate the failed subdisks of the log plex.

See the `vxrelocd`(1M) manual page.

Figure 29-1 shows the hot-relocation process in the case of the failure of a single subdisk of a RAID-5 volume.

Figure 29-1 Example of hot-relocation for a subdisk in a RAID-5 volume



a Disk group contains five disks. Two RAID-5 volumes are configured across four of the disks. One spare disk is availavle for hot-relocation.

b Subdisk mydg02-01 in one RAID-5 volume fails. Hot-relocation replaces it with subdisk mydg05-01 that it has created on the spare disk, and then initiates recovery on the RAID-5 volume.

c RAID-5 recovery recreates subdisk mydg02-01's data and parity on subdisk mygd05-01 from the data and parity information remaining on subdisks mydg01-01 and mydg03-01.

Partial disk failure mail messages

If hot-relocation is enabled when a plex or disk is detached by a failure, mail indicating the failed objects is sent to `root`. If a partial disk failure occurs, the mail identifies the failed plexes. For example, if a disk containing mirrored volumes fails, you can receive mail information as shown in the following example:

```
To: root
Subject: Volume Manager failures on host teal
Failures have been detected by the Veritas Volume Manager:

failed plexes:
home-02
src-02
```

Mail can be sent to users other than `root`.

See “Modifying the behavior of hot-relocation” on page 612.

You can determine which disk is causing the failures in the above example message by using the following command:

```
# vxstat -g mydg -s -ff home-02 src-02
```

The `-s` option asks for information about individual subdisks, and the `-ff` option displays the number of failed read and write operations. The following output display is typical:

```
                                FAILED
TYP NAME                     READS   WRITES
sd mydg01-04                     0        0
sd mydg01-06                     0        0
sd mydg02-03                     1        0
sd mydg02-04                     1        0
```

This example shows failures on reading from subdisks `mydg02-03` and `mydg02-04` of disk `mydg02`.

Hot-relocation automatically relocates the affected subdisks and initiates any necessary recovery procedures. However, if relocation is not possible or the hot-relocation feature is disabled, you must investigate the problem and attempt to recover the plexes. Errors can be caused by cabling failures, so check the cables connecting your disks to your system. If there are obvious problems, correct them and recover the plexes using the following command:

```
# vxrecover -b -g mydg home src
```

This starts recovery of the failed plexes in the background (the command prompt reappears before the operation completes). If an error message appears later, or if the plexes become detached again and there are no obvious cabling failures, replace the disk.

See “Removing and replacing disks” on page 724.

Complete disk failure mail messages

If a disk fails completely and hot-relocation is enabled, the mail message lists the disk that failed and all plexes that use the disk. For example, you can receive mail as shown in this example display:

```
To: root
Subject: Volume Manager failures on host teal
```

```
Failures have been detected by the Veritas Volume Manager:

failed disks:
mydg02

failed plexes:
home-02
src-02
mkting-01

failing disks:
mydg02
```

This message shows that `mydg02` was detached by a failure. When a disk is detached, I/O cannot get to that disk. The plexes `home-02`, `src-02`, and `mkting-01` were also detached (probably because of the failure of the disk).

One possible cause of the problem could be a cabling error.

See “Partial disk failure mail messages” on page 601.

If the problem is not a cabling error, replace the disk.

See “Removing and replacing disks” on page 724.

How space is chosen for relocation

A spare disk must be initialized and placed in a disk group as a spare before it can be used for replacement purposes. If no disks have been designated as spares when a failure occurs, VxVM automatically uses any available free space in the disk group in which the failure occurs. If there is not enough spare disk space, a combination of spare space and free space is used.

When selecting space for relocation, hot-relocation preserves the redundancy characteristics of the VxVM object to which the relocated subdisk belongs. For example, hot-relocation ensures that subdisks from a failed plex are not relocated to a disk containing a mirror of the failed plex. If redundancy cannot be preserved using any available spare disks and/or free space, hot-relocation does not take place. If relocation is not possible, the system administrator is notified and no further action is taken.

From the eligible disks, hot-relocation attempts to use the disk that is “closest” to the failed disk. The value of “closeness” depends on the controller and disk number of the failed disk. A disk on the same controller as the failed disk is closer than a disk on a different controller.

Hot-relocation tries to move all subdisks from a failing drive to the same destination disk, if possible.

When hot-relocation takes place, the failed subdisk is removed from the configuration database, and VxVM ensures that the disk space used by the failed subdisk is not recycled as free space.

Configuring a system for hot-relocation

By designating spare disks and making free space on disks available for use by hot relocation, you can control how disk space is used for relocating subdisks in the event of a disk failure. If the combined free space and space on spare disks is not sufficient or does not meet the redundancy constraints, the subdisks are not relocated.

Find out which disks are spares or are excluded from hot-relocation.

See "Displaying spare disk information" on page 604.

You can prepare for hot-relocation by designating one or more disks per disk group as hot-relocation spares.

See "Marking a disk as a hot-relocation spare" on page 605.

If required, you can remove a disk from use as a hot-relocation spare

See "Removing a disk from use as a hot-relocation spare" on page 606.

If no spares are available at the time of a failure or if there is not enough space on the spares, free space on disks in the same disk group as where the failure occurred is automatically used, unless it has been excluded from hot-relocation use.

See "Excluding a disk from hot-relocation use" on page 607.

See "Making a disk available for hot-relocation use" on page 608.

Depending on the locations of the relocated subdisks, you can choose to move them elsewhere after hot-relocation occurs.

See "Configuring hot-relocation to use only spare disks" on page 608.

After a successful relocation, remove and replace the failed disk.

See "Removing and replacing disks" on page 724.

Displaying spare disk information

Use the following command to display information about spare disks that are available for relocation:

```
# vxdg [-g diskgroup] spare
```

The following is example output:

```
GROUP DISK      DEVICE       TAG       OFFSET    LENGTH     FLAGS
mydg  mydg02    sdc          sdc       0         658007     s
```

Here mydg02 is the only disk designated as a spare in the mydg disk group. The LENGTH field indicates how much spare space is currently available on mydg02 for relocation.

The following commands can also be used to display information about disks that are currently designated as spares:

- vxdisk list lists disk information and displays spare disks with a spare flag.
- vxprint lists disk and other information and displays spare disks with a SPARE flag.
- The list menu item on the vxdiskadm main menu lists all disks including spare disks.

Marking a disk as a hot-relocation spare

Hot-relocation allows the system to react automatically to I/O failure by relocating redundant subdisks to other disks. Hot-relocation then restores the affected Veritas Volume Manager (VxVM) objects and data. If a disk has already been designated as a spare in the disk group, the subdisks from the failed disk are relocated to the spare disk.

If no disk has been designated as a spare, any suitable free space in the disk group is used.

To designate a disk as a hot-relocation spare, enter the following command:

```
# vxedit [-g diskgroup] set spare=on diskname
```

where *diskname* is the disk media name.

For example, to designate mydg01 as a spare in the disk group, mydg, enter the following command:

```
# vxedit -g mydg set spare=on mydg01
```

You can use the vxdisk list command to confirm that this disk is now a spare; mydg01 should be listed with a spare flag.

Any VxVM disk in this disk group can now use this disk as a spare in the event of a failure. If a disk fails, hot-relocation automatically occurs (if possible). You are notified of the failure and relocation through electronic mail. After successful relocation, you may want to replace the failed disk.

To use vxdiskadm to designate a disk as a hot-relocation spare

1 Select `Mark a disk as a spare for a disk group` from the `vxdiskadm` main menu.

2 At the following prompt, enter a disk media name (such as `mydg01`):

```
Enter disk name [<disk>,list,q,?] mydg01
```

The following notice is displayed when the disk has been marked as spare:

```
VxVM NOTICE V-5-2-219 Marking of mydg01 in mydg as a spare disk
is complete.
```

3 At the following prompt, indicate whether you want to add more disks as spares (`y`) or return to the `vxdiskadm` main menu (`n`):

```
Mark another disk as a spare? [y,n,q,?] (default: n)
```

Any VxVM disk in this disk group can now use this disk as a spare in the event of a failure. If a disk fails, hot-relocation should automatically occur (if possible). You should be notified of the failure and relocation through electronic mail. After successful relocation, you may want to replace the failed disk.

Removing a disk from use as a hot-relocation spare

While a disk is designated as a spare, the space on that disk is not used for the creation of VxVM objects within its disk group. If necessary, you can free a spare disk for general use by removing it from the pool of hot-relocation disks.

To remove a spare from the hot-relocation pool, use the following command:

```
# vxedit [-g diskgroup] set spare=off diskname
```

where *diskname* is the disk media name.

For example, to make `mydg01` available for normal use in the disk group, `mydg`, use the following command:

```
# vxedit -g mydg set spare=off mydg01
```

To use vxdiskadm to remove a disk from the hot-relocation pool

1. Select `Turn off the spare flag on a disk` from the `vxdiskadm` main menu.
2. At the following prompt, enter the disk media name of a spare disk (such as `mydg01`):

   ```
   Enter disk name [<disk>,list,q,?] mydg01
   ```

   The following confirmation is displayed:

   ```
   VxVM NOTICE V-5-2-143 Disk mydg01 in mydg no longer marked as
   a spare disk.
   ```

3. At the following prompt, indicate whether you want to disable more spare disks (`y`) or return to the `vxdiskadm` main menu (`n`):

   ```
   Turn off spare flag on another disk? [y,n,q,?] (default: n)
   ```

Excluding a disk from hot-relocation use

To exclude a disk from hot-relocation use, use the following command:

```
# vxedit [-g diskgroup] set nohotuse=on diskname
```

where *diskname* is the disk media name.

To use vxdiskadm to exclude a disk from hot-relocation use

1. Select `Exclude a disk from hot-relocation use` from the `vxdiskadm` main menu.
2. At the following prompt, enter the disk media name (such as `mydg01`):

   ```
   Enter disk name [<disk>,list,q,?] mydg01
   ```

   The following confirmation is displayed:

   ```
   VxVM INFO V-5-2-925 Excluding mydg01 in mydg from hot-
   relocation use is complete.
   ```

3. At the following prompt, indicate whether you want to add more disks to be excluded from hot-relocation (`y`) or return to the `vxdiskadm` main menu (`n`):

   ```
   Exclude another disk from hot-relocation use? [y,n,q,?]
   (default: n)
   ```

Making a disk available for hot-relocation use

Free space is used automatically by hot-relocation in case spare space is not sufficient to relocate failed subdisks. You can limit this free space usage by hot-relocation by specifying which free disks should not be touched by hot-relocation. If a disk was previously excluded from hot-relocation use, you can undo the exclusion and add the disk back to the hot-relocation pool.

To make a disk available for hot-relocation use, use the following command:

```
# vxedit [-g diskgroup] set nohotuse=off diskname
```

To use vxdiskadm to make a disk available for hot-relocation use

1. Select `Make a disk available for hot-relocation use` from the `vxdiskadm` main menu.
2. At the following prompt, enter the disk media name (such as `mydg01`):

   ```
   Enter disk name [<disk>,list,q,?] mydg01
   ```

   The following confirmation is displayed:

   ```
   V-5-2-932 Making mydg01 in mydg available for hot-relocation
   use is complete.
   ```

3. At the following prompt, indicate whether you want to add more disks to be excluded from hot-relocation (`y`) or return to the `vxdiskadm` main menu (`n`):

   ```
   Make another disk available for hot-relocation use? [y,n,q,?]
   (default: n)
   ```

Configuring hot-relocation to use only spare disks

If you want VxVM to use only spare disks for hot-relocation, add the following line to the file `/etc/default/vxassist`:

```
spare=only
```

If not enough storage can be located on disks marked as spare, the relocation fails. Any free space on non-spare disks is not used.

Moving relocated subdisks

When hot-relocation occurs, subdisks are relocated to spare disks and/or available free space within the disk group. The new subdisk locations may not provide the same performance or data layout that existed before hot-relocation took place. You can move the relocated subdisks (after hot-relocation is complete) to improve performance.

You can also move the relocated subdisks of the spare disks to keep the spare disk space free for future hot-relocation needs. Another reason for moving subdisks is to recreate the configuration that existed before hot-relocation occurred.

During hot-relocation, one of the electronic mail messages sent to `root` is shown in the following example:

```
To: root
Subject: Volume Manager failures on host teal

Attempting to relocate subdisk mydg02-03 from plex home-02.
Dev_offset 0 length 1164 dm_name mydg02 da_name sdh.
The available plex home-01 will be used to recover the data.
```

This message has information about the subdisk before relocation and can be used to decide where to move the subdisk after relocation.

Here is an example message that shows the new location for the relocated subdisk:

```
To: root
Subject: Attempting VxVM relocation on host teal

Volume home Subdisk mydg02-03 relocated to mydg05-01,
but not yet recovered.
```

Before you move any relocated subdisks, fix or replace the disk that failed.

See “Removing and replacing disks” on page 724.

Once this is done, you can move a relocated subdisk back to the original disk as described in the following sections.

Warning: During subdisk move operations, RAID-5 volumes are not redundant.

Moving relocated subdisks using vxunreloc

VxVM hot-relocation allows the system to automatically react to I/O failures on a redundant VxVM object at the subdisk level and then take necessary action to make the object available again. This mechanism detects I/O failures in a subdisk, relocates the subdisk, and recovers the plex associated with the subdisk. After the disk has been replaced, `vxunreloc` allows you to restore the system back to the configuration that existed before the disk failure. `vxunreloc` allows you to move the hot-relocated subdisks back onto a disk that was replaced due to a failure.

When `vxunreloc` is invoked, you must specify the disk media name where the hot-relocated subdisks originally resided. When `vxunreloc` moves the subdisks, it moves them to the original offsets. If you try to unrelocate to a disk that is smaller than the original disk that failed,`vxunreloc` does nothing except return an error.

vxunreloc provides an option to move the subdisks to a different disk from where they were originally relocated. It also provides an option to unrelocate subdisks to a different offset as long as the destination disk is large enough to accommodate all the subdisks.

If `vxunreloc` cannot replace the subdisks back to the same original offsets, a force option is available that allows you to move the subdisks to a specified disk without using the original offsets.

See the `vxunreloc`(1M) manual page.

The examples in the following sections demonstrate the use of `vxunreloc`.

Moving hot-relocated subdisks back to their original disk

Assume that `mydg01` failed and all the subdisks were relocated. After `mydg01` is replaced, `vxunreloc` can be used to move all the hot-relocated subdisks back to `mydg01`.

```
# vxunreloc -g mydg mydg01
```

Moving hot-relocated subdisks back to a different disk

The `vxunreloc` utility provides the `-n` option to move the subdisks to a different disk from where they were originally relocated.

Assume that `mydg01` failed, and that all of the subdisks that resided on it were hot-relocated to other disks. `vxunreloc` provides an option to move the subdisks to a different disk from where they were originally relocated. After the disk is repaired, it is added back to the disk group using a different name, for example, `mydg05`. If you want to move all the hot-relocated subdisks back to the new disk, the following command can be used:

```
# vxunreloc -g mydg -n mydg05 mydg01
```

The destination disk should have at least as much storage capacity as was in use on the original disk. If there is not enough space, the unrelocate operation will fail and none of the subdisks will be moved.

Forcing hot-relocated subdisks to accept different offsets

By default, vxunreloc attempts to move hot-relocated subdisks to their original offsets. However, vxunreloc fails if any subdisks already occupy part or all of the area on the destination disk. In such a case, you have two choices:

- Move the existing subdisks somewhere else, and then re-run vxunreloc.
- Use the -f option provided by vxunreloc to move the subdisks to the destination disk, but leave it to vxunreloc to find the space on the disk. As long as the destination disk is large enough so that the region of the disk for storing subdisks can accommodate all subdisks, all the hot-relocated subdisks will be unrelocated without using the original offsets.

Assume that mydg01 failed and the subdisks were relocated and that you want to move the hot-relocated subdisks to mydg05 where some subdisks already reside. You can use the force option to move the hot-relocated subdisks to mydg05, but not to the exact offsets:

```
# vxunreloc -g mydg -f -n mydg05 mydg01
```

Examining which subdisks were hot-relocated from a disk

If a subdisk was hot relocated more than once due to multiple disk failures, it can still be unrelocated back to its original location. For instance, if mydg01 failed and a subdisk named mydg01-01 was moved to mydg02, and then mydg02 experienced disk failure, all of the subdisks residing on it, including the one which was hot-relocated to it, will be moved again. When mydg02 was replaced, a vxunreloc operation for mydg02 will do nothing to the hot-relocated subdisk mydg01-01. However, a replacement of mydg01 followed by a vxunreloc operation, moves mydg01-01 back to mydg01 if vxunreloc is run immediately after the replacement.

After the disk that experienced the failure is fixed or replaced, vxunreloc can be used to move all the hot-relocated subdisks back to the disk. When a subdisk is hot-relocated, its original disk-media name and the offset into the disk are saved in the configuration database. When a subdisk is moved back to the original disk or to a new disk using vxunreloc, the information is erased. The original disk-media name and the original offset are saved in the subdisk records. To print all of the subdisks that were hot-relocated from mydg01 in the mydg disk group, use the following command:

```
# vxprint -g mydg -se 'sd_orig_dmname="mydg01"'
```

Restarting vxunreloc after errors

vxunreloc moves subdisks in three phases:

- vxunreloc creates as many subdisks on the specified destination disk as there are subdisks to be unrelocated. The string UNRELOC is placed in the comment field of each subdisk record.
  Creating the subdisk is an all-or-nothing operation. If vxunreloc cannot create all the subdisks successfully, none are created, and vxunreloc exits.
- vxunreloc moves the data from each subdisk to the corresponding newly created subdisk on the destination disk.
- When all subdisk data moves have been completed successfully, vxunreloc sets the comment field to the null string for each subdisk on the destination disk whose comment field is currently set to UNRELOC.

The comment fields of all the subdisks on the destination disk remain marked as UNRELOC until phase 3 completes. If its execution is interrupted, vxunreloc can subsequently re-use subdisks that it created on the destination disk during a previous execution, but it does not use any data that was moved to the destination disk.

If a subdisk data move fails, vxunreloc displays an error message and exits. Determine the problem that caused the move to fail, and fix it before re-executing vxunreloc.

If the system goes down after the new subdisks are created on the destination disk, but before all the data has been moved, re-execute vxunreloc when the system has been rebooted.

Warning: Do not modify the string UNRELOC in the comment field of a subdisk record.

Modifying the behavior of hot-relocation

Hot-relocation is turned on as long as the vxrelocd process is running. You should normally leave hot-relocation turned on so that you can take advantage of this feature if a failure occurs. However, if you choose to disable hot-relocation (perhaps because you do not want the free space on your disks to be used for relocation), you can prevent vxrelocd from starting at system startup time by editing the vxvm-recover startup file that invokes vxrelocd.

If the hot-relocation daemon is disabled, then automatic storage reclamation on deleted volumes is also disabled.

You can alter the behavior of vxrelocd as follows:

1. To prevent `vxrelocd` starting, comment out the entry that invokes it in the startup file:

   ```
   # nohup vxrelocd root &
   ```

2. By default, `vxrelocd` sends electronic mail to `root` when failures are detected and relocation actions are performed. You can instruct `vxrelocd` to notify additional users by adding the appropriate user names as shown here:

   ```
   # nohup vxrelocd root user1 user2 &
   ```

3. To reduce the impact of recovery on system performance, you can instruct `vxrelocd` to increase the delay between the recovery of each region of the volume, as shown in the following example:

   ```
   # nohup vxrelocd -o slow[=IOdelay] root &
   ```

   where the optional *IOdelay* value indicates the desired delay in milliseconds. The default value for the delay is 250 milliseconds.

Chapter 30

Deduplicating data

This chapter includes the following topics:

About deduplicating data

The data deduplication feature eliminates duplicate blocks used by your data by comparing blocks across the file system. When the data deduplication feature finds a duplicate block, it removes the space used and instead creates a pointer to the common block. If you change the duplicate file, thus making the files no longer share the same block, then that changed block is saved to disk instead of the pointer. You can perform post-process periodic deduplication in a file system to eliminate duplicate data without any continuous cost in CPU overhead. You can verify whether data is duplicated on demand, and then efficiently and securely eliminate the duplicates. The deduplication process performs the following tasks:

- Scans the file system for changes
- Fingerprints the data
- Identifies duplicates
- Eliminates duplicates after verifying the duplicates

The amount of space savings that you get from deduplicating depends on your data. Deduplicating different data gives different space savings.

You deduplicate data using the `fsdedupadm` command.

See the `fsdedupadm`(1M) manual page.

Deduplication functionality is available with both Veritas InfoScale Storage and Veritas InfoScale Enterprise licenses.

About deduplication chunk size

The deduplication chunk size, which is also referred to as deduplication granularity, is the unit at which fingerprints are computed. A valid chunk size is between 4k and 128k and power of two. Once set, the only way to change the chunk size is to remove and re-enable deduplication on the file system.

You should carefully select the chunk size, as the size has significant impact on deduplication as well as resource requirements. The size directly affects the number of fingerprint records in the deduplication database as well as temporary space required for sorting these records. A smaller chunk size results in a large number of fingerprints and hence requires a significant amount of space for the deduplication database.

While the amount of storage that you save after deduplication depends heavily on the dataset and distribution of duplicates within the dataset, the chunk size can also affect the savings significantly. You must understand your dataset to get the best results after deduplication. A general rule of thumb is that a smaller chunk size saves more storage. A smaller chunk size results in more granular fingerprints and in general results in identifying more duplicates. However, smaller chunks have additional costs in terms of database size, deduplication time, and, more importantly, fragmentation. The deduplication database size can be significantly large for small chunk sizes. Higher fragmentation normally results in more file system metadata and hence can require more storage. The space consumed by the deduplication database and the increased file system metadata can reduce the savings achieved via deduplication. Additionally, fragmentation can also have a negative effect on performance. The Veritas File System (VxFS) deduplication algorithms try to reduce fragmentation by coalescing multiple contiguous duplicate chunks.

Larger chunk sizes normally result in a smaller deduplication database size, faster deduplication, and less fragmentation. These benefits sometimes come at the cost of less storage savings. If you have a large number duplicate files that are small in size, you still can choose a chunk size that is larger than the file size. A larger chunk size does not affect the deduplication of files that are smaller than the chunk size. In such cases, the fingerprint is calculated on the whole file, and the files are still deduplicated.

Veritas recommends a chunk size of 16k or higher.

The space consumed by the deduplication database is a function of the amount of data in the file system and the deduplication chunk size. The space consumed by the deduplication database grows with time as new data is added to file system. Additional storage is required for temporary use, such as sorting fingerprints. The temporary storage may be freed after the work completes. Ensure that sufficient free space is available for deduplication to complete successfully. The deduplication might not start if the file system free space is less than approximately 15%. The deduplication sometimes needs more than 15% free space for smaller chunk sizes. In general, the space consumed reduces significantly with larger chunk sizes. Veritas recommends that you have approximately 20% free space for 4k chunks.

Deduplication and file system performance

Veritas File System (VxFS) deduplication uses shared extents to save storage when duplicate data is identified. Shared extents can significantly boost read performance for certain types of applications. These benefits are the result of the innovative use of file system page cache for data that resides in shared extents.

The description of the FileSnaps feature contains more information about shared extents.

See “About FileSnaps” on page 373.

In general, any application or set of applications that read data residing in shared extents via multiple files are expected to have better read performance.

About the deduplication scheduler

The deduplication scheduler is a daemon that runs on all nodes and is responsible for deduplicating data as per the user-specified schedule.

The deduplication scheduler does not start automatically after enabling deduplication on the file system. You have to externally start the scheduler.

Each file system can have its own schedule. The schedule and other configuration information for a given file system is stored within the file system. The location of the configuration file is `lost+found/dedup/local_config`.

The scheduler checks the configuration file every 30 minutes for changes and incorporates the changes if there are any. This periodic check also looks for newly mounted file systems. You can incorporate configuration changes immediately by restarting the scheduler.

When using the scheduler to deduplicate a file system's data automatically, the evaluation of changes in the file system is done by the File Change Log (FCL) feature. Scheduling deduplication to occur too infrequently in terms of days can cause the FCL to roll over, and thus the FCL feature can miss changes to the file system.

Veritas recommends that you schedule deduplication when the system activity is low. This ensures that the scheduler does not interfere with the regular workload of the system.

Deduplicating data

You deduplicate data using the `fsdedupadm` command. The `fsdedupadm` command performs the following functions:

Functionality	Command syntax
Enable the deduplication of a file system.	**fsdedupadm enable [-c *chunk_size*] [-q] *mount_point***
Disable the deduplication of a file system.	**fsdedupadm disable [-q] *mount_point***
Query the deduplication configuration of a file system.	**fsdedupadm list *mount_point*|all**
Start a deduplication run on a file system.	**fsdedupadm start [-s] [-q] *mount_point***
Stop a deduplication run on a file system.	**fsdedupadm stop [-q] *mount_point***
Query the deduplication status of a file system.	**fsdedupadm status *mount_point*|all**
Enable or disable the skipping of shared extents.	**fsdedupadm skipshared {true|false} *mount_point***
Set the node on which the scheduled deduplication job will run.	**fsdedupadm setnodelist *nodelist* *mount_point*|all**
Set the deduplication schedule for a file system.	**fsdedupadm setschedule *time* *mount_point***
Initiate a deduplication dry run on a file system.	**fsdedupadm dryrun [-o threshold=#] *mount_point***

Functionality	Command syntax
Remove the deduplication configuration file and deduplication database on a file system.	**fsdedupadm remove *mount_point***

For more information about the keywords, see the `fsdedupadm`(1M) manual page.

The following example creates a file system, creates duplicate data on the file system, and deduplicates the file system.

Example of deduplicating a file system

1. Create the file system `fsvol1`:

   ```
   # mkfs -t vxfs /dev/vx/rdsk/fsdg/fsvol1
   ```

2. Mount the file system as `/mnt1`:

   ```
   # mount -t vxfs /dev/vx/dsk/fsdg/fsvol1 /mnt1
   ```

3. Make a temporary directory, `temp1`, on `/mnt1` and copy the `file1` file into the directory:

   ```
   # mkdir /mnt1/temp1
   # cd /mnt1/temp1
   # cp /root/file1 .
   # /opt/VRTS/bin/fsadm -S shared /mnt1
   ```

```
Mountpoint    Size(KB)    Available(KB)   Used(KB)   Logical_Size(KB) Space_Saved(KB)
          /mnt1        20971520      19335962      346609            602609        0
```

   The `file1` file is approximately 250 MB, as shown by the output of the `fsadm` command.

4 Make another temporary directory, `temp2`, and copy the same file, `file1`, into the new directory:

```
# mkdir /mnt1/temp2
# cd /mnt1/temp2
# cp /root/file1 .
# /opt/VRTS/bin/fsadm -S shared /mnt1
```

```
Mountpoint  Size(KB) Available(KB) Used(KB) Logical_Size(KB) Space_Saved(KB)
/mnt1       4194304  3588700       548740   548740           0%
```

By copying the same file into `temp2`, you now have duplicate data. The output of the `fsadm` command show that you are now using twice the amount of space.

5 Enable deduplication on the mount point `/mnt1`:

```
# /opt/VRTS/bin/fsdedupadm enable -c 4096 /mnt1
# /opt/VRTS/bin/fsdedupadm list /mnt1
```

```
Chunksize Enabled SkipShared Schedule NodeList            Filesystem
---------------------------------------------------------------------
4096      YES     True       NONE     node1               /mnt1
```

6 Start a deduplication run on the mount point `/mnt1`:

```
# /opt/VRTS/bin/fsdedupadm start /mnt1
UX:vxfs fsdedupadm: INFO: V-3-20: 0000: deduplication is started
on /mnt1.
```

7 Check status of deduplication:

```
# /opt/VRTS/bin/fsdedupadm status /mnt1
Saving    Status    Node            Type        Filesystem
-------------------------------------------------------------------------
42%       COMPLETED fsaixp702-v05   MANUAL      /mnt1
2014/09/23 22:48:22 Begin full scan.
2014/09/23 22:51:45 End detecting duplicates and filesystem changes.
```

8 Verify that the file system was deduplicated by checking how much space you are using:

```
# /opt/VRTS/bin/fsadm -S shared /mnt1
```

```
Mountpoint    Size(KB)    Available(KB)   Used(KB)   Logical_Size(KB)  Space_Saved(KB)
             /mnt1       20971520      19335962       346609            602609        256000
```

The output shows that the used space is nearly identical to when you had only one copy of the `file1` file on the file system.

Enabling and disabling deduplication on a file system

You must enable deduplication on a file system by using the `fsdedupadm enable` command before you can use any of the deduplication functionality.

The following example enables deduplication on the file system mounted at `/mnt1`, and specifies a chunk size of 4096 bytes for deduplication:

```
# /opt/VRTS/bin/fsdedupadm enable -c 4096 /mnt1
```

You can disable deduplication on a file system by using the `fsdedupadm disable` command.

The following example disables deduplication on the file system mounted at `/mnt1`:

```
# /opt/VRTS/bin/fsdedupadm disable /mnt1
```

Scheduling deduplication of a file system

You can set a schedule to deduplicate a file system automatically by using the `fsdedupadm setschedule` command. You can specify two categories of schedule options: run periodicity, and type periodicity. The granularity of schedule is limited to the time of day and the day of the month. The `fsdedupadm` command applies any relevant File Change Log tunables when setting the schedule.

See “Veritas File System File Change Log administrative interface” on page 768.

You must enable deduplication on the file system before you can set a schedule.

See “Enabling and disabling deduplication on a file system” on page 620.

You can schedule the deduplication run every hour or every specified number of hours, and every day or every specified number of days. You can also schedule the actual deduplication run to occur each time, or every specified number of times that the scheduled time elapses. During times that deduplication does not occur, the deduplication run only updates the fingerprints in the database.

The schedule commands are not cumulative. If a deduplication schedule comes up while the previous deduplication process is running for any reason, the upcoming deduplication is discarded and an warning message displays.

You can remove a schedule by specifying an empty string enclosed by double quotes ("") for the schedule.

See the `fsdedupadm`(1M) manual page.

You must start the `fsdedupschd` daemon before scheduling the task:

For RHEL 7, SLES 12, and supported RHEL-compatible distributions:

```
# systemctl enable fsdedupschd
# systemctl start fsdedupschd
```

For earlier versions of RHEL, SLES and supported RHEL-compatible distributions:

```
# chkconfig --add fsdedupschd
# service fsdedupschd start
```

In the following example, deduplication for the file system `/vx/fs1` will be done at midnight, every other day:

```
# fsdedupadm setschedule "0 */2" /vx/fs1
```

In the following example, deduplication for the file system `/vx/fs1` will be done twice every day, once at midnight and once at noon:

```
# fsdedupadm setschedule "0,12 *" /vx/fs1
```

In the following example, deduplication for the file system `/vx/fs1` will be done four times every day, but only the fourth deduplication run will actually deduplicate the file system. The other runs will do the scanning and processing. This option achieves load distribution not only in a system, but also across the cluster.

```
# fsdedupadm setschedule "0,6,12,18 * 4" /vx/fs1
```

The following example removes the deduplication schedule from the file system `/vx/fs1`:

```
# fsdedupadm setschedule "" /vx/fs1
```

Performing a deduplication dry run

You can perform a dry run to determine the space savings of deduplication without actually modifying the file system. You must enable deduplication on the file system before you can perform a dry run. You can perform a dry run only on a file system that has not been deduplicated previously.

See "Enabling and disabling deduplication on a file system" on page 620.

The following command initiates a deduplication dry run on the file system /mnt1:

```
# fsdedupadm dryrun /mnt1
```

You can specify fsdedupadm to perform the actual deduplication by specifying the -o threshold option. In this case, the fsdedupadm command performs an actual deduplication run if the expected space savings meets the specified threshold.

The following command initiates a deduplication dry run on the file system /mnt1, and performs the actual deduplication if the expected space savings crosses the threshold of 60 percent:

```
# fsdedupadm dryrun -o threshold=60 /mnt1
```

Specifying the -o threshold option causes the fsdedupadm command to take Storage Checkpoints and enable the File Change Log for the file system.

Querying the deduplication status of a file system

You can query the deduplication status of a file system by using the fsdedupadm status command.

You must enable deduplication on the file system before you can query the deduplication status.

See "Enabling and disabling deduplication on a file system" on page 620.

The following command queries the deduplication status of the file system /mnt1:

```
# fsdedupadm status /mnt1
```

The following command queries the deduplication status of all running deduplication jobs:

```
# fsdedupadm status all
```

Starting and stopping the deduplication scheduler daemon

The state of the deduplication scheduler daemon, `fsdedupschd`, is maintained across reboots. If you started the `fsdedupschd` daemon prior to a reboot, the daemon is automatically restarted after the reboot. If you stopped the `fsdedupschd` daemon prior to a reboot, it remains stopped after the reboot. The default `fsdedupschd` daemon state is stopped.

You must enable deduplication on the file system before you can start or stop the scheduler daemon.

See "Enabling and disabling deduplication on a file system" on page 620.

The following command starts the `fsdedupschd` daemon:

For RHEL 7, SLES 12, and supported RHEL-compatible distributions:

```
# systemctl enable fsdedupschd
# systemctl start fsdedupschd
```

For earlier versions of RHEL, SLES and supported RHEL-compatible distributions:

```
# chkconfig --add fsdedupschd
# service fsdedupschd start
```

The following command stops the `fsdedupschd` daemon:

For RHEL 7, SLES 12, and supported RHEL-compatible distributions:

```
# systemctl stop fsdedupschd
# systemctl disable fsdedupschd
```

For earlier versions of RHEL, SLES and supported RHEL-compatible distributions:

```
# service fsdedupschd stop
# chkconfig --del fsdedupschd
```

Deduplication results

The nature of the data is very important for deciding whether to enable deduplication. Databases or media files, such as JPEG, MP3, and MOV, might not be the best candidates for deduplication, as they have very little or no duplicate data. Virtual machine boot image files (`vmdk` files), user home directories, and file system with multiple copies of files are good candidates for deduplication. While smaller deduplication chunk size normally results into higher storage saving, it takes longer to deduplicate and requires a larger deduplication database.

Deduplication supportability

Veritas File System (VxFS) supports deduplication on file system disk layout version 9 and later.

Deduplication use cases

The following list includes several cases for which you would want to use the deduplication feature:

Home directories	User home directories often have multiple versions of the same files or file that have similar content, and therefore have redundant data that you can deduplicate.
Source code directories	Source code repositories usually have multiple files with incremental changes. The data that does not change from one file to the next can be deduplicated.
`vmdk` files	Once several virtual machines are cloned by using the FileSnap feature, the cloned virtual machines are subjected to operating system and security patches over their lifetime. As individual virtual machines cloned from a common source–the golden image–deviate from the source as a result of such activity, there is large amount of common content between them. Over time, this results in the loss of the initial storage savings. Deduplication of the new blocks added to these files restores the storage savings.

Deduplication limitations

The deduplication feature has the following limitations:

- A full backup of a deduplicated Veritas File System (VxFS) file system can require as much space in the target as a file system that has not been deduplicated. For example, if you have 2 TB of data that occupies 1 TB worth of disk space in the file system after deduplication, this data requires 2 TB of space on the target to back up the file system, assuming that the backup target does not do any deduplication. Similarly, when you restore such a file system, you must have 2 TB on the file system to restore the complete data. However, this freshly restored file system can be deduplicated again to regain the space savings. After a full file system restore, Veritas recommends that you remove any existing deduplication configuration using the `fsdedupadm remove` command and that you reconfigure deduplication using the `fsdedupadm enable` command.

- Deduplication is limited to a volume's primary fileset.
- Deduplication does not support mounted clone and snapshot mounted file system.
- After you restore data from a backup, you must deduplicate the restored data to regain any space savings provided by deduplication.
- If you use the cross-platform data sharing feature to convert data from one platform to another, you must remove the deduplication configuration file and database, re-enable deduplication, and restart deduplication after the conversion. The following example shows the commands that you must run, and you must run the commands in the order shown:

  ```
  # /opt/VRTS/bin/fsdedupadm remove /mnt1
  # /opt/VRTS/bin/fsdedupadm enable /mnt1
  # /opt/VRTS/bin/fsdedupadm start /mnt1
  ```

- You cannot use the FlashBackup feature of NetBackup in conjunction with the data deduplication feature, because FlashBackup does not support disk layout Version 8 and 9.

Chapter 31

Compressing files

This chapter includes the following topics:

About compressing files

Compressing files reduces the space used, while retaining the accessibility of the files and being transparent to applications. Compressed files look and behave almost exactly like uncompressed files: the compressed files have the same name, and can be read and written as with uncompressed files. Reads cause data to be uncompressed in memory, only; the on-disk copy of the file remains compressed. In contrast, after a write, the new data is uncompressed on disk.

Only user data is compressible. You cannot compress Veritas File System (VxFS) metadata.

After you compress a file, the inode number does not change, and file descriptors opened before the compression are still valid after the compression.

Compression is a property of a file. Thus, if you compress all files in a directory, for example, any files that you later copy into that directory do not automatically get compressed. You can compress the new files at any time by compressing the files in the directory again.

You compress files with the `vxcompress` command.

See “Compressing files with the vxcompress command” on page 628.

See the `vxcompress`(1) manual page.

To compress files, you must have VxFS file systems with disk layout Version 9 or later.

Note: When you back up compressed files to tape, the backup program stores the data in an uncompressed format. The files are uncompressed in memory and subsequently written to the tape. This results in increased CPU and memory usage when you back up compressed files.

About the compressed file format

A compressed file is a file with compressed extents. A `vxcompress` call compresses all extents of a file. However, writes to the file cause the affected extents to get uncompressed; the result can be files with both compressed and uncompressed extents.

About the file compression attributes

When you compress a file with the `vxcompress` command, `vxcompress` attaches the following information to the inode:

- Compression algorithm
- Compression strength, which is a number from 1 to 9
- Compression block size

This information is referred to as the file compression attributes. The purpose of the attributes are to collect the parameters used to create the compressed file. The information can then be read by a backup program.

The file compression attributes guarantee that a particular compressed file can only use one type and strength of compression. Recompressing a file using different attributes fails. To change the file compression attributes, you must explicitly uncompress first, and then recompress with the new options, even in the case where all extents are already uncompressed.

The file compression attributes do not indicate if all extents are compressed. Some extents might be incompressible, and other extents or even all extents might be uncompressed due to writes, but the file compression attributes remain. Only an explicit file uncompression can remove the attributes.

About the file compression block size

The file compression algorithm compresses data in the specified block size, which defaults to 1MB. Each compression block has its own extent descriptor in the inode. If the file or the last extent is smaller than the compression block size, then that smaller size gets compressed. The maximum block size is 1MB.

Extents with data that cannot be compressed are still marked as compressed extents. Even though such extents could not be compressed, marking these extents as compressed allows successive compression runs to skip these extents to save time. Shared extents cannot be compressed and do not get marked as compressed. Since the file compression algorithm looks at fixed-size blocks, the algorithm finds these incompressible extents in units of the file compression block size.

Compressing files with the vxcompress command

You can compress files with the `vxcompress` command. The `vxcompress` command performs the following functions:

Functionality	Command syntax
Compress files or directory trees	**`vxcompress [-r] file_or_dir ...`**
Uncompress files or directory trees	**`vxcompress -u [-r] file_or_dir ...`**
Report the compression savings in a file or directory tree	**`vxcompress {-l|-L} [-r] file_or_dir ...`**
List the supported compression algorithms	**`vxcompress -a`**

See the `vxcompress`(1) manual page.

You can specify one or more filenames. If you specify the `-r` option, then you can specify directories, and the `vxcompress` command operates recursively on the directories.

You can specify the file compression algorithm and strength with the `vxcompress -t` command. The default algorithm is gzip, which is currently the only supported algorithm. The strength is a number from 1 to 9, with a default of 6. Strength 1 gives the fastest performance with least compression, while strength 9 gives the slowest performance with the greatest compression. For example, you specify strength 3 gzip compression as "gzip-3".

When reporting the compression details for a file, the `vxcompress -l` command or `vxcompress -L` command displays the following information:

- Compression algorithm
- Strength
- Compression block size
- % of file data saved by compression
- % of extents that are compressed
  This is the percentage of extents in the file that are compressed, without regard to the size of the extents. This percentage provides an idea of whether it is worthwhile to recompress the file. After recompression, the percentage is always 100%. However, shared extents are counted as uncompressed, and thus the percentage will be less than 100% if the file has shared extents.

If you attempt to compress a file with the `vxcompress` command and the extents have data that cannot be compressed, the command still marks the file as compressed and replaces the extents with compressed extent descriptors.

If you recompress a file, you do not need to specify any options with the `vxcompress` command. The command automatically uses the options that you used to compress the file previously.

The following command compresses the `file1` file, using the default algorithm and strength of gzip-6:

```
$ vxcompress file1
```

The following command recursively compresses all files below the `dir1` directory, using the gzip algorithm at the highest strength (9):

```
$ vxcompress -r -t gzip-9 dir1
```

The following command compresses the `file2` file and all files below the `dir2` directory, using the gzip algorithm at strength 3, while limiting the `vxcompress` command to a single thread and reducing the scheduling priority:

```
$ vxcompress -r -t gzip-3 file2 dir2
```

The following command displays the results of compressing the `file1` file in human-friendly units:

```
$ vxcompress -L file1
%Comp   Physical    Logical  %Exts  Alg-Str  BSize  Filename
  99%       1 KB     159 KB   100%   gzip-6  1024k  file1
```

The following command uncompresses the file1 file:

```
$ vxcompress -u file1
```

Interaction of compressed files and other commands

Table 31-1 describes how compressed files interact with other Storage Foundation commands and base operating system commands.

Table 31-1

Command	Interaction with compressed files
df	The df command shows the actual blocks in use by the file system. This number includes the compression savings, but the command does not display the savings explicitly. See the df(1) manual page.
du	The du command usually uses the block count and thus implicitly shows the results of compression, but the GNU du command has an option to use the file size instead, which is not changed by compression. See the du(1) manual page.
fsadm -S	The fsadm -S compressed command reports the space savings due to compressed files. See the fsadm_vxfs(1) manual page.
fsmap -p	The fsmap command can report information on compressed and uncompressed extents with the -p option. The reported logical size is the size of the uncompressed data, while the reported extent size is the size of the compressed data on disk. For compressed extents, the two sizes might differ. See the fsmap(1) manual page.
ls -l ls -s	The inode size reported by a stat call is the logical size, as shown by the ls -l command. This size is not affected by compression. On the other hand, the block count reflects the actual blocks used. As such, the ls -s command shows the result of compression. See the ls(1) manual page.

Table 31-1 *(continued)*

Command	Interaction with compressed files
vxdump	The vxdump command uncompresses compressed extents as it encounters them, meaning that compression is not preserved across a backup or restore operation.
vxquota	Your quota usage decreases based on the space saved due to compression. See the vxquota(1M) manual page.

Interaction of compressed files and other features

Table 31-2 describes how compressed files interact with other Storage Foundation features.

Table 31-2

Feature	Interaction with compressed files
Cross-Platform Data Sharing	If you convert a disk or file system from one platform that supports compression to a platform that does not support compression and the file system contains compressed files, the fscdsconv command displays a message that some files violate the CDS limits and prompts you to confirm if you want to continue the conversion. If you continue, the conversion completes successfully, but the compressed files will not be accessible on the new platform.
File Change Log	The File Change Log feature does not detect file compressions nor uncompressions.
Shared extents (FileSnap and deduplication)	Shared extents do not get compressed. Compressed files may be shared with the vxfilesnap command, though this results in a performance impact when accessing those files.
SmartTier	The SmartTier feature does not support compression. A placement policy cannot move existing compressed extents. Newly-allocated compressed extents follow the existing placement policy.

Table 31-2 *(continued)*

Feature	Interaction with compressed files
Space reservation (`setext -r`)	When a file is compressed, any space reserved via the `setext -r` command beyond the end-of-file is discarded, and is not restored when the file is uncompressed. The `setext -r` command cannot be used to reserve space for files that are compressed.
Storage Checkpoints	If a file system contains compressed files and you create a Storage Checkpoint of that file system, you can access those files normally through the Storage Checkpoint. However, you cannot compress nor uncompress a file that is already in a mounted Storage Checkpoint.

Interaction of compressed files and applications

In general, applications notice no difference between compressed and uncompressed files, although reads and writes to compressed extents are slower than reads and writes to uncompressed extents. When an application reads a compressed file, the file system does not perform its usual readahead to avoid the CPU load that this can require. However, when reading from the primary fileset, the file system uncompresses an entire compression block (default 1 MB) and leaves these pages in the page cache. Thus, sequential reads of the file usually only incur an extra cost when crossing a compression block boundary. The situation is different when reading from a file in a Storage Checkpoint; in this case, nothing goes into the page cache beyond the data actually requested. For optimal read performance of a compressed file accessed through a Storage Checkpoint, the application should use a read size that matches the compression block size.

When writing to compressed extents, ensure that you have sufficient disk space and disk quota limits for the new uncompressed extents since the write uncompresses the extents. If you do not have sufficient disk space, the write can fail with the ENOSPC error. If you do not have enough disk quota, the write can fail with the EDQUOT error.

An application that reads data from a compressed file and then copies the file elsewhere, such as `tar`, `cpio`, `cp`, or `vi`, does not preserve compression in the new data. The same is true of some backup programs.

Backup programs that read file data through the name space do not notice that the file is compressed. The backup program receives uncompressed data, and the compression is lost.

You cannot use the FlashBackup feature of NetBackup in conjunction with the file compression feature, because FlashBackup does not support disk layout Version 8 and 9.

Use cases for compressing files

The following list contains common use case categories:

- Compressed files and databases
- Compressing all files that meet the specified criteria

Compressed files and databases

Compressing files helps to reduce the storage cost in a database environment. For Oracle databases, compression provides an excellent value add to reduce storage cost for archived logs, partitioned tables, and infrequently accessed tablespaces and datafiles. The compression ratio of database files depends on the type of object stored in the datafiles. Oracle traditionally stores TABLES and INDEXES in datafiles, in which case the compression ratio depends on type of columns associated with the TABLE and the type of keys in the INDEXES. Oracle also has the ability to store unstructured data, such as XML, spreadsheets, MS Word documents, and pictures, within a TABLE via the Secured Files feature. These types of unstructured data are very good candidates for compression. You can achieve up to 90% compression for archived logs, and about 50% to 65% compression for Oracle datafiles and indexes.

Oracle database files can be compressed and uncompressed as needed while the database is active, although this can have a significant performance impact on the database. Other than reduced I/O response time, compression runs seamlessly while the Oracle database is online and actively doing transactions to the files. Compression works seamlessly with advanced I/O methods, such as direct I/O, asychronous I/O, concurrent I/O, ODM, and Cached ODM. Any updates and new inserts to the datafile result in uncompressing the portion of the file associated with the write. The queries get uncompressed data in memory and the file remains compressed.

Note: You can run the `vxcompress` command as a DBA user.

The following use cases apply to databases:

- Supported database versions and environment
- Compressing archive logs

- Compressing read-only tablespaces
- Compressing infrequently accessed table partitions
- Compressing infrequently accessed datafiles
- Best practices for compressing files in an Oracle database

Supported database versions and environment

Compression is supported in Storage Foundation (SF), Storage Foundation and High Availability (SFHA), Storage Foundation for Oracle RAC (SF Oracle RAC), and Storage Foundation Cluster File System High Availability (SFCFSHA). In a clustered environment, such as SF Oracle RAC and SFCFSHA, Veritas recommends that you compress files on a node that has minimal load. In a Fast Failover SFCFSHA environment, Veritas recommends that you compress files on a passive node where the database is offline.

Compressing archive logs

Archive logs are critical files required for database recovery. In a busy online transaction processing (OLTP) database, several gigabytes of archive logs are generated each day. Company guidelines often mandate preserving archive logs for several days. The Oracle archive logs are read-only files and are never updated after they are generated. During recovery, Oracle reads archive logs sequentially. As such, archive logs are very good candidates for compression, and archive logs are highly compressible.

The following example procedure compresses all archive logs that are older than a day.

To compress all archive logs that are older than a day

1 As an Oracle DBA, run the following query and get the archive log location:

```
SQL> select destination from v$archive_dest where status = 'VALID'
and valid_now = 'YES';
```

Assume `/oraarch/MYDB` is the archive log destination.

2 Compress all of the archive logs that are older than a day:

```
$ find /oraarch/MYDB -mtime +1 -exec /opt/VRTS/bin/vxcompress {} \;
```

You can run this step daily via a scheduler, such as `cron`.

Compressing read-only tablespaces

In a large database environment, it is a common practice to keep static tablespaces that do not have any changes in read-only mode. The primary purpose of read-only tablespaces is to eliminate the need to perform backup and recovery of large, static portions of a database. Read-only tablespaces also provide a way to protecting historical data so that users cannot modify it. Making a tablespace read-only prevents updates on all tables and objects residing in the tablespace, regardless of a user's update privilege level. These kinds of read-only tablespaces are excellent candidates for compression. In some cases such as month end reports, there may be large queries executed against these read-only tablespaces. To make the report run faster, you can uncompress the tablespace on demand before running the monthly reports.

In the following example, a sporting goods company has its inventory divided into two tablespaces: `winter_items` and `summer_items`. In the end of the Spring season, you can compress the `winter_item` tablespace and uncompress the `summer_item` tablespace. You can do the reverse actions at end of the Summer season. The following example procedure performs these tasks.

To compress and uncompress tablespaces depending on the season

1. Using SQL, get a list of files in each tablespace and store the result in the files `summer_files` and `winter_files`:

   ```
   SQL> select file_name from dba_data_files where
   tablespace_name = 'WINTER_ITEM';
   ```

   Store the result in the `winter_files` file.

   ```
   SQL> select  file_name from dba_data_files where
   tablespace_name = 'SUMMER_ITEM';
   ```

   Store the result in the `summer_files` file.

2. Compress the `winter_files` file:

   ```
   $ /opt/VRTS/bin/vxcompress `/bin/cat winter_files`
   ```

3. Uncompress the `summer_files` file:

   ```
   $ /opt/VRTS/bin/vxcompress -u `/bin/cat summer_files`
   ```

Compressing infrequently accessed table partitions

Partitioned tables is a frequently used feature for large Oracle databases. Table partitioning improves database queries and updates because partitioning helps parallelizing transactions that use Parallel Queries. Partitioning also makes maintenance of database easy and improves the availability of TABLES. If a partition is down, only the corresponding portion of the TABLE goes offline and the rest of the TABLE remains online. In a telecommunications environment, a common practice is to partition a 'call_details' table by month or quarter. The contents in the partition are less active as the partition gets older. The new records are added to a new partition, and previous quarter records do not get updated. Since telecommunications databases are generally very large, compressing last year's data provides great savings.

In the following example, assume that the table 'CALL_DETAIL' is partitioned by quarters, and the partition names are CALL_2010_Q1, CALL_2010_Q2, and CALL_2011_Q1, and so on. In the first Quarter of 2011, you can compress the CALL_2010_Q1 data.

To compress the CALL_2010_Q1 partition

1. Use SQL to retrieve the filenames belonging to the CALL_2010_Q1 partition:

   ```
   SQL> select tablespace_name from dba_tab_partitions
   where table_name = 'CALL_DETAIL' and partition_name = 'CALL_2010_Q1';
   ```

   Assume that the query returns "TBS_2010_Q1".

2. Store the names in the `my_compress_files` file:

   ```
   SQL> select file_name from dba_data_files where
   tablespace_name = 'TBS_2010_Q1';
   ```

   Store the result in the `my_compress_files` file.

3. Compress the files:

   ```
   $ /opt/VRTS/bin/vxcompress `/bin/cat my_compress_files`
   ```

Compressing infrequently accessed datafiles

Many customer databases do not use the Oracle partitioning feature. If partitioning is not used, then you can use Oracle catalog queries to identify datafiles that are not very active. Periodically, you can query the catalog tables and identify the least active datafiles and compress those files, as illustrated in the following example procedure.

To identify the least active datafiles and compress those files

1 Query v$filestat and identify the least active datafiles:

```
SQL> select  name, phyrds + phywrts 'TOT_IO' from v$datafile d
and v$filestat f where d.file# = f.file# order by TOT_IO;
```

2 Select files that have the least I/O activity from the report and compress those files:

```
$ /opt/VRTS/bin/vxcompress file1 file2 file3 ...
```

3 Periodically run the query again to ensure that the compressed files do not have increased I/O activity. If I/O activity increases, uncompress the files:

```
$ /opt/VRTS/bin/vxcompress -u file1 file2 file3 ...
```

Best practices for compressing files in an Oracle database

Even though an Oracle database runs without any errors when files are compressed, increased I/O to compressed files decreases database performance. Use the following guidelines for compressing Oracle datafiles:

- Do not compress database control files.
- Do not compress files belonging to TEMPORARY tablespaces.
- Do not compress files belonging to SYSTEM and SYSAUX tablespace.
- Monitor the I/O activity on compressed files periodically and uncompress the files if I/O activity increases.

Compressing all files that meet the specified criteria

You can find all files that meet the specified criteria and pipe the results to the `vxcompress` command to compress all of those files. The following example compresses all files in `/mnt` that have not been modified for more than 30 days:

```
$ find /mnt -mtime +30 | xargs /opt/VRTS/bin/vxcompress
```

Section 9

Administering and protecting storage

Chapter 32

Managing volumes and disk groups

This chapter includes the following topics:

Rules for determining the default disk group

You should use the `-g` option to specify a disk group to Veritas Volume Manager (VxVM) commands that accept this option. If you do not specify the disk group, VxVM applies rules in the following order until it determines a disk group name:

- Use the default disk group name that is specified by the environment variable VXVM_DEFAULTDG. This variable can also be set to one of the reserved system-wide disk group names: bootdg, defaultdg, or nodg.

  See "Displaying the system-wide boot disk group" on page 640.

  If the variable is undefined, the following rule is applied.
- Use the disk group that has been assigned to the system-wide default disk group alias, defaultdg.

  See "Displaying and specifying the system-wide default disk group" on page 640.

  If this alias is undefined, the following rule is applied.
- If the operation can be performed without requiring a disk group name (for example, an edit operation on disk access records), do so.

If none of these rules succeeds, the requested operation fails.

Warning: In releases of VxVM prior to 4.0, a subset of commands tried to determine the disk group by searching for the object name that was being operated upon by a command. This functionality is no longer supported. Scripts that rely on determining the disk group from an object name may fail.

Displaying the system-wide boot disk group

To display the currently defined system-wide boot disk group, use the following command:

```
# vxdg bootdg
```

See the vxdg(1M) manual page.

Displaying and specifying the system-wide default disk group

Veritas Volume Manager (VxVM) enables you to define a system-wide default disk group.

To display the currently defined system-wide default disk group, use the following command:

```
#  vxdg defaultdg
```

If a default disk group has not been defined, nodg is displayed.

See the vxdg(1M) manual page.

You can also use the following command to display the default disk group:

```
# vxprint -Gng defaultdg 2>/dev/null
```

In this case, if there is no default disk group, nothing is displayed.

See the `vxprint`(1M) manual page.

Use the following command to specify the name of the disk group that is aliased by `defaultdg`:

```
# vxdctl defaultdg diskgroup
```

Where *diskgroup* is one of the following:

- A specified disk group name.
  The specified disk group is not required to exist on the system.
- `bootdg`
  Sets the default disk group to be the same as the currently defined system-wide boot disk group.
- `nodg`
  Specifies that the default disk group is undefined.

See the `vxdctl`(1M) manual page.

Moving volumes or disks

This section describes moving volumes or disks.

Moving volumes from a VxVM disk

Before you disable or remove a disk, you can move the data from that disk to other disks on the system that have sufficient space.

To move volumes from a disk

1. From the `vxdiskadm` main menu, select `Move volumes from a disk .`
2. At the following prompt, enter the disk name of the disk whose volumes you want to move, as follows:

   ```
   Enter disk name [<disk>,list,q,?] mydg01
   ```

   You can now optionally specify a list of disks to which the volume(s) should be moved. At the prompt, do one of the following:

   - Press **Enter** to move the volumes onto available space in the disk group.
   - Specify the disks in the disk group that should be used, as follows:

   ```
   Enter disks [<disk ...>,list]
   VxVM NOTICE V-5-2-283 Requested operation is to move all
   ```

```
volumes from disk mydg01 in group mydg.

NOTE: This operation can take a long time to complete.

Continue with operation? [y,n,q,?] (default: y)
```

As the volumes are moved from the disk, the `vxdiskadm` program displays the status of the operation:

```
VxVM vxevac INFO V-5-2-24 Move volume voltest ...
```

When the volumes have all been moved, the `vxdiskadm` program displays the following success message:

```
VxVM INFO V-5-2-188 Evacuation of disk mydg02 is complete.
```

3 At the following prompt, indicate whether you want to move volumes from another disk (`y`) or return to the `vxdiskadm` main menu (`n`):

```
Move volumes from another disk? [y,n,q,?] (default: n)
```

Moving disks between disk groups

To move an unused disk between disk groups, remove the disk from one disk group and add it to the other. For example, to move the physical disk `sdc` (attached with the disk name `salesdg04`) from disk group `salesdg` and add it to disk group `mktdg`, use the following commands:

```
# vxdg -g salesdg rmdisk salesdg04
# vxdg -g mktdg adddisk mktdg02=sdc
```

Warning: This procedure does not save the configurations nor data on the disks.

You can also move a disk by using the `vxdiskadm` command. Select `Remove a disk` from the main menu, and then select `Add or initialize a disk`.

To move disks and preserve the data on these disks, along with VxVM objects, such as volumes:

See "Moving objects between disk groups" on page 650.

Reorganizing the contents of disk groups

There are several circumstances under which you might want to reorganize the contents of your existing disk groups:

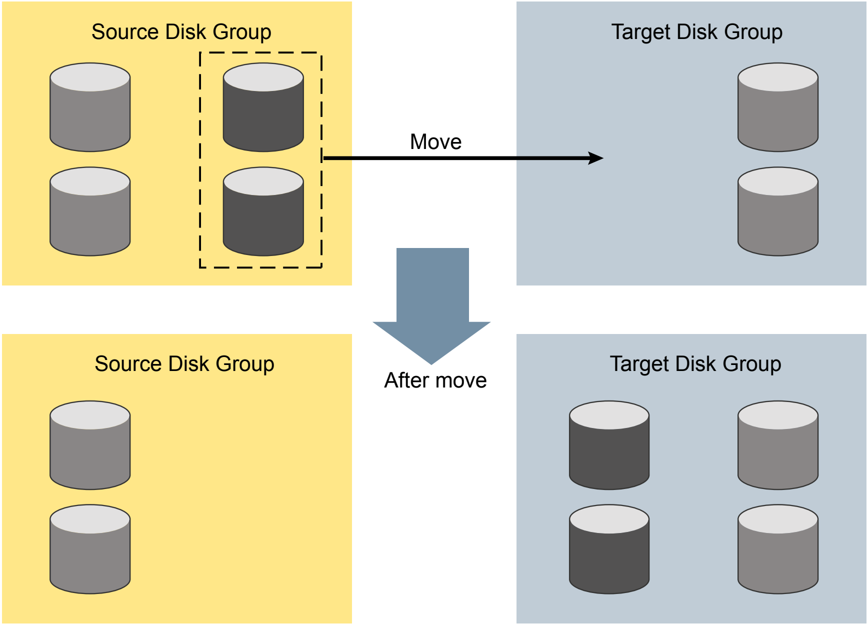
- To group volumes or disks differently as the needs of your organization change. For example, you might want to split disk groups to match the boundaries of separate departments, or to join disk groups when departments are merged.
- To isolate volumes or disks from a disk group, and process them independently on the same host or on a different host. This allows you to implement off-host processing solutions for the purposes of backup or decision support.
- To reduce the size of a disk group's configuration database in the event that its private region is nearly full. This is a much simpler solution than the alternative of trying to grow the private region.
- To perform online maintenance and upgrading of fault-tolerant systems that can be split into separate hosts for this purpose, and then rejoined.

Use the `vxdg` command to reorganize your disk groups.

The `vxdg` command provides the following operations for reorganizing disk groups:

- The `move` operation moves a self-contained set of VxVM objects between imported disk groups. This operation fails if it would remove all the disks from the source disk group. Volume states are preserved across the move.
  Figure 32-1 shows the `move` operation.

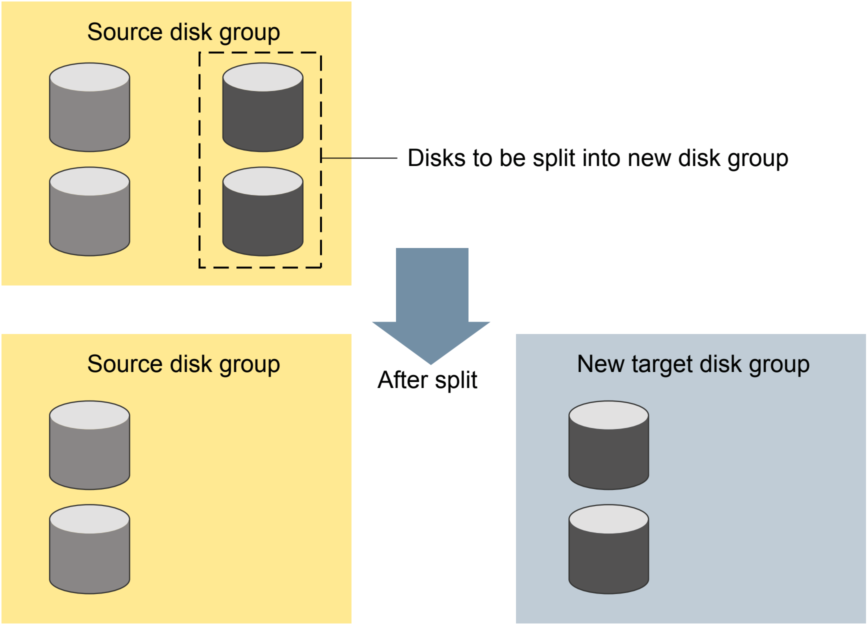
Figure 32-1 Disk group move operation



- The `split` operation removes a self-contained set of VxVM objects from an imported disk group, and moves them to a newly created target disk group. This operation fails if it would remove all the disks from the source disk group, or if an imported disk group exists with the same name as the target disk group.

Figure 32-2 shows the `split` operation.

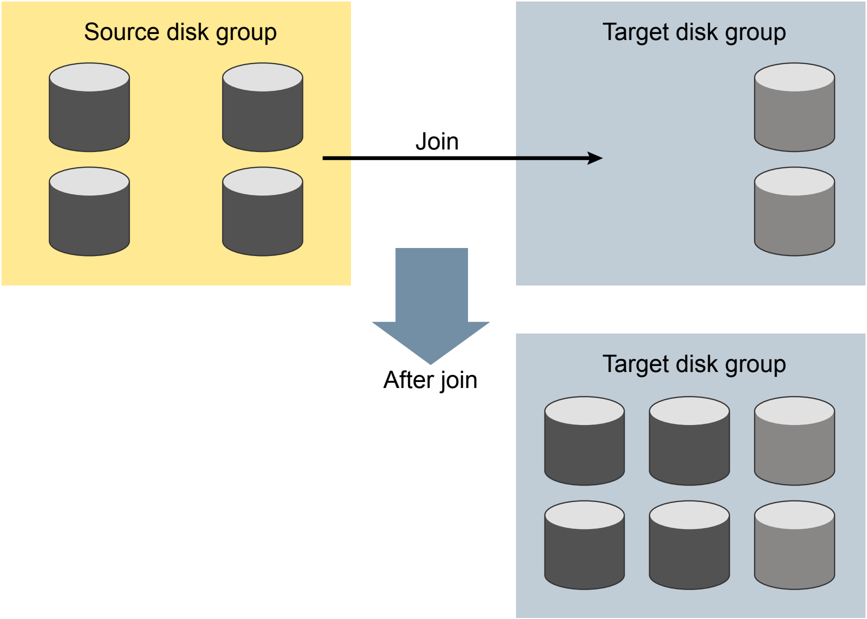
Figure 32-2 Disk group split operation



- The `join` operation removes all VxVM objects from an imported disk group and moves them to an imported target disk group. The source disk group is removed when the join is complete.

  Figure 32-3 shows the `join` operation.

Figure 32-3 Disk group join operation



These operations are performed on VxVM objects such as disks or top-level volumes, and include all component objects such as sub-volumes, plexes and subdisks. The objects to be moved must be self-contained, meaning that the disks that are moved must not contain any other objects that are not intended for the move.

For site-consistent disk groups, any of the move operations (move, split, and join) fail if the VxVM objects that are moved would not meet the site consistency conditions after the move. For example, a volume that is being moved may not have a plex on one of the sites configured in the target disk group. The volume would not meet the conditions for the allsites flag in the target disk group. Use the -f (force) option to enable the operation to succeed, by turning off the allsites flag on the object.

If you specify one or more disks to be moved, all VxVM objects on the disks are moved. You can use the `-o expand` option to ensure that `vxdg` moves all disks on which the specified objects are configured. Take care when doing this as the result may not always be what you expect. You can use the `listmove` operation with `vxdg` to help you establish what is the self-contained set of objects that corresponds to a specified set of objects.

Warning: Before moving volumes between disk groups, stop all applications that are accessing the volumes, and unmount all file systems that are configured on these volumes.

If the system crashes or a hardware subsystem fails, VxVM attempts to complete or reverse an incomplete disk group reconfiguration when the system is restarted or the hardware subsystem is repaired, depending on how far the reconfiguration had progressed. If one of the disk groups is no longer available because it has been imported by another host or because it no longer exists, you must recover the disk group manually.

See the *Veritas InfoScale Troubleshooting Guide*.

Limitations of disk group split and join

The disk group split and join feature has the following limitations:

- Disk groups involved in a move, split or join must be version 90 or greater.
  See "Upgrading the disk group version" on page 678.
- The reconfiguration must involve an integral number of physical disks.
- Objects to be moved must not contain open volumes.
- Disks cannot be moved between CDS and non-CDS compatible disk groups.
- By default, VxVM automatically recovers and starts the volumes following a disk group move, split or join. If you have turned off the automatic recovery feature, volumes are disabled after a move, split, or join. Use the `vxrecover -m` and `vxvol startall` commands to recover and restart the volumes.
  See "Setting the automatic recovery of volumes" on page 686.
- Data change objects (DCOs) and snap objects that have been dissociated by Persistent FastResync cannot be moved between disk groups.
- Veritas Volume Replicator (VVR) objects cannot be moved between disk groups.
- For a disk group move to succeed, the source disk group must contain at least one disk that can store copies of the configuration database after the move.
- For a disk group split to succeed, both the source and target disk groups must contain at least one disk that can store copies of the configuration database after the split.
- For a disk group move or join to succeed, the configuration database in the target disk group must be able to accommodate information about all the objects in the enlarged disk group.

- Splitting or moving a volume into a different disk group changes the volume's record ID.
- The operation can only be performed on the master node of a cluster if either the source disk group or the target disk group is shared.
- In a cluster environment, disk groups involved in a move or join must both be private or must both be shared.
- If a cache object or volume set that is to be split or moved uses ISP volumes, the storage pool that contains these volumes must also be specified.

Listing objects potentially affected by a move

To display the VxVM objects that would be moved for a specified list of objects, use the following command:

```
# vxdg [-o expand] listmove sourcedg targetdg object ...
```

The following example lists the objects that would be affected by moving volume `vol1` from disk group `mydg` to `newdg`:

```
# vxdg listmove mydg newdg vol1
mydg01 sda mydg05 sde vol1 vol1-01 vol1-02 mydg01-01 mydg05-01
```

However, the following command produces an error because only a part of the volume `vol1` is configured on the disk `mydg01`:

```
# vxdg listmove mydg newdg mydg01
VxVM vxdg ERROR V-5-2-4597 vxdg listmove mydg newdg failed
VxVM vxdg ERROR V-5-2-3091 mydg05 : Disk not moving, but
subdisks on it are
```

Specifying the `-o expand` option, as shown below, ensures that the list of objects to be moved includes the other disks (in this case, `mydg05`) that are configured in `vol1`:

```
# vxdg -o expand listmove mydg newdg mydg01
mydg01 sda mydg05 sde vol1 vol1-01 vol1-02 mydg01-01
mydg05-01
```

Moving DCO volumes between disk groups

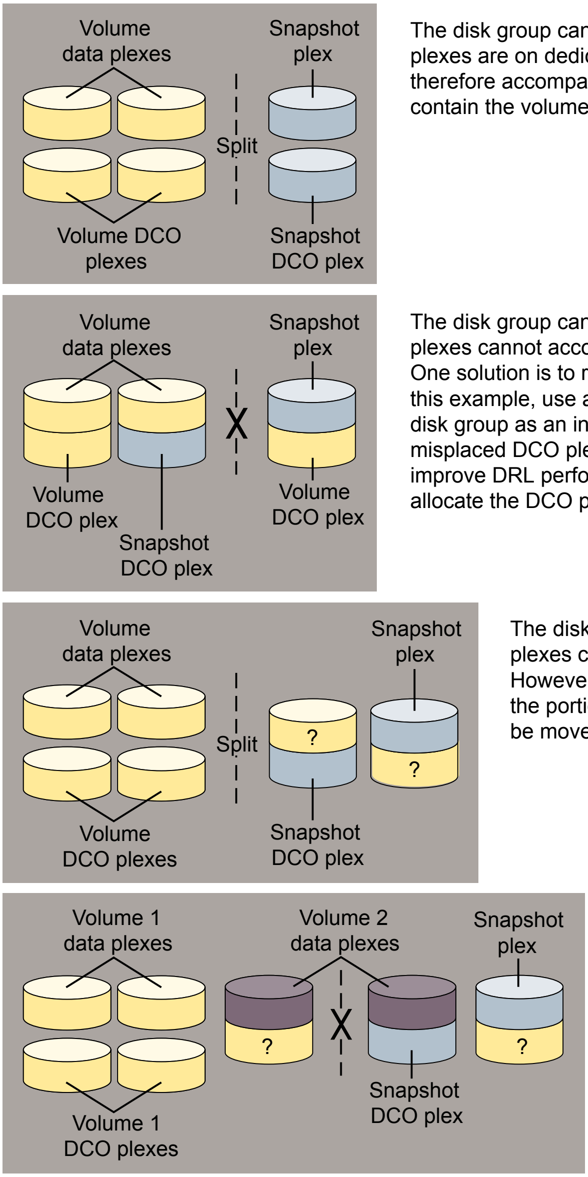
When you move the parent volume (such as a snapshot volume) to a different disk group, its DCO volume must accompany it. If you use the `vxassist addlog`, `vxmake` or `vxdco` commands to set up a DCO for a volume, you must ensure that the disks that contain the plexes of the DCO volume accompany their parent volume during the move. You can use the `vxprint` command on a volume to examine the configuration of its associated DCO volume.

If you use the `vxassist` command to create both a volume and its DCO, or the `vxsnap prepare` command to add a DCO to a volume, the DCO plexes are automatically placed on different disks from the data plexes of the parent volume. In previous releases, version 0 DCO plexes were placed on the same disks as the data plexes for convenience when performing disk group split and move operations. As version 30 DCOs support dirty region logging (DRL) in addition to Persistent FastResync, it is preferable for the DCO plexes to be separated from the data plexes. This improves the performance of I/O from/to the volume, and provides resilience for the DRL logs.

Figure 32-4 shows some instances in which it is not be possible to split a disk group because of the location of the DCO plexes on the disks of the disk group.

See "Volume snapshots" on page 78.

Figure 32-4 Examples of disk groups that can and cannot be split



The disk group can be split as the DCO plexes are on dedicated disks, and can therefore accompany the disks that contain the volume data

The disk group cannot be split as the DCO plexes cannot accompany their volumes. One solution is to relocate the DCO plexes. In this example, use an additional disk in the disk group as an intermediary to swap the misplaced DCO plexes. Alternatively, to improve DRL performance and resilience, allocate the DCO plexes to dedicated disks.

The disk group can be split as the DCO plexes can accompany their volumes. However, you may not wish the data in the portions of the disks marked “?” to be moved as well.

The disk group cannot be split as this would separate the disks containing Volume 2’s data plexes. Possible solutions are to relocate the snapshot DCO plex to the snapshot plex disk, or to another suitable disk that can be moved.

Moving objects between disk groups

To move a self-contained set of VxVM objects from an imported source disk group to an imported target disk group, use the following command:

```
# vxdg [-o expand] [-o override|verify] move sourcedg targetdg \
  object ...
```

The `-o expand` option ensures that the objects that are actually moved include all other disks containing subdisks that are associated with the specified objects or with objects that they contain.

The default behavior of `vxdg` when moving licensed disks in an EMC array is to perform an EMC disk compatibility check for each disk involved in the move. If the compatibility checks succeed, the move takes place. `vxdg` then checks again to ensure that the configuration has not changed since it performed the compatibility check. If the configuration has changed, `vxdg` attempts to perform the entire move again.

Note: You should only use the `-o override` and `-o verify` options if you are using an EMC array with a valid timefinder license. If you specify one of these options and do not meet the array and license requirements, a warning message is displayed and the operation is ignored.

The `-o override` option enables the move to take place without any EMC checking.

The `-o verify` option returns the access names of the disks that would be moved but does not perform the move.

The following output from `vxprint` shows the contents of disk groups `rootdg` and `mydg`.

The output includes two utility fields, `TUTIL0` and `PUTIL0`. VxVM creates these fields to manage objects and communications between different commands and Veritas InfoScale products. The `TUTIL0` values are temporary; they are not maintained on reboot. The `PUTIL0` values are persistent; they are maintained on reboot.

```
# vxprint
Disk group: rootdg
TY NAME        ASSOC      KSTATE   LENGTH     PLOFFS   STATE   TUTIL0   PUTIL0
dg rootdg      rootdg     -        -          -        -       -        -
dm rootdg02    sdb        -        17678493   -        -       -        -
dm rootdg03    sdc        -        17678493   -        -       -        -
dm rootdg04    csdd       -        17678493   -        -       -        -
dm rootdg06    sdf        -        17678493   -        -       -        -

Disk group: mydg
TY NAME        ASSOC      KSTATE   LENGTH     PLOFFS   STATE   TUTIL0   PUTIL0
dg mydg        mydg       -        -          -        -       -        -
dm mydg01      sda        -        17678493   -        -       -        -
dm mydg05      sde        -        17678493   -        -       -        -
dm mydg07      sdg        -        17678493   -        -       -        -
dm mydg08      sdh        -        17678493   -        -       -        -
v  vol1        fsgen      ENABLED  2048       -        ACTIVE  -        -
pl  vol1-01    vol1       ENABLED  3591       -        ACTIVE  -        -
sd mydg01-01   vol1-01    ENABLED  3591       0        -       -        -
pl vol1-02     vol1       ENABLED  3591       -        ACTIVE  -        -
sd mydg05-01   vol1-02    ENABLED  3591       0        -       -        -
```

The following command moves the self-contained set of objects implied by specifying disk `mydg01` from disk group `mydg` to `rootdg`:

```
# vxdg -o expand move mydg rootdg mydg01
```

By default, VxVM automatically recovers and starts the volumes following a disk group move. If you have turned off the automatic recovery feature, volumes are disabled after a move. Use the following commands to recover and restart the volumes in the target disk group:

```
# vxrecover -g targetdg -m [volume ...]
# vxvol -g targetdg startall
```

The output from `vxprint` after the move shows that not only `mydg01` but also volume `vol1` and `mydg05` have moved to `rootdg`, leaving only `mydg07` and `mydg08` in disk group `mydg`:

```
# vxprint
Disk group: rootdg
TY NAME        ASSOC      KSTATE    LENGTH     PLOFFS   STATE   TUTIL0   PUTIL0
dg rootdg      rootdg     -         -          -        -       -        -
dm mydg01      sda        -         17678493   -        -       -        -
dm rootdg02    sdb        -         17678493   -        -       -        -
dm rootdg03    sdc        -         17678493   -        -       -        -
dm rootdg04    sdd        -         17678493   -        -       -        -
dm mydg05      sde        -         17678493   -        -       -        -
dm rootdg06    sdf        -         17678493   -        -       -        -
v  vol1        fsgen      ENABLED   2048       -        ACTIVE  -        -
pl  vol1-01    vol1       ENABLED   3591       -        ACTIVE  -        -
sd mydg01-01   vol1-01    ENABLED   3591       0        -       -        -
pl vol1-02     vol1       ENABLED   3591       -        ACTIVE  -        -
sd mydg05-01   vol1-02    ENABLED   3591       0        -       -        -

Disk group: mydg
TY NAME        ASSOC      KSTATE    LENGTH     PLOFFS   STATE   TUTIL0   PUTIL0
dg mydg        mydg       -         -          -        -       -        -
```

```
dm mydg07      sdg        -          17678493   -        -       -        -
dm mydg08      sdh        -          17678493   -        -       -        -
```

If the volume is encrypted with a password or passphrase, you will be prompted to enter it when VxVM restarts the volumes.

The following commands would also achieve the same result:

```
# vxdg move mydg rootdg mydg01 mydg05
# vxdg move mydg rootdg vol1
```

Splitting disk groups

To remove a self-contained set of VxVM objects from an imported source disk group to a new target disk group, use the following command:

```
# vxdg [-o expand] [-o override|verify] split sourcedg targetdg \
  object ...
```

See “Moving objects between disk groups” on page 650.

The following output from `vxprint` shows the contents of disk group `rootdg`.

The output includes two utility fields, `TUTIL0` and `PUTIL0`.. VxVM creates these fields to manage objects and communications between different commands and Veritas InfoScale products. The `TUTIL0` values are temporary; they are not maintained on reboot. The `PUTIL0` values are persistent; they are maintained on reboot.

```
# vxprint
Disk group: rootdg
TY NAME        ASSOC      KSTATE     LENGTH     PLOFFS    STATE    TUTIL0   PUTIL0
dg rootdg      rootdg     -          -          -         -        -        -
dm rootdg01    sda        -          17678493   -         -        -        -
dm rootdg02    sdb        -          17678493   -         -        -        -
dm rootdg03    sdc        -          17678493   -         -        -        -
dm rootdg04    sdd        -          17678493   -         -        -        -
dm rootdg05    sde        -          17678493   -         -        -        -
dm rootdg06    sdf        -          17678493   -         -        -        -
dm rootdg07    sdg        -          17678493   -         -        -        -
dm rootdg08    sdh        -          17678493   -         -        -        -
v  vol1        fsgen      ENABLED    2048       -         ACTIVE   -        -
pl  vol1-01    vol1       ENABLED    3591       -         ACTIVE   -        -
sd rootdg01-01 vol1-01    ENABLED    3591       0         -        -        -
pl vol1-02     vol1       ENABLED    3591       -         ACTIVE   -        -
sd rootdg05-01 vol1-02    ENABLED    3591       0         -        -        -
```

The following command removes disks `rootdg07` and `rootdg08` from `rootdg` to form a new disk group, `mydg`:

```
# vxdg -o expand split rootdg mydg rootdg07 rootdg08
```

By default, VxVM automatically recovers and starts the volumes following a disk group split. If you have turned off the automatic recovery feature, volumes are disabled after a split. Use the following commands to recover and restart the volumes in the target disk group:

```
# vxrecover -g targetdg -m [volume ...]
# vxvol -g targetdg startall
```

If the volume is encrypted with a password or passphrase, you will be prompted to enter it when VxVM restarts the volumes.

The output from `vxprint` after the split shows the new disk group, `mydg`:

```
# vxprint
Disk group: rootdg
TY NAME        ASSOC     KSTATE     LENGTH     PLOFFS    STATE    TUTIL0     PUTIL0
dg rootdg      rootdg    -          -          -         -        -          -
dm rootdg01    sda       -          17678493   -         -        -          -
dm rootdg02    sdb       -          17678493   -         -        -          -
dm rootdg03    sdc       -          17678493   -         -        -          -
dm rootdg04    sdd       -          17678493   -         -        -          -
dm rootdg05    sde       -          17678493   -         -        -          -
dm rootdg06    sdf       -          17678493   -         -        -          -
v vol1         fsgen     ENABLED    2048       -         ACTIVE   -          -
pl vol1-01     vol1      ENABLED    3591       -         ACTIVE   -          -
sd rootdg01-01 vol1-01   ENABLED    3591       0         -        -          -
pl vol1-02     vol1      ENABLED    3591       -         ACTIVE   -          -
sd rootdg05-01 vol1-02   ENABLED    3591       0         -        -          -
Disk group: mydg
TY NAME        ASSOC     KSTATE     LENGTH     PLOFFS    STATE    TUTIL0     PUTIL0
dg mydg        mydg      -          -          -         -        -          -
dm rootdg07    sdg       -          17678493   -         -        -          -
dm rootdg08    sdh       -          17678493   -         -        -          -
```

Joining disk groups

To remove all VxVM objects from an imported source disk group to an imported target disk group, use the following command:

```
# vxdg [-o override|verify] join sourcedg targetdg
```

See “Moving objects between disk groups” on page 650.

Note: You cannot specify `rootdg` as the source disk group for a `join` operation.

The following output from `vxprint` shows the contents of the disk groups `rootdg` and `mydg`.

The output includes two utility fields, `TUTIL0` and `PUTIL0`.. VxVM creates these fields to manage objects and communications between different commands and Veritas InfoScale products. The `TUTIL0` values are temporary; they are not maintained on reboot. The `PUTIL0` values are persistent; they are maintained on reboot.

```
# vxprint
Disk group: rootdg
TY NAME         ASSOC       KSTATE     LENGTH     PLOFFS    STATE    TUTIL0    PUTIL0
dg rootdg       rootdg      -          -          -         -        -         -
dm rootdg01     sda         -          17678493   -         -        -         -
dm rootdg02     sdb         -          17678493   -         -        -         -
dm rootdg03     sdc         -          17678493   -         -        -         -
dm rootdg04     sdd         -          17678493   -         -        -         -
dm rootdg07     sdg         -          17678493   -         -        -         -
dm rootdg08     sdh         -          17678493   -         -        -         -

Disk group: mydg
TY NAME         ASSOC       KSTATE     LENGTH     PLOFFS    STATE    TUTIL0    PUTIL0
dg mydg         mydg        -          -          -         -        -         -
dm mydg05       sde         -          17678493   -         -        -         -
dm mydg06       sdf         -          17678493   -         -        -         -
v  vol1         fsgen       ENABLED    2048       -         ACTIVE   -         -
pl  vol1-01     vol1        ENABLED    3591       -         ACTIVE   -         -
sd mydg01-01    vol1-01     ENABLED    3591       0         -        -         -
pl vol1-02      vol1        ENABLED    3591       -         ACTIVE   -         -
sd mydg05-01    vol1-02     ENABLED    3591       0         -        -         -
```

The following command joins disk group `mydg` to `rootdg`:

```
# vxdg join mydg rootdg
```

By default, VxVM automatically recovers and starts the volumes following a disk group join. If you have turned off the automatic recovery feature, volumes are disabled after a join. Use the following commands to recover and restart the volumes in the target disk group:

```
# vxrecover -g targetdg -m [volume ...]
# vxvol -g targetdg startall
```

If the volume is encrypted with a password or passphrase, you will be prompted to enter it when VxVM restarts the volumes.

The output from `vxprint` after the join shows that disk group `mydg` has been removed:

```
# vxprint
Disk group: rootdg
TY NAME        ASSOC       KSTATE     LENGTH      PLOFFS     STATE    TUTIL0    PUTIL0
dg rootdg      rootdg      -          -           -          -        -         -
dm mydg01      sda         -          17678493    -          -        -         -
dm rootdg02    sdb         -          17678493    -          -        -         -
dm rootdg03    sdc         -          17678493    -          -        -         -
dm rootdg04    sdd         -          17678493    -          -        -         -
dm mydg05      sde         -          17678493    -          -        -         -
dm rootdg06    sdf         -          17678493    -          -        -         -
dm rootdg07    sdg         -          17678493    -          -        -         -
dm rootdg08    sdh         -          17678493    -          -        -         -
v  vol1        fsgen       ENABLED    2048        -          ACTIVE   -         -
pl  vol1-01    vol1        ENABLED    3591        -          ACTIVE   -         -
sd mydg01-01   vol1-01     ENABLED    3591        0          -        -         -
pl vol1-02     vol1        ENABLED    3591        -          ACTIVE   -         -
sd mydg05-01   vol1-02     ENABLED    3591        0          -        -         -
```

Monitoring and controlling tasks

The VxVM task monitor tracks the progress of system recovery by monitoring task creation, maintenance, and completion. The task monitor lets you monitor task progress and modify characteristics of tasks, such as pausing and recovery rate (for example, to reduce the impact on system performance).

Note: VxVM supports this feature only for private disk groups, not for shared disk groups in a CVM environment.

The tasks that are created on the system are logged for diagnostic purposes in the `tasklog` file in the `/etc/vx/log/` directory. This file logs an entry for all task-related operations (creation, completion, pause, resume, and abort). The following is a sample log entry:

```
# 159571211, 16905,  Thu Aug 21 02:54:18 2014
  184     -   SNAPSYNC  Starting 00.00% 0/65536/0 SNAPSYNC full1-v1 vvrdg
```

Specifying task tags

Every task is given a unique task identifier. This is a numeric identifier for the task that can be specified to the `vxtask` utility to specifically identify a single task. Several VxVM utilities also provide a `-t` option to specify an alphanumeric tag of up to 16 characters in length. This allows you to group several tasks by associating them with the same tag.

The following utilities accept the `-t` option:

- `vxassist`
- `vxevac`
- `vxmirror`
- `vxplex`
- `vxrecover`
- `vxrelayout`
- `vxresize`
- `vxsd`
- `vxvol`

For example, to execute a `vxrecover` command and track the resulting tasks as a group with the task tag `myrecovery`, use the following command:

```
# vxrecover -g mydg -t myrecovery -b mydg05
```

To track the resulting tasks, use the following command:

```
# vxtask monitor myrecovery
```

Any tasks started by the utilities invoked by `vxrecover` also inherit its task ID and task tag, establishing a parent-child task relationship.

For more information about the utilities that support task tagging, see their respective manual pages.

Managing tasks with vxtask

You can use the `vxtask` command to administer operations on VxVM tasks. Operations include listing tasks, modifying the task state (pausing, resuming, aborting) and modifying the task's progress rate.

VxVM tasks represent long-term operations in progress on the system. Every task gives information on the time the operation started, the size and progress of the operation, and the state and rate of progress of the operation. You can change the state of a task, giving coarse-grained control over the progress of the operation. For those operations that support it, you can change the rate of progress of the task, giving more fine-grained control over the task.

New tasks take time to be set up, and so may not be immediately available for use after a command is invoked. Any script that operates on tasks may need to poll for the existence of a new task.

See the `vxtask`(1M) manual page.

vxtask operations

The `vxtask` command supports the following operations:

`abort`

Stops the specified task. In most cases, the operations “back out” as if an I/O error occurred, reversing what has been done so far to the largest extent possible.

`list`

Displays a one-line summary for each task running on the system. The `-l` option prints tasks in long format. The `-h` option prints tasks hierarchically, with child tasks following the parent tasks. By default, all tasks running on the system are printed. If you include a `taskid` argument, the output is limited to those tasks whose `taskid` or task tag match `taskid`. The remaining arguments filter tasks and limit which ones are listed.

In this release, the `vxtask list` command supports SmartMove and thin reclamation operation.

- If you use SmartMove to resync or sync the volume, plex, or subdisk, the `vxtask list` displays whether the operations is using SmartMove or not.
- In a LUN level reclamation, the `vxtask list` command provides information on the amount of the reclaim performed on each LUN.
- The `init=zero` on the thin volume may trigger the reclaim on the thin volume and the progress is seen in the `vxtask list` command.

`monitor` Prints information continuously about a task or group of tasks as task information changes. This lets you track task progress. Specifying `-l` prints a long listing. By default, one-line listings are printed. In addition to printing task information when a task state changes, output is also generated when the task completes. When this occurs, the state of the task is printed as `EXITED`.

`pause` Pauses a running task, causing it to suspend operation.

`resume` Causes a paused task to continue operation.

`set` Changes a task's modifiable parameters. Currently, there is only one modifiable parameter, *`slow[=iodelay]`* , which can be used to reduce the impact that copy operations have on system performance. If you specify `slow`, this introduces a delay between such operations with a default value for `iodelay` of 250 milliseconds. The larger `iodelay` value you specify, the slower the task progresses and the fewer system resources that it consumes in a given time. (The `vxplex`, `vxvol` and `vxrecover` commands also accept the `slow` attribute.)

Using the vxtask command

To list all tasks running on the system, use the following command:

```
# vxtask list
```

To print tasks hierarchically, with child tasks following the parent tasks, specify the `-h` option, as follows:

```
# vxtask -h list
```

To trace all paused tasks in the disk group `mydg`, as well as any tasks with the tag `sysstart`, use the following command:

```
# vxtask -g mydg -p -i sysstart list
```

To list all paused tasks, use the `vxtask -p list` command. To continue execution (the task may be specified by its ID or by its tag), use `vxtask resume` :

```
# vxtask -p list
# vxtask resume 167
```

To monitor all tasks with the tag `myoperation`, use the following command:

```
# vxtask monitor myoperation
```

To cause all tasks tagged with `recovall` to exit, use the following command:

```
# vxtask abort recovall
```

This command causes VxVM to try to reverse the progress of the operation so far. For example, aborting an Online Relayout results in VxVM returning the volume to its original layout.

See "Controlling the progress of a relayout" on page 666.

Using vxnotify to monitor configuration changes

You can use the `vxnotify` utility to display events relating to disk and configuration changes that are managed by the `vxconfigd` configuration daemon. If `vxnotify` is running on a system where the VxVM clustering feature is active, it displays events that are related to changes in the cluster state of the system on which it is running. The `vxnotify` utility displays the requested event types until you kill it, until it has received a specified number of events, or until a specified period of time has elapsed.

Examples of configuration events that can be detected include disabling and enabling of controllers, paths and DMP nodes, RAID-5 volumes entering degraded mode, detachment of disks, plexes and volumes, and nodes joining and leaving a cluster.

For example, the following `vxnotify` command displays information about all disk, plex, and volume detachments as they occur:

```
# vxnotify -f
```

The following command provides information about cluster configuration changes, including the import and deport of shared disk groups:

```
# vxnotify -s -i
```

See the `vxnotify`(1M) manual page.

Performing online relayout

You can use the `vxassist relayout` command to reconfigure the layout of a volume without taking it offline. The general form of this command is as follows:

```
# vxassist [-b] [-g diskgroup] relayout volume [layout=layout] \
  [relayout_options]
```

If you specify the `-b` option, relayout of the volume is a background task.

Note: If a relayout operation is done on a volume that has SmartIO VxVM caching enabled, the contents of the cache for the volume may be invalidated.

The following destination layout configurations are supported.

`concat-mirror`	Concatenated-mirror
`concat`	Concatenated
`nomirror`	Concatenated
`nostripe`	Concatenated
`raid5`	RAID-5 (not supported for shared disk groups)
`span`	Concatenated
`stripe`	Striped

See "Permitted relayout transformations" on page 661.

For example, the following command changes the concatenated volume `vol02`, in disk group `mydg`, to a striped volume. By default, the striped volume has 2 columns and a 64 KB striped unit size.:

```
# vxassist -g mydg relayout vol02 layout=stripe
```

Sometimes, you may need to perform a relayout on a plex rather than on a volume.

See "Specifying a plex for relayout" on page 665.

Permitted relayout transformations

Table 32-1 shows the supported relayout transformations for concatenated volumes.

Table 32-1 Supported relayout transformations for concatenated volumes

Relayout to	From concat
concat	No.
concat-mirror	No. Add a mirror, and then use `vxassist convert` instead.
mirror-concat	No. Add a mirror instead.
mirror-stripe	No. Use `vxassist convert` after relayout to the striped-mirror volume instead.
raid5	Yes. The stripe width and number of columns may be defined.

Table 32-1 Supported relayout transformations for concatenated volumes *(continued)*

Relayout to	From concat
stripe	Yes. The stripe width and number of columns may be defined.
stripe-mirror	Yes. The stripe width and number of columns may be defined.

Table 32-2 shows the supported relayout transformations for concatenated-mirror volumes.

Table 32-2 Supported relayout transformations for concatenated-mirror volumes

Relayout to	From concat-mirror
concat	No. Use `vxassist convert`, and then remove the unwanted mirrors from the resulting mirrored-concatenated volume instead.
concat-mirror	No.
mirror-concat	No. Use `vxassist convert` instead.
mirror-stripe	No. Use `vxassist convert` after relayout to the striped-mirror volume instead.
raid5	Yes.
stripe	Yes. This relayout removes a mirror and adds striping. The stripe width and number of columns may be defined.
stripe-mirror	Yes. The stripe width and number of columns may be defined.

Table 32-3 shows the supported relayout transformations for RAID-5 volumes.

Table 32-3 Supported relayout transformations for RAID-5 volumes

Relayout to	From RAID-5
concat	Yes.
concat-mirror	Yes.
mirror-concat	No. Use `vxassist convert` after relayout to the concatenated-mirror volume instead.
mirror-stripe	No. Use `vxassist convert` after relayout to the striped-mirror volume instead.

Table 32-3 Supported relayout transformations for RAID-5 volumes *(continued)*

Relayout to	From RAID-5
raid5	Yes. The stripe width and number of columns may be changed.
stripe	Yes. The stripe width or number of columns must be changed.
stripe-mirror	Yes. The stripe width or number of columns must be changed. Otherwise, use `vxassist convert`.

Table 32-4 shows the supported relayout transformations for mirror-concatenated volumes.

Table 32-4 Supported relayout transformations for mirrored-concatenated volumes

Relayout to	From mirror-concat
concat	No. Remove the unwanted mirrors instead.
concat-mirror	No. Use `vxassist convert` instead.
mirror-concat	No.
mirror-stripe	No. Use `vxassist convert` after relayout to the striped-mirror volume instead.
raid5	Yes. The stripe width and number of columns may be defined. Choose a plex in the existing mirrored volume on which to perform the relayout. The other plexes are removed at the end of the relayout operation.
stripe	Yes.
stripe-mirror	Yes.

Table 32-5 shows the supported relayout transformations for mirrored-stripe volumes.

Table 32-5 Supported relayout transformations for mirrored-stripe volumes

Relayout to	From mirror-stripe
concat	Yes.
concat-mirror	Yes.
mirror-concat	No. Use `vxassist convert` after relayout to the concatenated-mirror volume instead.

Table 32-5 Supported relayout transformations for mirrored-stripe volumes *(continued)*

Relayout to	From mirror-stripe
mirror-stripe	No. Use `vxassist convert` after relayout to the striped-mirror volume instead.
raid5	Yes. The stripe width and number of columns may be changed.
stripe	Yes. The stripe width or number of columns must be changed.
stripe-mirror	Yes. The stripe width or number of columns must be changed. Otherwise, use `vxassist convert`.

Table 32-6 shows the supported relayout transformations for unmirrored stripe and layered striped-mirror volumes.

Table 32-6 Supported relayout transformations for unmirrored stripe and layered striped-mirror volumes

Relayout to	From stripe or stripe-mirror
concat	Yes.
concat-mirror	Yes.
mirror-concat	No. Use `vxassist convert` after relayout to the concatenated-mirror volume instead.
mirror-stripe	No. Use `vxassist convert` after relayout to the striped-mirror volume instead.
raid5	Yes. The stripe width and number of columns may be changed.
stripe	Yes. The stripe width or number of columns must be changed.
stripe-mirror	Yes. The stripe width or number of columns must be changed.

Specifying a non-default layout

You can specify one or more of the following relayout options to change the default layout configuration:

`ncol=`*`number`*	Specifies the number of columns.
`ncol=+`*`number`*	Specifies the number of columns to add.
`ncol=-`*`number`*	Specifies the number of columns to remove.

`stripeunit=`*`size`*	Specifies the stripe width.

The following examples use `vxassist` to change the stripe width and number of columns for a striped volume in the disk group `dbasedg`:

```
# vxassist -g dbasedg relayout vol03 stripeunit=64k ncol=6
# vxassist -g dbasedg relayout vol03 ncol=+2
# vxassist -g dbasedg relayout vol03 stripeunit=128k
```

The following example changes a concatenated volume to a RAID-5 volume with four columns:

```
# vxassist -g dbasedg relayout vol04 layout=raid5 ncol=4
```

Specifying a plex for relayout

If you have enough disks and space in the disk group, you can change any layout to RAID-5 . To convert a mirrored volume to RAID-5, you must specify which plex is to be converted. When the conversion finishes, all other plexes are removed, releasing their space for other purposes. If you convert a mirrored volume to a layout other than RAID-5, the unconverted plexes are not removed. Specify the plex to be converted by naming it in place of a volume as follows:

```
# vxassist [-g diskgroup] relayout plex [layout=layout] \
  [relayout_options]
```

Tagging a relayout operation

To control the progress of a relayout operation, for example to pause or reverse it, use the `-t` option to `vxassist` to specify a task tag for the operation. For example, the following relayout is performed as a background task and has the tag `myconv`:

```
# vxassist -b -g dbasedg -t myconv relayout vol04 layout=raid5 \
  ncol=4
```

See “Viewing the status of a relayout” on page 665.

See “Controlling the progress of a relayout” on page 666.

Viewing the status of a relayout

Online relayout operations take time to perform. You can use the `vxrelayout` command to obtain information about the status of a relayout operation. For example, the following command:

```
# vxrelayout -g mydg status vol04
```

might display output similar to the following:

```
STRIPED, columns=5, stwidth=128--> STRIPED, columns=6,
stwidth=128
Relayout running, 68.58% completed.
```

In this example, the reconfiguration is in progress for a striped volume from 5 to 6 columns, and is over two-thirds complete.

See the `vxrelayout`(1M) manual page.

If you specify a task tag to `vxassist` when you start the relayout, you can use this tag with the `vxtask` command to monitor the progress of the relayout. For example, to monitor the task that is tagged as `myconv`, enter the following:

```
# vxtask monitor myconv
```

Controlling the progress of a relayout

You can use the `vxtask` command to stop (`pause`) the relayout temporarily, or to cancel it (`abort`). If you specify a task tag to `vxassist` when you start the relayout, you can use this tag to specify the task to `vxtask`. For example, to pause the relayout operation that is tagged as `myconv`, enter:

```
# vxtask pause myconv
```

To resume the operation, use the `vxtask` command as follows:

```
# vxtask resume myconv
```

For relayout operations that have not been stopped using the `vxtask pause` command (for example, the `vxtask abort` command was used to stop the task, the transformation process died, or there was an I/O failure), resume the relayout by specifying the `start` keyword to `vxrelayout`, as follows:

```
# vxrelayout -g mydg -o bg start vol04
```

If you use the `vxrelayout start` command to restart a relayout that you previously suspended using the `vxtask pause` command, a new untagged task is created to complete the operation. You cannot then use the original task tag to control the relayout.

The `-o bg` option restarts the relayout in the background. You can also specify the `slow` and `iosize` option modifiers to control the speed of the relayout and the size of each region that is copied. For example, the following command inserts a delay of 1000 milliseconds (1 second) between copying each 10 MB region:

```
# vxrelayout -g mydg -o bg,slow=1000,iosize=10m start vol04
```

The default delay and region size values are 250 milliseconds and 1 MB respectively.

To reverse the direction of relayout operation that is stopped, specify the `reverse` keyword to `vxrelayout` as follows:

```
# vxrelayout -g mydg -o bg reverse vol04
```

This undoes changes made to the volume so far, and returns it to its original layout.

If you cancel a relayout using `vxtask abort`, the direction of the conversion is also reversed, and the volume is returned to its original configuration.

See "Managing tasks with vxtask" on page 658.

See the `vxrelayout`(1M) manual page.

See the `vxtask`(1M) manual page.

Adding a mirror to a volume

You can add a mirror to a volume with the `vxassist` command, as follows:

```
# vxassist [-b] [-g diskgroup] mirror volume
```

Specifying the `-b` option makes synchronizing the new mirror a background task.

For example, to create a mirror of the volume `voltest` in the disk group, `mydg`, use the following command:

```
# vxassist -b -g mydg mirror voltest
```

You can also mirror a volume by creating a plex and then attaching it to a volume using the following commands:

```
# vxmake [-g diskgroup] plex plex sd=subdisk ...
# vxplex [-g diskgroup] att volume plex
```

Mirroring all volumes

To mirror all volumes in a disk group to available disk space, use the following command:

```
# /etc/vx/bin/vxmirror -g diskgroup -a
```

To configure VxVM to create mirrored volumes by default, use the following command:

```
# vxmirror -d yes
```

If you make this change, you can still make unmirrored volumes by specifying `nmirror=1` as an attribute to the `vxassist` command. For example, to create an unmirrored 20-gigabyte volume named `nomirror` in the disk group `mydg`, use the following command:

```
# vxassist -g mydg make nomirror 20g nmirror=1
```

Mirroring volumes on a VxVM disk

Mirroring volumes creates one or more copies of your volumes on another disk. By creating mirror copies of your volumes, you protect your volumes against loss of data if a disk fails.

You can use this task on your root disk to make a second copy of the boot information available on an alternate disk. This lets you boot your system even if your root disk fails.

Note: This task only mirrors concatenated volumes. Volumes that are already mirrored or that contain subdisks that reside on multiple disks are ignored

To mirror volumes on a disk

1. Make sure that the target disk has an equal or greater amount of space as the source disk.
2. From the `vxdiskadm` main menu, select `Mirror volumes on a disk` .
3. At the prompt, enter the disk name of the disk that you wish to mirror:

   ```
   Enter disk name [<disk>,list,q,?] mydg02
   ```

4. At the prompt, enter the target disk name (this disk must be the same size or larger than the originating disk):

   ```
   Enter destination disk [<disk>,list,q,?] (default: any) mydg01
   ```

5 At the prompt, press **Return** to make the mirror:

```
Continue with operation? [y,n,q,?] (default: y)
```

The `vxdiskadm` program displays the status of the mirroring operation, as follows:

```
VxVM vxmirror INFO V-5-2-22 Mirror volume voltest-bk00
   .
   .
   .
VxVM INFO V-5-2-674 Mirroring of disk mydg02 is complete.
```

6 At the prompt, indicate whether you want to mirror volumes on another disk (`y`) or return to the `vxdiskadm` main menu (`n`):

```
Mirror volumes on another disk? [y,n,q,?] (default: n)
```

Configuring SmartMove

By default, the SmartMove utility is enabled for all volumes. Configuring the SmartMove feature is only required if you want to change the default behavior, or if you have modified the behavior previously.

SmartMove has three values where SmartMove can be applied or not. The three values are:

Value	Meaning
`none`	Do not use SmartMove at all.
`thinonly`	Use SmartMove for thin aware LUNs only.
`all`	Use SmartMove for all types of LUNs. This is the default value.

To configure the SmartMove value

1 To display the current and default SmartMove values, type the following command:

```
# vxdefault list
KEYWORD                         CURRENT-VALUE   DEFAULT-VALUE
usefssmartmove                  all             all
...
```

2 To set the SmartMove value, type the following command:

```
# vxdefault set usefssmartmove value
```

where *value* is either `none`, `thinonly`, or `all`.

Removing a mirror

When you no longer need a mirror, you can remove it to free disk space.

Note: VxVM will not allow you to remove the last valid plex associated with a volume.

To remove a mirror from a volume, use the following command:

```
# vxassist [-g diskgroup] remove mirror volume
```

You can also use storage attributes to specify the storage to be removed. For example, to remove a mirror on disk `mydg01` from volume `vol01`, enter the following.

Note: The ! character is a special character in some shells. The following example shows how to escape it in a bash shell.

```
# vxassist -g mydg remove mirror vol01 \!mydg01
```

See “Creating a volume on specific disks” on page 151.

Alternatively, use the following command to dissociate and remove a mirror from a volume:

```
# vxplex [-g diskgroup] -o rm dis mirror
```

For example, to dissociate and remove a mirror named `vol01-02` from the disk group `mydg`, use the following command:

```
# vxplex -g mydg -o rm dis vol01-02
```

This command removes the mirror vol01-02 and all associated subdisks. This is equivalent to entering the following commands separately:

```
# vxplex -g mydg dis vol01-02
# vxedit -g mydg -r rm vol01-02
```

Setting tags on volumes

Volume tags implement the SmartTier feature. You can also apply tags to vsets using the same vxvm command syntax as shown below.

The following forms of the vxassist command let you do the following:

- Set a named tag and optional tag value on a volume.
- Replace a tag.
- Remove a tag from a volume.

```
# vxassist [-g diskgroup] settag volume|vset tagname[=tagvalue]
# vxassist [-g diskgroup] replacetag volume|vset oldtag newtag
# vxassist [-g diskgroup] removetag volume|vset tagname
```

To list the tags that are associated with a volume, use the following command:

```
# vxassist [-g diskgroup] listtag [volume|vset]
```

If you do not specify a volume name, all the volumes and vsets in the disk group are displayed. The acronym vt in the TY field indicates a vset.

The following is a sample listtag command:

```
# vxassist -g dg1 listtag vol
```

To list the volumes that have a specified tag name, use the following command:

```
# vxassist [-g diskgroup] list tag=tagname volume
```

Tag names and tag values are case-sensitive character strings of up to 256 characters. Tag names can consist of the following ASCII characters:

- Letters (A through Z and a through z)
- Numbers (0 through 9)
- Dashes (-)
- Underscores (_)
- Periods (.)

A tag name must start with either a letter or an underscore. A tag name must not be the same as the name of a disk in the disk group.

The tag names `site`, `udid`, and `vdid` are reserved. Do not use them. To avoid possible clashes with future product features, do not start tag names with any of the following strings: `asl`, `be`, `nbu`, `sf`, `symc`, or `vx`.

Tag values can consist of any ASCII character that has a decimal value from 32 through 127. If a tag value includes spaces, quote the specification to protect it from the shell, as follows:

```
# vxassist -g mydg settag myvol "dbvol=table space 1"
```

The `list` operation understands dotted tag hierarchies. For example, the listing for `tag=a.b` includes all volumes that have tag names starting with `a.b`.

Managing disk groups

This section describes managing disk groups.

Disk group versions

All disk groups have a version number associated with them. Each major Veritas Volume Manager (VxVM) release introduces a disk group version. To support the new features in the release, the disk group must be the latest disk group version. By default, VxVM creates disk groups with the latest disk group version. For example, Veritas Volume Manager 8.0 creates disk groups with version 310.

Each VxVM release supports a specific set of disk group versions. VxVM can import and perform operations on a disk group of any supported version. However, the operations are limited by what features and operations the disk group version supports. If you import a disk group from a previous version, the latest features may not be available. If you attempt to use a feature from a newer version of VxVM, you receive an error message similar to this:

```
VxVM vxedit ERROR V-5-1-2829 Disk group version doesn't support
feature
```

You must explicitly upgrade the disk group to the appropriate disk group version to use the feature.

See “Upgrading the disk group version” on page 678.

Table 32-7 summarizes the Veritas Volume Manager releases that introduce and support specific disk group versions. It also summarizes the features that are supported by each disk group version.

Table 32-7 Disk group version assignments

VxVM release	Introduces disk group version	New features supported	Supports disk group versions *
8.0	310		20, 30, 40, 50, 60, 70, 80, 90, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 220, 230, 240, 250, 260, 280, 290
7.4.2	290	DCM logging in DCO Disk group level encryption and re-key feature	20, 30, 40, 50, 60, 70, 80, 90, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 220, 230, 240, 250, 260, 280
7.4.1	280	Technology preview: Adaptive synchronous mode in VVR	20, 30, 40, 50, 60, 70, 80, 90, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 220, 230, 240, 250, 260
7.4	260	Volume level I/O shipping Intent lock Encryption of data at rest and over wire	20, 30, 40, 50, 60, 70, 80, 90, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 220, 230, 240, 250
7.3.1	240	Volume encryption for replication	20, 30, 40, 50, 60, 70, 80, 90, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 220, 230
7.2, 7.3	230	Hot-relocation in FSS environments Erasure coded volumes (technology preview) 4K sector size disk support	20, 30, 40, 50, 60, 70, 80, 90, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 220

Table 32-7 Disk group version assignments *(continued)*

VxVM release	Introduces disk group version	New features supported	Supports disk group versions *
7.1	220	■ Block-level encryption support for VxVM volume ■ Maximum IOPS support for VxVM	20, 30, 40, 50, 60, 70, 80, 90, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200
7.0, 6.2	200	■ Atomic Write I/O support ■ SmartIO support for shared volumes	20, 30, 40, 50, 60, 70, 80, 90, 110, 120, 130, 140, 150, 160, 170, 180, 190
6.1	190	■ SmartIO caching ■ Flexible storage sharing ■ CVM enhancements	20, 30, 40, 50, 60, 70, 80, 90, 110, 120, 130, 140, 150, 160, 170, 180
6.0.1	180	■ TRIM support for Solid State Devices (SSDs) ■ CVM availability enhancements	20, 30, 40, 50, 60, 70, 80, 90, 110, 120, 130, 140, 150, 160, 170
6.0	170	■ VVR compression ■ VVR Secondary logging ■ CVM availability enhancements ■ DCO version 30 ■ Recovery for synchronization tasks.	20, 30, 40, 50, 60, 70, 80, 90, 110, 120, 130, 140, 150, 160

Table 32-7 Disk group version assignments *(continued)*

VxVM release	Introduces disk group version	New features supported	Supports disk group versions *
5.1SP1	160	■ Automated bunker replay as part of GCO failover ■ Ability to elect primary during GCO takeover ■ CVM support for more than 32 nodes and up to 64 nodes ■ CDS layout support for large luns (> 1 TB) ■ vxrootadm enhancements	20, 30, 40, 50, 60, 70, 80, 90, 110, 120, 130, 140, 150, 160
5.1	150	SSD device support, migration of ISP dg	20, 30, 40, 50, 60, 70, 80, 90, 110, 120, 130, 140, 150
5.0	140	Data migration, Remote Mirror, coordinator disk groups (used by VCS), linked volumes, snapshot LUN import.	20, 30, 40, 50, 60, 70, 80, 90, 110, 120, 130, 140
5.0	130	■ VVR Enhancements	20, 30, 40, 50, 60, 70, 80, 90, 110, 120, 130
4.1	120	■ Automatic Cluster-wide Failback for A/P arrays ■ Persistent DMP Policies ■ Shared Disk Group Failure Policy	20, 30, 40, 50, 60, 70, 80, 90, 110, 120

Table 32-7 Disk group version assignments *(continued)*

VxVM release	Introduces disk group version	New features supported	Supports disk group versions *
4.0	110	■ Cross-platform Data Sharing (CDS) ■ Device Discovery Layer (DDL) 2.0 ■ Disk Group Configuration Backup and Restore ■ Elimination of `rootdg` as a Special Disk Group ■ Full-Sized and Space-Optimized Instant Snapshots ■ Intelligent Storage Provisioning (ISP) ■ Serial Split Brain Detection ■ Volume Sets (Multiple Device Support for VxFS)	20, 30, 40, 50, 60, 70, 80, 90, 110
3.2, 3.5	90	■ Cluster Support for Oracle Resilvering ■ Disk Group Move, Split and Join ■ Device Discovery Layer (DDL) 1.0 ■ Layered Volume Support in Clusters ■ Ordered Allocation ■ OS Independent Naming Support ■ Persistent FastResync	20, 30, 40, 50, 60, 70, 80, 90

Table 32-7 Disk group version assignments *(continued)*

VxVM release	Introduces disk group version	New features supported	Supports disk group versions *
3.1.1	80	■ VVR Enhancements	20, 30, 40, 50, 60, 70, 80
3.1	70	■ Non-Persistent FastResync ■ Sequential DRL ■ Unrelocate ■ VVR Enhancements	20, 30, 40, 50, 60, 70
3.0	60	■ Online Relayout ■ Safe RAID-5 Subdisk Moves	20, 30, 40, 60
2.5	50	■ SRVM (now known as Veritas Volume Replicator or VVR)	20, 30, 40, 50
2.3	40	■ Hot-Relocation	20, 30, 40
2.2	30	■ VxSmartSync Recovery Accelerator	20, 30
2.0	20	■ Dirty Region Logging (DRL) ■ Disk Group Configuration Copy Limiting ■ Mirrored Volumes Logging ■ New-Style Stripes ■ RAID-5 Volumes ■ Recovery Checkpointing	20
1.3	15		15
1.2	10		10

* To support new features, the disk group must be at least the disk group version of the release when the feature was introduced.

If you need to import a disk group on a system running an older version of Veritas Volume Manager, you can create a disk group with an earlier disk group version.

See “Creating a disk group with an earlier disk group version” on page 679.

Upgrading the disk group version

All Veritas Volume Manager disk groups have an associated version number. Each VxVM release supports a specific set of disk group versions and can import and perform tasks on disk groups with those versions. Some new features and tasks work only on disk groups with the current disk group version.

When you upgrade, VxVM does not automatically upgrade the versions of existing disk groups. If the disk group is a supported version, the disk group can be used “as is”, as long as you do not attempt to use the features of the current version. Until the disk group is upgraded, it may still be deported back to the release from which it was imported.

To use the features in the upgraded release, you must explicitly upgrade the existing disk groups. There is no "downgrade" facility. After you upgrade a disk group, the disk group is incompatible with earlier releases of VxVM that do not support the new version. For disk groups that are shared among multiple servers for failover or for off-host processing, verify that the VxVM release on all potential hosts that may use the disk group supports the disk group version to which you are upgrading.

After upgrading to Storage Foundation 8.0, you must upgrade any existing disk groups that are organized by ISP. Without the version upgrade, configuration query operations continue to work fine. However, configuration change operations will not function correctly.

To list the version of a disk group, use this command:

```
# vxdg list dgname
```

You can also determine the disk group version by using the `vxprint` command with the `-l` format option.

To upgrade a disk group to the highest version supported by the release of VxVM that is currently running, use this command:

```
# vxdg upgrade dgname
```

Creating a disk group with an earlier disk group version

You may sometimes need to create a disk group that can be imported on a system running an older version of Veritas Volume Manager. You must specify the disk group version when you create the disk group, since you cannot downgrade a disk group version.

For example, the default disk group version for a disk group created on a system running Veritas Volume Manager 6.0 is 170. Such a disk group cannot be imported on a system running Veritas Volume Manager 4.1, as that release only supports up to version 120. Therefore, to create a disk group on a system running Veritas Volume Manager 6.0 that can be imported by a system running Veritas Volume Manager 4.1, the disk group must be created with a version of 120 or less.

To create a disk group with a previous version, specify the `-T` *`version`* option to the `vxdg init` command.

Displaying disk group information

To display information on existing disk groups, enter the following command:

```
# vxdg list
NAME    STATE     ID
rootdg  enabled   730344554.1025.tweety
newdg   enabled   731118794.1213.tweety
```

To display more detailed information on a specific disk group, use the following command:

```
# vxdg list diskgroup
```

When you apply this command to a disk group named `mydg`, the output is similar to the following:

```
# vxdg list mydg

Group: mydg
dgid: 962910960.1025.bass
import-id: 0.1
flags:
version: 160
local-activation: read-write
alignment: 512 (bytes)
ssb: on
detach-policy: local
copies: nconfig=default nlog=default
```

```
config: seqno=0.1183 permlen=3448 free=3428 templen=12 loglen=522
config disk sda copy 1 len=3448 state=clean online
config disk sdb copy 1 len=3448 state=clean online
log disk sdc copy 1 len=522
log disk sdd copy 1 len=522
```

To verify the disk group ID and name that is associated with a specific disk (for example, to import the disk group), use the following command:

```
# vxdisk -s list devicename
```

This command provides output that includes the following information for the specified disk. For example, output for disk `sdc` as follows:

```
Disk: sdc
type:   simple
flags:  online ready private autoconfig autoimport imported
diskid: 963504891.1070.bass
dgname: newdg
dgid:   963504895.1075.bass
hostid: bass
info:   privoffset=128
```

Displaying free space in a disk group

Before you add volumes and file systems to your system, make sure that you have enough free disk space to meet your needs.

To display free space in the system, use the following command:

```
# vxdg free
```

The following is example output:

```
GROUP DISK      DEVICE    TAG       OFFSET  LENGTH   FLAGS
mydg  mydg01    sda       sda       0       4444228  -
mydg  mydg02    sdb       sdb       0       4443310  -
newdg znewdg01  sdc       sdc       0       4443310  -
newdg newdg02   sdd       sdd       0       4443310  -
oradg oradg01   sde       sde       0       4443310  -
```

To display free space for a disk group, use the following command:

```
# vxdg -g diskgroup free
```

where *-g diskgroup* optionally specifies a disk group.

For example, to display the free space in the disk group, `mydg`, use the following command:

```
# vxdg -g mydg free
```

The following example output shows the amount of free space in sectors:

```
DISK    DEVICE     TAG        OFFSET  LENGTH    FLAGS
mydg01  sda        sda        0       4444228   -
mydg02  sdb        sdb        0       4443310   -
```

Creating a disk group

You must associate a disk group with at least one disk. You can create a new disk group when you select `Add or initialize one or more disks` from the main menu of the `vxdiskadm` command to add disks to VxVM control. The disks to be added to a disk group must not belong to an existing disk group. A disk group name cannot include a period (.) character.

You can create a shared disk group.

You can also use the `vxdiskadd` command to create a new disk group. The command dialog is similar to that described for the `vxdiskadm` command.

In the following example, `sdd` is the device name of a disk that is not currently assigned to a disk group.

```
# vxdiskadd sdd
```

See “Adding a disk to VxVM” on page 293.

You can also create disk groups using the following `vxdg init` command:

```
# vxdg init diskgroup [cds=on|off] diskname=devicename
```

For example, to create a disk group named `mktdg` on device `sdc`, enter the following:

```
# vxdg init mktdg mktdg01=sdc
```

The disk that is specified by the device name, `sdc`, must have been previously initialized with `vxdiskadd` or `vxdiskadm`. The disk must not currently belong to a disk group.

You can use the `cds` attribute with the `vxdg init` command to specify whether a new disk group is compatible with the Cross-platform Data Sharing (CDS) feature. Newly created disk groups are compatible with CDS by default (equivalent to specifying `cds=on`). If you want to change this behavior, edit the file

/etc/default/vxdg and set the attribute-value pair cds=off in this file before creating a new disk group.

You can also use the following command to set this attribute for a disk group:

```
# vxdg -g diskgroup set cds=on|off
```

You can use the -o encrypted option to encrypt the volumes in a disk group. All volumes created in the disk group are encrypted. The vxencrypt list command output displays the message "encrypted by default" for such disk groups.

```
# vxdg -o encrypted init mydg disk1 disk2 disk3
# vxencrypt list
```

If you want to encrypt all the volumes on a host, update the file /etc/default/vxassist to include the following line:

```
encrypted=on
```

Removing a disk from a disk group

Before you can remove the last disk from a disk group, you must disable the disk group.

See "Disabling a disk group" on page 710.

As an alternative to disabling the disk group, you can destroy it.

See "Destroying a disk group" on page 710.

If a disk contains no subdisks, you can remove it from its disk group with the following command:

```
# vxdg [-g diskgroup ] rmdisk diskname
```

For example, to remove mydg02 from the disk group mydg, enter the following:

```
# vxdg -g mydg rmdisk mydg02
```

If the disk has subdisks on it when you try to remove it, the following error message is displayed:

```
VxVM vxdg ERROR V-5-1-552 Disk diskname is used by one or more
subdisks
Use -k to remove device assignment.
```

Using the -k option lets you remove the disk even if it has subdisks.

See the vxdg(1M) manual page.

Warning: Use of the `-k` option to `vxdg` can result in data loss.

After you remove the disk from its disk group, you can (optionally) remove it from VxVM control completely. Enter the following:

```
# vxdiskunsetup devicename
```

For example, to remove the disk `sdc` from VxVM control, enter the following:

```
# vxdiskunsetup sdc
```

You can remove a disk on which some subdisks of volumes are defined. For example, you can consolidate all the volumes onto one disk. If you use `vxdiskadm` to remove a disk, you can choose to move volumes off that disk. To do this, run `vxdiskadm` and select `Remove a disk` from the main menu.

If the disk is used by some volumes, this message is displayed:

```
VxVM ERROR V-5-2-369 The following volumes currently use part of
disk mydg02:

home usrvol

Volumes must be moved from mydg02 before it can be removed.

Move volumes to other disks? [y,n,q,?] (default: n)
```

If you choose `y`, all volumes are moved off the disk, if possible. Some volumes may not be movable. The most common reasons why a volume may not be movable are as follows:

- There is not enough space on the remaining disks.
- Plexes or striped subdisks cannot be allocated on different disks from existing plexes or striped subdisks in the volume.

If `vxdiskadm` cannot move some volumes, you may need to remove some plexes from some disks to free more space before proceeding with the disk removal operation.

Deporting a disk group

Deporting a disk group disables access to a disk group that is enabled (imported) by the system. Deport a disk group if you intend to move the disks in a disk group to another system.

To deport a disk group

1. Stop all activity by applications to volumes that are configured in the disk group that is to be deported. Unmount file systems and shut down databases that are configured on the volumes.

   If the disk group contains volumes that are in use (for example, by mounted file systems or databases), deportation fails.

2. To stop the volumes in the disk group, use the following command

   ```
   # vxvol -g diskgroup stopall
   ```

3. From the `vxdiskadm` main menu, select `Remove access to (deport) a disk group .`

4. At prompt, enter the name of the disk group to be deported. In the following example it is `newdg`):

   ```
   Enter name of disk group [<group>,list,q,?] (default: list)
   newdg
   ```

5. At the following prompt, enter `y` if you intend to remove the disks in this disk group:

   ```
   Disable (offline) the indicated disks? [y,n,q,?]  (default: n) y
   ```

6. At the following prompt, press **Return** to continue with the operation:

   ```
   Continue with operation? [y,n,q,?] (default: y)
   ```

   After the disk group is deported, the `vxdiskadm` utility displays the following message:

   ```
   VxVM INFO V-5-2-269 Removal of disk group newdg was
     successful.
   ```

7. At the following prompt, indicate whether you want to disable another disk group (`y`) or return to the `vxdiskadm` main menu (`n`):

   ```
   Disable another disk group? [y,n,q,?] (default: n)
   ```

   You can use the following `vxdg` command to deport a disk group:

   ```
   # vxdg deport diskgroup
   ```

Importing a disk group

Importing a disk group enables access by the system to a disk group. To move a disk group from one system to another, first disable (deport) the disk group on the original system, and then move the disk between systems and enable (import) the disk group.

By default, VxVM recovers and starts any disabled volumes in the disk group when you import the disk group. To prevent VxVM from recovering the disabled volumes, turn off the automatic recovery feature. For example, after importing the disk group, you may want to do some maintenance before starting the volumes.

See "Setting the automatic recovery of volumes" on page 686.

To import a disk group

1. To ensure that the disks in the deported disk group are online, use the following command:

   ```
   # vxdisk -s list
   ```

2. From the `vxdiskadm` main menu, select `Enable access to (import) a disk group`.

3. At the following prompt, enter the name of the disk group to import (in this example, `newdg`):

   ```
   Select disk group to import [<group>,list,q,?] (default: list)
     newdg
   ```

   When the import finishes, the `vxdiskadm` utility displays the following success message:

   ```
   VxVM INFO V-5-2-374 The import of newdg was successful.
   ```

4. At the following prompt, indicate whether you want to import another disk group (`y`) or return to the `vxdiskadm` main menu (`n`):

   ```
   Select another disk group? [y,n,q,?] (default: n)
   ```

   You can also use the following `vxdg` command to import a disk group:

   ```
   # vxdg import diskgroup
   ```

   You can also import the disk group as a shared disk group.

Setting the automatic recovery of volumes

By default, VxVM recovers and starts any disabled volumes in the disk group when you import the disk group. To prevent VxVM from recovering the disabled volumes, turn off the automatic volume recovery. For example, after importing the disk group, you may want to do some maintenance before starting the volumes.

To turn off the automatic volume recovery feature

- To turn off automatic volume recovery for the entire system, use the following command.

  ```
  # vxtune autostartvolumes off
  ```

  OR

  To turn off automatic volume recovery for a specific disk group import operation, use the `noautostart` option.

  ```
  # vxdg -o noautostart import diskgroup
  ```

Handling of minor number conflicts

The volume device minor numbers in a disk group to be imported may conflict with existing volume devices. In releases of VxVM before release 5.1, the conflicts resulted in failures. Either the disk group import operation failed, or the slave node failed to join for a shared disk group. When this situation happened, you had to run the `vxdg reminor` command manually to resolve the minor conflicts. Starting in release 5.1, VxVM can automatically resolve minor number conflicts.

If a minor conflict exists when a disk group is imported, VxVM automatically assigns a new base minor to the disk group, and reminors the volumes in the disk group, based on the new base minor. You do not need to run the `vxdg reminor` command to resolve the minor conflicts.

To avoid any conflicts between shared and private disk groups, the minor numbers are divided into shared and private pools. VxVM allocates minor numbers of shared disk groups only from the shared pool, and VxVM allocates minor numbers of private disk groups only from the private pool. If you import a private disk group as a shared disk group or vice versa, the device minor numbers are re-allocated from the correct pool. The disk group is dynamically reminored.

By default, private minor numbers range from 0-32999, and shared minor numbers start from 33000. You can change the division if required. For example, you can set the range for shared minor numbers to start from a lower number. This range provides more minor numbers for shared disk groups and fewer minor numbers for private disk groups.

Normally the minor numbers in private and shared pools are sufficient, so there is no need to make changes to the division.

Note: To make the new division take effect, you must run `vxdctl enable` or restart `vxconfigd` after the tunable is changed in the defaults file. The division on all the cluster nodes must be exactly the same, to prevent node failures for node join, volume creation, or disk group import operations.

To change the division between shared and private minor numbers

1. Add the tunable `sharedminorstart` to the defaults file `/etc/default/vxsf` . For example, to change the shared minor numbers so that the range starts from 20000, set the following line in the `/etc/default/vxsf` file.

   ```
   sharedminorstart=20000
   ```

   You cannot set the shared minor numbers to start at less than 1000. If `sharedminorstart` is set to values between 0 to 999, the division of private minor numbers and shared disk group minor numbers is set to 1000. The value of 0 disables dynamic renumbering.

2. Run the following command:

   ```
   # vxdctl enable
   ```

In certain scenarios, you may need to disable the division of between shared minor numbers and private minor numbers. For example, to prevent the device minor numbers from being changed when you upgrade from a previous release. In this case, disable the dynamic reminoring before you install the new VxVM rpm.

To disable the division between shared and private minor numbers

1. Set the tunable `sharedminorstart` in the defaults file `/etc/default/vxsf` to 0 (zero). Set the following line in the `/etc/default/vxsf` file.

   ```
   sharedminorstart=0
   ```

2. Run the following command:

   ```
   # vxdctl enable
   ```

Moving disk groups between systems

An important feature of disk groups is that they can be moved between systems. If all disks in a disk group are moved from one system to another, then the disk group can be used by the second system. You do not have to re-specify the configuration.

To move a disk group between systems

1 Confirm that all disks in the diskgroup are visible on the target system. This may require masking and zoning changes.

2 On the source system, stop all volumes in the disk group, then deport (disable local access to) the disk group with the following command:

```
# vxdg deport diskgroup
```

3 Move all the disks to the target system and perform the steps necessary (system-dependent) for the target system and VxVM to recognize the new disks.

This can require a reboot, in which case the `vxconfigd` daemon is restarted and recognizes the new disks. If you do not reboot, use the command `vxdctl enable` to restart the `vxconfigd` program so VxVM also recognizes the disks.

4 Import (enable local access to) the disk group on the target system with this command:

```
# vxdg import diskgroup
```

Warning: All disks in the disk group must be moved to the other system. If they are not moved, the import fails.

5 By default, VxVM enables and starts any disabled volumes after the disk group is imported.

See "Setting the automatic recovery of volumes" on page 686.

If the automatic volume recovery feature is turned off, start all volumes with the following command:

```
# vxrecover -g diskgroup -sb
```

You can also move disks from a system that has crashed. In this case, you cannot deport the disk group from the source system. When a disk group is created or imported on a system, that system writes a lock on all disks in the disk group.

Warning: The purpose of the lock is to ensure that SAN-accessed disks are not used by both systems at the same time. If two systems try to access the same disks at the same time, this must be managed using software such as the clustering functionality of VxVM. Otherwise, data and configuration information stored on the disk may be corrupted, and may become unusable.

Handling errors when importing disks

When you move disks from a system that has crashed or that failed to detect the group before the disk was moved, the locks stored on the disks remain and must be cleared. The system returns the following error message:

```
VxVM vxdg ERROR V-5-1-587 disk group groupname: import failed:
Disk is in use by another host
```

The next message indicates that the disk group does not contains any valid disks (not that it does not contains any disks):

```
VxVM vxdg ERROR V-5-1-587 Disk group groupname: import failed:
No valid disk found containing disk group
```

The disks may be considered invalid due to a mismatch between the host ID in their configuration copies and that stored in the `/etc/vx/volboot` file.

To clear locks on a specific set of devices, use the following command:

```
# vxdisk clearimport devicename ...
```

To clear the locks during import, use the following command:

```
# vxdg -C import diskgroup
```

Warning: Be careful when using the `vxdisk clearimport` or `vxdg -C import` command on systems that see the same disks via a SAN. Clearing the locks allows those disks to be accessed at the same time from multiple hosts and can result in corrupted data.

A disk group can be imported successfully if all the disks are accessible that were visible when the disk group was last imported successfully. However, sometimes you may need to specify the `-f` option to forcibly import a disk group if some disks are not available. If the `import` operation fails, an error message is displayed.

The following error message indicates a fatal error that requires hardware repair or the creation of a new disk group, and recovery of the disk group configuration and data:

```
VxVM vxdg ERROR V-5-1-587 Disk group groupname: import failed:
Disk group has no valid configuration copies
```

The following error message indicates a recoverable error.

```
VxVM vxdg ERROR V-5-1-587 Disk group groupname: import failed:
Disk for disk group not found
```

If some of the disks in the disk group have failed, you can force the disk group to be imported by specifying the `-f` option to the `vxdg import` command:

```
# vxdg -f import diskgroup
```

Warning: Be careful when using the `-f` option. It can cause the same disk group to be imported twice from different sets of disks. This can cause the disk group configuration to become inconsistent.

See “Handling conflicting configuration copies” on page 703.

As using the `-f` option to force the import of an incomplete disk group counts as a successful import, an incomplete disk group may be imported subsequently without this option being specified. This may not be what you expect.

You can also import the disk group as a shared disk group.

These operations can also be performed using the `vxdiskadm` utility. To deport a disk group using `vxdiskadm`, select `Remove access to (deport) a disk group` from the main menu. To import a disk group, select `Enable access to (import) a disk group`. The `vxdiskadm` import operation checks for host import locks and prompts to see if you want to clear any that are found. It also starts volumes in the disk group.

Reserving minor numbers for disk groups

A device minor number uniquely identifies some characteristic of a device to the device driver that controls that device. It is often used to identify some characteristic mode of an individual device, or to identify separate devices that are all under the control of a single controller. VxVM assigns unique device minor numbers to each object (volume, plex, subdisk, disk, or disk group) that it controls.

When you move a disk group between systems, it is possible for the minor numbers that it used on its previous system to coincide with those of objects known to VxVM on the new system. To get around this potential problem, you can allocate separate ranges of minor numbers for each disk group. VxVM uses the specified range of minor numbers when it creates volume objects from the disks in the disk group. This guarantees that each volume has the same minor number across reboots or reconfigurations. Disk groups may then be moved between machines without causing device number collisions.

VxVM chooses minor device numbers for objects created from this disk group starting at the base minor number *base_minor*. Minor numbers can range from this value up to 65,535 on 2.6 and later kernels. Try to leave a reasonable number of unallocated minor numbers near the top of this range to allow for temporary device number remapping in the event that a device minor number collision may still occur.

VxVM reserves the range of minor numbers from 0 to 999 for use with volumes in the boot disk group. For example, the `rootvol` volume is always assigned minor number 0.

If you do not specify the base of the minor number range for a disk group, VxVM chooses one at random. The number chosen is at least 1000, is a multiple of 1000, and yields a usable range of 1000 device numbers. The chosen number also does not overlap within a range of 1000 of any currently imported disk groups, and it does not overlap any currently allocated volume device numbers.

Note: The default policy ensures that a small number of disk groups can be merged successfully between a set of machines. However, where disk groups are merged automatically using failover mechanisms, select ranges that avoid overlap.

To view the base minor number for an existing disk group, use the `vxprint` command as shown in the following examples for the disk group, `mydg`:

```
# vxprint -l mydg | grep minors
minors: >=45000
```

```
# vxprint -g mydg -m | egrep base_minor
base_minor=45000
```

To set a base volume device minor number for a disk group that is being created, use the following command:

```
# vxdg init diskgroup minor=base_minor disk_access_name ...
```

For example, the following command creates the disk group, `newdg`, that includes the specified disks, and has a base minor number of 30000:

```
# vxdg init newdg minor=30000 sdc sdd
```

If a disk group already exists, you can use the `vxdg reminor` command to change its base minor number:

```
# vxdg -g diskgroup reminor new_base_minor
```

For example, the following command changes the base minor number to 30000 for the disk group, `mydg`:

```
# vxdg -g mydg reminor 30000
```

If a volume is open, its old device number remains in effect until the system is rebooted or until the disk group is deported and re-imported. If you close the open volume, you can run `vxdg reminor` again to allow the renumbering to take effect without rebooting or re-importing.

An example of where it is necessary to change the base minor number is for a cluster-shareable disk group. The volumes in a shared disk group must have the same minor number on all the nodes. If there is a conflict between the minor numbers when a node attempts to join the cluster, the join fails. You can use the `reminor` operation on the nodes that are in the cluster to resolve the conflict. In a cluster where more than one node is joined, use a base minor number which does not conflict on any node.

See the vxdg(1M) manual page.

See "Handling of minor number conflicts" on page 686.

Compatibility of disk groups between platforms

For disk groups that support the Cross-platform Data Sharing (CDS) feature, the upper limit on the minor number range is restricted on AIX, HP-UX, Linux (with a 2.6 or later kernel) and Solaris to 65,535 to ensure portability between these operating systems.

On a Linux platform with a pre-2.6 kernel, the number of minor numbers per major number is limited to 256 with a base of 0. This has the effect of limiting the number of volumes and disks that can be supported system-wide to a smaller value than is allowed on other operating system platforms. The number of disks that are supported by a pre-2.6 Linux kernel is typically limited to a few hundred. With the extended major numbering scheme that was implemented in VxVM 4.0 on Linux, a maximum of 4079 volumes could be configured, provided that a contiguous block of 15 extended major numbers was available.

VxVM 4.1 and later releases run on a 2.6 version Linux kernel, which increases the number of minor devices that are configurable from 256 to 65,536 per major device number. This allows a large number of volumes and disk devices to be configured on a system. The theoretical limit on the number of DMP and volume devices that can be configured on such a system are 65,536 and 1,048,576 respectively. However, in practice, the number of VxVM devices that can be configured in a single disk group is limited by the size of the private region.

When a CDS-compatible disk group is imported on a Linux system with a pre-2.6 kernel, VxVM attempts to reassign the minor numbers of the volumes, and fails if this is not possible.

To help ensure that a CDS-compatible disk group is portable between operating systems, including Linux with a pre-2.6 kernel, use the following command to set the maxdev attribute on the disk group:

```
# vxdg -g diskgroup set maxdev=4079
```

Note: Such a disk group may still not be importable by VxVM 4.0 on Linux with a pre-2.6 kernel if it would increase the number of minor numbers on the system that are assigned to volumes to more than 4079, or if the number of available extended major numbers is smaller than 15.

You can use the following command to discover the maximum number of volumes that are supported by VxVM on a Linux host:

```
# cat /proc/sys/vxvm/vxio/vol_max_volumes
4079
```

See the vxdg(1M) manual page.

Importing a disk group containing hardware cloned disks

When you import a disk group, VxVM tries to ensure that the set of disks is consistent and does not contain duplicate disks. By default, VxVM does not import disks which have the udid_mismatch flag or the clone_disk flag. If the disk group contains both clone and non-clone disks, the default is to import the disk group with only the original (non-cloned) disks. VxVM does not support importing a mix of cloned and non-cloned disks in the same disk group.

See "How VxVM handles hardware clones or snapshots" on page 91.

If you want to import the cloned disks (disk copies), you can do one of the following:

- Update the UDID and clear the clone disk flag.
  If you want to use the clone disk as a standard data disk, you can update the UDID for the disk and clear the clone_disk flag. Use this option if the disk is not really a clone or if you want to use the entire set of clone disks on a separate host and no longer care to maintain the distinction between clone disks and original disks.
  See "Writing a new UDID to a disk" on page 695.

- Import the disk group with the cloned disks.
  Import only the cloned disks, and do not import the original standard disks. Specify the useclonedev option. By default, the clone_disk flag remains on the disks, so you can continue to distinguish between the original disks and the copies.
  See "Importing the existing disk group with only the cloned disks" on page 696.

- Import the cloned disks as a new disk group.
  Import only the cloned disks into a new disk group. VxVM no longer recognized the disks as copies, so the clone_disk flag is removed. The disk group becomes a standard disk group with the copies of the disks. This option only applies if a single consistent set of clone disks exists in the disk group. If more than one disk has the same UDID, VxVM prevents the disk group from being imported. For example, you cannot use the option if two different point-in-time snapshots were created, because the disk group has two copies of each disk.
  See "Importing the cloned disks as a new standard disk group" on page 697.

- Import a consistent set of cloned disks with tagging

If there are multiple copies of the same disks in a source disk group, because of multiple hardware snapshots, you can use VxVM disk tagging to identify a consistent set of cloned disks. Then you can import the tagged disks.

See "Importing a set of cloned disks with tags" on page 698.

Considerations when using EMC CLARiiON SNAPSHOT LUNs

If you need to import the Snapshot LUN of a primary LUN to the same host as the original LUN, be aware of the following limitation.

If you are using Enclosure-based naming (EBN) with the Array Volume id (AVID) enabled, turn off name persistence during device discovery before importing the snapshot LUN to the original host.

To turn off name persistence, use the following command:

```
# vxddladm set namingscheme=ebn persistence=no use_avid=yes
```

After DDL recognizes the LUN, turn on name persistence using the following command:

```
# vxddladm set namingscheme=ebn persistence=yes use_avid=yes
```

Writing a new UDID to a disk

If VxVM has marked a disk with the `udid_mismatch` flag or the `clone_disk` flag, you can update the stored UDID for the disk. This procedure uses the current value of the UDID that is stored in the Device Discovery Layer (DDL) database to correct the value in the private region. Use this option any time you do not need to maintain the distinction between clone disks and the original disks. For example, if the disk is not really a clone or if you are using the entire set of clone disks on a separate host.

To update the UDID for a disk

- Use the following command to update the unique disk identifier (UDID) for one or more disks that are not part of an imported disk group.

  ```
  # vxdisk [-cf] [-g diskgroup ] updateudid disk ...
  ```

  For example, the following command updates the UDIDs for the disks `sdg` and `sdh`:

  ```
  # vxdisk updateudid sdg sdh
  ```

  The `-f` option must be specified if VxVM has not set the `udid_mismatch` flag for a disk.

  If VxVM has set the `udid_mismatch` flag on a disk that is not a clone, specify the `-c` option to remove the `udid_mismatch` flag and the `clone` flag.

Importing the existing disk group with only the cloned disks

If the standard (non-clone) disks in a disk group are not imported, you can import the existing disk group with only the cloned disks. By default, the `clone_disk` flag is set on the disks so that you can continue to distinguish between the original disks and the cloned disks.

This procedure is useful for temporary scenarios. For example, if you want to import only the clone disks to verify the point-in-time copy. After you have verified the clone disks, you can deport the clone disks and import the standard disks again.

Be sure to import a consistent set of cloned disks, which represent a single point of time copy of the original disks. Each of the disks must have either the `udid_mismatch` flag or the `clone_disk` flag or both. No two of the disks should have the same UDID. That is, there must not be two copies of the same original disk.

You must use disk tags if multiple copies of disks in a disk group are present on the system.

See "Importing a set of cloned disks with tags" on page 698.

VxVM does not support a disk group with both clone and non-clone disks. If you want to import both clone disks and standard disks simultaneously, you must specify a new disk group name for the clone disk group.

See "Importing the cloned disks as a new standard disk group" on page 697.

To import the disk group with only the cloned disks

1. Make sure that at least one of the cloned disks has a copy of the current configuration database in its private region.

   See "Setting up configuration database copies (metadata) for a disk group" on page 700.

2. Import only the cloned disks by specifying the `-o useclonedev=on` option to the `vxdg import` command, as shown in this example:

   ```
   # vxdg -o useclonedev=on [-o updateid] import mydg
   ```

   This form of the command allows only cloned disks to be imported. All non-cloned disks remain unimported.

   Specify the `-o updateid` option to write new identification attributes to the disks, and to set the `clone_disk` flag on the disks. (The `vxdisk set clone=on` command can also be used to set the flag.)

Importing the cloned disks as a new standard disk group

If the standard (non-clone) disks in a disk group are already imported, you cannot simultaneously import the cloned disks in the same disk group. VxVM does not support a disk group with both clone and non-clone disks. If you want to import the cloned disks, you must specify a new disk group name for the disk group containing the cloned disks.

Be sure to import a consistent set of cloned disks, which represent a single point of time copy of the original disks. Each of the disks must have either the `udid_mismatch` flag or the `clone_disk` flag or both. No two of the disks should have the same stored UDID. That is, there must not be two copies of the same original disk.

You must use tags if multiple copies of one or more cloned disks exist.

See "Importing a set of cloned disks with tags" on page 698.

After the import operation, the `udid_mismatch` flag and the `clone_disk` flag are cleared from the disks in the new disk group. The new disk group becomes a standard disk group, and the disks become standard disks.

To import the cloned disks as a new disk group

1. Make sure that at least one of the cloned disks has a copy of the current configuration database in its private region.

   See "Setting up configuration database copies (metadata) for a disk group" on page 700.

2. Import only the cloned disks by specifying the `-o useclonedev=on` option to the `vxdg import` command, as shown in this example:

   ```
   # vxdg -n clonedg -o useclonedev=on -o tag=my_tagged_disks \
     import mydg
   ```

   This form of the command allows only cloned disks to be imported.

Importing a set of cloned disks with tags

If you copy the same set of disks more than once, each disk has multiple clone disks. VxVM recognizes the difference between the source disks and the clone disks, but cannot determine which clone disks represent a consistent set of data. For example, if you take several hardware snapshots at different times, one set of cloned disks represents each snapshot point in time.

If you attempt to import the cloned disks of the disk group (using the `-o useclonedev` option), the import operation fails if VxVM detects multiple clones of the same disks. This behavior prevents import of an inconsistent set of disks. For example, a mix of snapshot disks from different points in time.

If multiple clone sets exist, you must indicate to VxVM which clone disks to select for import. Identify the disks that make up a consistent set and assign VxVM tags to the disks. Then, you can import the cloned disks with the specified tag.

To import a set of cloned disks with tags

1. Identify which disks need to be imported together as a set.

```
# vxdisk -o alldgs list

DEVICE       TYPE         DISK    GROUP  STATUS
EMC0_4       auto:cdsdisk mydg01  mydg   online
EMC0_6       auto:cdsdisk mydg02  mydg   online
EMC0_8       auto:cdsdisk -      (mydg)  online udid_mismatch
EMC0_15      auto:cdsdisk -      (mydg)  online udid_mismatch
EMC0_18      auto:cdsdisk -      (mydg)  online udid_mismatch
EMC0_24      auto:cdsdisk -      (mydg)  online udid_mismatch
```

2. If the disks are not already tagged, use the following command to tag all the disks in the disk group that are to be imported:

```
# vxdisk [-g diskgroup ] settag tagname
       disk ...
```

   where *tagname* is a string of up to 128 characters, not including spaces or tabs.

   For example, suppose the **udid_mismatch** disks represent two different point-in-time snapshots. To distinguish between the snapshots, you can tag the disks as follows:

```
# vxdisk settag snaptag1=snap1 EMC0_8 EMC0_15

# vxdisk settag snaptag2=snap2 EMC0_18 EMC0_24
```

   See the `vxdisk`(1M) manual page.

3 To check which disks are tagged, use the `vxdisk listtag` command:

```
# vxdisk listtag

DEVICE       NAME     VALUE
EMC0_8       snaptag1 snap1
EMC0_15      snaptag1 snap1
EMC0_18      snaptag2 snap2
EMC0_24      snaptag2 snap2
```

4 To import the cloned disks that are tagged as `snaptag1`, update the UDIDs. You must assign a disk group name other than `mydg`, because the `mydg` disk group is already imported.

```
# vxdg -n bcvdg -o useclonedev=on -o tag=snaptag1 -o updateid \
import mydg
# vxdisk -o alldgs list

DEVICE        TYPE          DISK    GROUP  STATUS
EMC0_4        auto:cdsdisk mydg01  mydg   online
EMC0_6        auto:cdsdisk mydg02  mydg   online
EMC0_8        auto:cdsdisk mydg01  bcvdg  online
EMC0_15       auto:cdsdisk mydg02  bcvdg  online
EMC0_18       auto:cdsdisk -       (mydg) online udid_mismatch
EMC0_24       auto:cdsdisk -       (mydg) online udid_mismatch
```

The cloned disks `EMC0_18` and `EMC0_24` are not imported, since they do not have the `snaptag1`.

The state of the imported cloned disks has changed from `online udid_mismatch` to `online`. The disks are now in a new disk group, so VxVM removes the `clone_disk` flag.

See the `vxdg`(1M) manual page.

Setting up configuration database copies (metadata) for a disk group

Each VxVM disk group has a configuration database that contains persistent configuration data (or metadata) about the objects in the disk group. VxVM consults this database when the disk group is imported. The configuration database is stored in the private region of one or more VxVM disks in the disk group. The number of stored configuration copies is set for each disk group.

See the `vxdg`(1M) manual page.

If you import only a partial set of disks in a disk group, you must ensure that at least one of the imported disks contains a copy of the current configuration database.

To set up the configuration copies on a set of disks

1 Use the following command to place a copy of the configuration database and kernel log on all disks in a disk group that share the specified tag:

```
# vxdg [-g diskgroup] set tagmeta=on tag=tagname nconfig=all \
  nlog=all
```

2 If you have set `tagmeta=on` for a disk group, use the following command to view the disk tags and the value set for the number of configuration copies. A value of -1 indicates that all tagged disks maintain configuration or log copies.

```
# vxdg listmeta diskgroup
```

To set up the configuration copy on a specific disk

- Use the following command to place a copy of the configuration copy (metadata) on the specified disk, regardless of the placement policy for the disk group. You can set this attribute before or after the disk is added to a disk group.

```
# vxdisk [-g diskgroup] set disk keepmeta=always
```

Renaming a disk group

Only one disk group of a given name can exist per system. It is not possible to import or deport a disk group when the target system already has a disk group of the same name. To avoid this problem, VxVM allows you to rename a disk group during import or deport.

To rename a disk group during import, use the following command:

```
# vxdg [-t] -n newdg import diskgroup
```

If the `-t` option is included, the import is temporary and does not persist across reboots. In this case, the stored name of the disk group remains unchanged on its original host, but the disk group is known by the name specified by *`newdg`* to the importing host. If the `-t` option is not used, the name change is permanent.

For example, this command temporarily renames the disk group, `mydg`, as `mytempdg` on import:

```
# vxdg -t -n mytempdg import mydg
```

To rename a disk group during deport, use the following command:

```
# vxdg [-h hostname] -n newdg deport diskgroup
```

When renaming on deport, you can specify the `-h hostname` option to assign a lock to an alternate host. This ensures that the disk group is automatically imported when the alternate host reboots.

For example, this command renames the disk group, `mydg`, as `myexdg`, and deports it to the host, `jingo`:

```
# vxdg -h jingo -n myexdg deport mydg
```

You cannot use this method to rename the boot disk group because it contains volumes that are in use by mounted file systems (such as `/`). To rename the boot disk group, you must first unmirror and unencapsulate the root disk, and then re-encapsulate and remirror the root disk in a different disk group. This disk group becomes the new boot disk group.

See “Rootability” on page 737.

To temporarily move the boot disk group, `bootdg`, from one host to another (for repair work on the root volume, for example) and then move it back

1. On the original host, identify the disk group ID of the `bootdg` disk group to be imported with the following command:

   ```
   # vxdisk -g bootdg -s list

   dgname: rootdg
   dgid: 774226267.1025.tweety
   ```

   In this example, the administrator has chosen to name the boot disk group as `rootdg`. The ID of this disk group is `774226267.1025.tweety`.

   This procedure assumes that all the disks in the boot disk group are accessible by both hosts.

2. Shut down the original host.

3 On the importing host, import and rename the rootdg disk group with this command:

```
# vxdg -tC -n newdg import diskgroup
```

The -t option indicates a temporary import name, and the -C option clears import locks. The -n option specifies an alternate name for the rootdg being imported so that it does not conflict with the existing rootdg. *diskgroup* is the disk group ID of the disk group being imported (for example, 774226267.1025.tweety).

If a reboot or crash occurs at this point, the temporarily imported disk group becomes unimported and requires a reimport.

4 After the necessary work has been done on the imported disk group, deport it back to its original host with this command:

```
# vxdg -h hostname deport diskgroup
```

Here *hostname* is the name of the system whose rootdg is being returned (the system name can be confirmed with the command uname -n).

This command removes the imported disk group from the importing host and returns locks to its original host. The original host can then automatically import its boot disk group at the next reboot.

Handling conflicting configuration copies

If an incomplete disk group is imported on several different systems, this can create inconsistencies in the disk group configuration copies that you may need to resolve manually. This section and following sections describe how such a condition can occur, and how to correct it. (When the condition occurs in a cluster that has been split, it is usually referred to as a serial split brain condition).

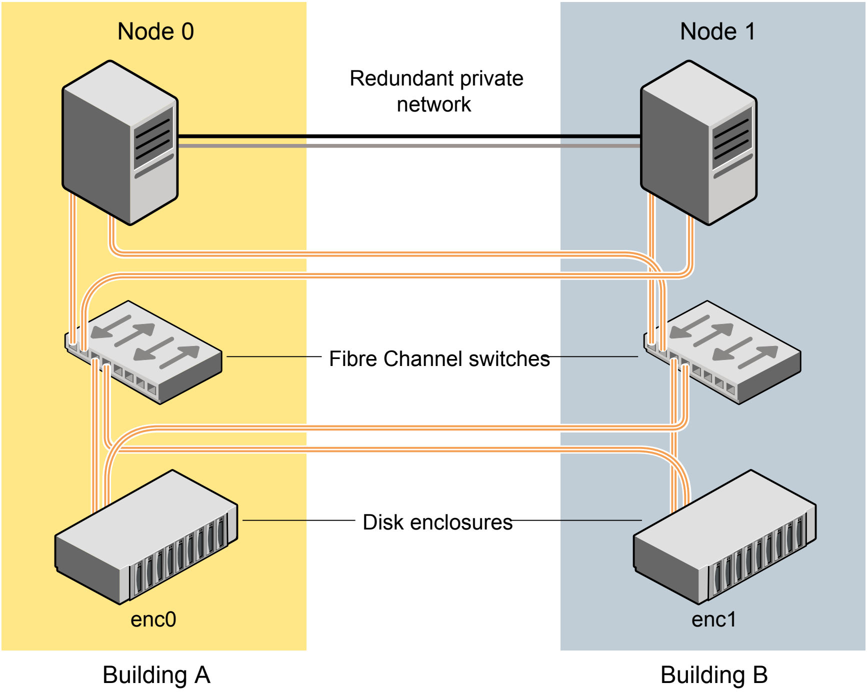
Example of a serial split brain condition in a cluster

This section presents an example of how a serial split brain condition might occur for a shared disk group in a cluster. Conflicts between configuration copies can also occur for private disk groups in clustered and non-clustered configurations where the disk groups have been partially imported on different systems.

A campus cluster (also known as a stretch cluster or remote mirror configuration) typically consists of a 2-node cluster where each component (server, switch and storage) of the cluster exists in a separate building.

Figure 32-5 shows a 2-node cluster with node 0, a Fibre Channel switch and disk enclosure `enc0` in building A, and node 1, another switch and enclosure `enc1` in building B.

Figure 32-5 Typical arrangement of a 2-node campus cluster



The Fibre Channel connectivity is multiply redundant to implement redundant-loop access between each node and each enclosure. As usual, the two nodes are also linked by a redundant private network.

A serial split brain condition typically arises in a cluster when a private (non-shared) disk group is imported on Node 0 with Node 1 configured as the failover node.

If the network connections between the nodes are severed, both nodes think that the other node has died. (This is the usual cause of the split brain condition in clusters). If a disk group is spread across both enclosure `enc0` and `enc1`, each portion loses connectivity to the other portion of the disk group. Node 0 continues to update to the disks in the portion of the disk group that it can access. Node 1, operating as the failover node, imports the other portion of the disk group (with the `-f` option set), and starts updating the disks that it can see.

When the network links are restored, attempting to reattach the missing disks to the disk group on Node 0, or to re-import the entire disk group on either node, fails.

VxVM increments the serial ID in the disk media record of each imported disk in all the disk group configuration databases on those disks, and also in the private region of each imported disk. The value that is stored in the configuration database represents the serial ID that the disk group expects a disk to have. The serial ID that is stored in a disk's private region is considered to be its actual value. VxVM detects the serial split brain when the actual serial ID of the disks that are being attached mismatches with the serial ID in the disk group configuration database of the imported disk group.

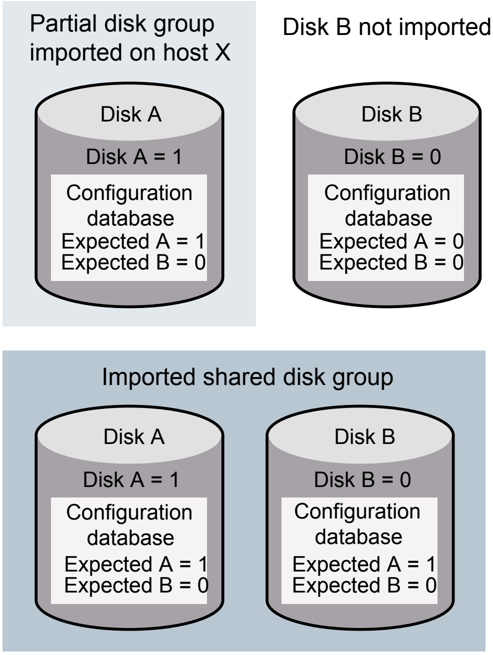
If some disks went missing from the disk group (due to physical disconnection or power failure) and those disks were imported by another host, the serial IDs for the disks in their copies of the configuration database, and also in each disk's private region, are updated separately on that host. When the disks are subsequently re-imported into the original shared disk group, the actual serial IDs on the disks do not agree with the expected values from the configuration copies on other disks in the disk group.

Depending on what happened to the different portions of the split disk group, there are two possibilities for resolving inconsistencies between the configuration databases:

- If the other disks in the disk group were not imported on another host, VxVM resolves the conflicting values of the serial IDs by using the version of the configuration database from the disk with the greatest value for the updated ID (shown as `update_id` in the output from the `vxdg list` *`diskgroup`* command).

  Figure 32-6 shows an example of a serial split brain condition that can be resolved automatically by VxVM.

Figure 32-6 Example of a serial split brain condition that can be resolved automatically



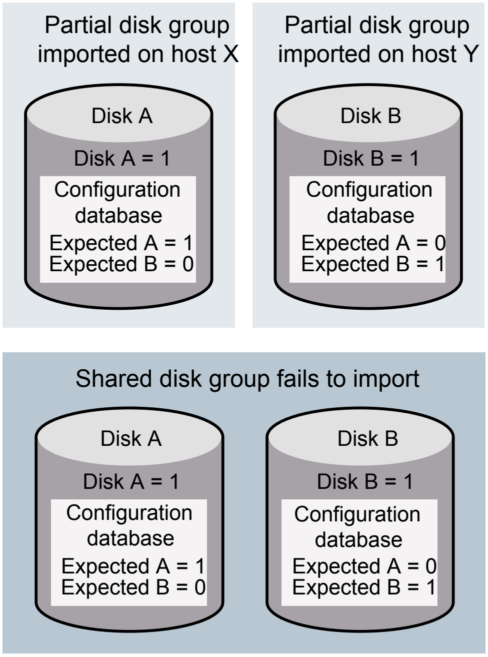
1. Disk A is imported on a separate host. Disk B is not imported. The actual and expected serial IDs are updated only on Disk A.

2. The disk group is re-imported on the cluster. The configuration copy on Disk A is used to correct the configuration copy on Disk B as the actual value of the updated ID on Disk A is the greatest.

- If the other disks were also imported on another host, no disk can be considered to have a definitive copy of the configuration database.

  Figure 32-7 shows an example of a true serial split brain condition that cannot be resolved automatically by VxVM.

Figure 32-7 Example of a true serial split brain condition that cannot be resolved automatically



1. Disks A and B are imported independently on separate hosts. The actual and expected serial IDs are updated independently on each disk.

2. The disk group cannot be re-imported on the cluster. This is because the databases to not agree on the actual and expected serial IDs. You must choose which configuration database to use.

In this case, the disk group import fails, and the `vxdg` utility outputs error messages similar to the following before exiting:

```
VxVM vxconfigd NOTICE V-5-0-33 Split Brain. da id is 0.1, while dm id
is 0.0 for DM mydg01
VxVM vxdg ERROR V-5-1-587 Disk group newdg: import failed: Serial
Split Brain detected. Run vxsplitlines
```

The import does not succeed even if you specify the `-f` flag to `vxdg`.

Although it is usually possible to resolve this conflict by choosing the version of the configuration database with the highest valued configuration ID (shown as the value of `seqno` in the output from the `vxdg list` *`diskgroup`*`| grep config` command), this may not be the correct thing to do in all circumstances.

See “Correcting conflicting configuration information” on page 708.

See “About sites and remote mirrors” on page 313.

Correcting conflicting configuration information

To resolve conflicting configuration information, you must decide which disk contains the correct version of the disk group configuration database. To assist you in doing this, you can run the `vxsplitlines` command to show the actual serial ID on each disk in the disk group and the serial ID that was expected from the configuration database. For each disk, the command also shows the `vxdg` command that you must run to select the configuration database copy on that disk as being the definitive copy to use for importing the disk group.

Note: The disk group must have a version number of at least 110.

The following is sample output from running `vxsplitlines` on the disk group `newdg`:

```
# vxsplitlines -v -g newdg

VxVM. vxsplitlines NOTICE V-0-0-0 There are 2 pools
All the disks in the first pool have the same config copies
All the disks in the second pool may not have the same config copies
```

To see the configuration copy from a disk, enter the following command:

```
# /etc/vx/diag.d/vxprivutil dumpconfig private path
```

To import the disk group with the configuration copy from a disk, enter the following command:

```
# /usr/sbin/vxdg (-s) -o selectcp=diskid import newdg

             Pool 0
DEVICE DISK DISK ID DISK PRIVATE PATH
newdg1 sdp 1215378871.300.vm2850lx13 /dev/vx/rdmp/sdp5
newdg2 sdq 1215378871.300.vm2850lx13 /dev/vx/rdmp/sdp5

              Pool 1
DEVICE DISK DISK ID DISK PRIVATE PATH
newdg3 sdo 1215378871.294.vm2850lx13 /dev/vx/rdmp/sdo5
```

If you do not specify the `-v` option, the command has the following output:

```
# vxsplitlines -g mydg listssbinfo

VxVM vxdg listssbinfo NOTICE V-0-0-0 There are 2 pools
All the disks in the first pool have the same config copies
All the disks in the second pool may not have the same config copies
```

```
Number of disks in the first pool: 1
Number of disks in the second pool: 1
```

To import the disk group with the configuration copy from the first pool, enter the following command:

```
# /usr/sbin/vxdg (-s) -o selectcp=1221451925.395.vm2850lx13 import mydg
```

To import the disk group with the configuration copy from the second pool, enter the following command:

```
# /usr/sbin/vxdg (-s) -o selectcp=1221451927.401.vm2850lx13 import mydg
```

In this example, the disk group has four disks, and is split so that two disks appear to be on each side of the split.

You can specify the `-c` option to `vxsplitlines` to print detailed information about each of the disk IDs from the configuration copy on a disk specified by its disk access name:

```
# vxsplitlines  -g newdg -c sde

DANAME(DMNAME) || Actual SSB      || Expected SSB
sdd( sdd ) || 0.1                 || 0.0 ssb ids don’t match
sde( sde ) || 0.1                 || 0.1 ssb ids match
sdf( sdf ) || 0.1                 || 0.1 ssb ids match
sdg( sdg ) || 0.1                 || 0.0 ssb ids don’t match

Please note that even though some disks ssb ids might match
that does not necessarily mean that those disks’ config copies
have all the changes. From some other configuration copies,
those disks’ ssb ids might not match. To see the configuration
from this disk, run
/etc/vx/diag.d/vxprivutil dumpconfig /dev/vx/dmp/sde
```

Based on your knowledge of how the serial split brain condition came about, you must choose one disk’s configuration to be used to import the disk group. For example, the following command imports the disk group using the configuration copy that is on side 0 of the split:

```
# /usr/sbin/vxdg -o selectcp=1045852127.32.olancha import newdg
```

When you have selected a preferred configuration copy, and the disk group has been imported, VxVM resets the serial IDs to 0 for the imported disks. The actual and expected serial IDs for any disks in the disk group that are not imported at this time remain unaltered.

Disabling a disk group

To disable a disk group, unmount and stop any volumes in the disk group, and then use the following command to deport it:

```
# vxdg deport diskgroup
```

Deporting a disk group does not actually remove the disk group. It disables use of the disk group by the system. Disks in a deported disk group can be reused, reinitialized, added to other disk groups, or imported for use on other systems. Use the `vxdg import` command to re-enable access to the disk group.

Destroying a disk group

The `vxdg` command provides a `destroy` option that removes a disk group from the system and frees the disks in that disk group for reinitialization:

```
# vxdg destroy diskgroup
```

Warning: This command destroys all data on the disks.

When a disk group is destroyed, the disks that are released can be re-used in other disk groups.

Recovering a destroyed disk group

If a disk group has been accidentally destroyed, you can recover it, provided that the disks that were in the disk group have not been modified or reused elsewhere.

To recover a destroyed disk group

1 Enter the following command to find out the disk group ID (`dgid`) of one of the disks that was in the disk group:

```
# vxdisk -s list disk_access_name
```

The disk must be specified by its disk access name, such as `sdc`. Examine the output from the command for a line similar to the following that specifies the disk group ID.

```
dgid:   963504895.1075.bass
```

2 Use the disk group ID to import the disk group:

```
# vxdg import dgid
```

Backing up and restoring disk group configuration data

The disk group configuration backup and restoration feature allows you to back up and restore all configuration data for disk groups, and for VxVM objects such as volumes that are configured within the disk groups. The `vxconfigbackupd` daemon monitors changes to the VxVM configuration and automatically records any configuration changes that occur. By default, `vxconfigbackup` stores 5 copies of the configuration backup and restoration (cbr) data. You can customize the number of cbr copies, between 1 to 5 copies.

See the `vxconfigbackupd`(1M) manual page.

VxVM provides the utilities, `vxconfigbackup` and `vxconfigrestore`, for backing up and restoring a VxVM configuration for a disk group.

See the *Veritas InfoScale Troubleshooting Guide*.

See the `vxconfigbackup`(1M) manual page.

See the `vxconfigrestore`(1M) manual page.

Backing up and restoring Flexible Storage Sharing disk group configuration data

The disk group configuration backup and restoration feature also lets you back up and restore configuration data for Flexible Storage Sharing (FSS) disk groups. The `vxconfigbackupd` daemon automatically records any configuration changes that occur on all cluster nodes. When restoring FSS disk group configuration data, you must first restore the configuration data on the secondary (slave) nodes in the cluster, which creates remote disks by exporting any locally connected disks. After restoring the configuration data on the secondary nodes, you must restore the configuration data on the primary (master) node that will import the disk group.

To back up FSS disk group configuration data

◆ To back up FSS disk group configuration data on all cluster nodes that have connectivity to at least one disk in the disk group, type the following command:

```
# /etc/vx/bin/vxconfigbackup -T diskgroup
```

To restore the configuration data for an FSS disk group

1. Identify the master node:

   ```
   # vxclustadm nidmap
   ```

2. Check if the primary node has connectivity to at least one disk in the disk group. The disk can be a direct attached storage (DAS) disk, partially shared disk, or fully shared disks.
3. If the primary node does not have connectivity to any disk in the disk group, switch the primary node to a node that has connectivity to at least one DAS or partially shared disk, using the following command:

   ```
   # vxclustadm setmaster node_name
   ```

4. Restore the configuration data on all the secondary nodes:

   ```
   # vxconfigrestore diskgroup
   ```

   Note: You must restore the configuration data on all secondary nodes that have connectivity to at least one disk in the disk group.

5. Restore the configuration data on the primary node:

   ```
   # vxconfigrestore diskgroup
   ```

6. Verify the configuration data:

   ```
   # vxprint -g diskgroup
   ```

7. If the configuration data is correct, commit the configuration:

   ```
   # vxconfigrestore -c diskgroup
   ```

To abort or decommit configuration restoration for an FSS disk group

1 Identify the master node:

```
# vxclustadm nidmap
```

2 Abort or decommit the configuration data on the master node:

```
# vxconfigrestore -d diskgroup
```

3 Abort or decommit the configuration data on all secondary nodes.

```
# vxconfigrestore -d diskgroup
```

Note: You must abort or decommit the configuration data on all secondary nodes that have connectivity to at least one disk in the disk group, and all secondary nodes from which you triggered the precommit.

See the *Veritas InfoScale 8.0 Troubleshooting Guide*.

See the `vxconfigbackup`(1M) manual page.

See the `vxconfigrestore`(1M) manual page.

Working with existing ISP disk groups

The Intelligent Storage Provisioning (ISP) feature of Veritas Volume Manager (VxVM) has been deprecated. This release does not support creating ISP disk groups. If you have existing ISP disk groups, you can import the disk groups without upgrading the disk group version. In this case, you cannot perform any operations on ISP volumes that would result in a configuration change. In addition, you cannot use any of the current release functionality that requires the upgraded disk group version.

You can upgrade an ISP disk group to the current disk group version. This operation converts all ISP volumes to standard (non-ISP) volumes and deletes ISP-specific objects. The ISP-specific objects include st pool, volume template, capability, and rules. This operation does not affect non–ISP volumes.

Note: When you upgrade the ISP disk group, all intent and storage pools information is lost. Only upgrade the disk group when this condition is acceptable.

To determine whether a disk group is an ISP disk group

- Check for the presence of storage pools, using the following command:

  ```
  # vxprint
  ```

  Sample output:

```
Disk group: mydg
TY NAME      ASSOC        KSTATE  LENGTH  PLOFFS STATE     TUTIL0 PUTIL0
dg mydg      mydg         -       -       -      ALLOC_SUP -      -

dm mydg2     ams_wms0_359 -       4120320 -      -         -      -
dm mydg3     ams_wms0_360 -       4120320 -      -         -      -

st mypool    -            -       -       -      DATA      -      -
dm mydg1     ams_wms0_358 -       4120320 -      -         -      -

v  myvol0    fsgen        ENABLED 20480   -      ACTIVE    -      -
pl myvol0-01 myvol0       ENABLED 20480   -      ACTIVE    -      -
sd mydg1-01  myvol0-01    ENABLED 20480   0      -         -      -

v  myvol1    fsgen        ENABLED 20480   -      ACTIVE    -      -
pl myvol1-01 myvol1       ENABLED 20480   -      ACTIVE    -      -
sd mydg1-02  myvol1-01    ENABLED 20480   0      -         -      -
```

  In the sample output, `st mypool` indicates that `mydg` is an ISP disk group.

To upgrade an ISP disk group

- Upgrade the ISP disk group using the following command:

  ```
  # vxdg upgrade ISP_diskgroup
  ```

To use an ISP disk group as is

- To import an ISP disk group, use the following command:

  ```
  # vxdg import ISP_diskgroup
  ```

The ISP volumes in the disk group are not allowed to make any configuration changes until the disk group is upgraded. Attempting any operations such as grow shrink, add mirror, disk group split join, etc, on ISP volumes would give the following error:

```
This disk group is a ISP disk group. Dg needs to be migrated to
non-ISP dg to allow any configuration changes. Please upgrade
the dg to perform the migration.
```

Note: Non-ISP or VxVM volumes in the ISP disk group are not affected.

Operations that still work on ISP disk group without upgrading:

- Setting, removing, and replacing volume tags.
- Renaming of any VxVM objects such as volume, dg, plex, etc.
- Plex attach and detach.
- The vxconfigbackup and vxconfigrestore command can be used at the cost of losing any intent information

Managing plexes and subdisks

This section describes managing plexes and subdisks.

A subdisk is a set of contiguous disk blocks. VxVM allocates disk space using subdisks.

A plex is a logical groupings of subdisks that creates an area of disk space independent of physical disk size or other restrictions. Replication (mirroring) of disk data is set up by creating multiple data plexes for a single volume. Each data plex in a mirrored volume contains an identical copy of the volume data.

A plex becomes a participating plex for a volume when it is attached to a volume. Attaching a plex associates it with the volume and enables the plex for use.

Reattaching plexes

When a mirror plex encounters irrecoverable errors, Veritas Volume Manager (VxVM) detaches the plex from the mirrored volume. An administrator may also detach a plex manually using a utility such as vxplex or vxassist. In order to use a plex that was previously attached to a volume, the plex must be reattached to the volume. The reattach operation also ensures that the plex mirror is resynchronized to the other plexes in the volume.

See “Plex synchronization” on page 718.

The following methods are available for reattaching plexes:

- By default, VxVM automatically reattaches the affected mirror plexes when the underlying failed disk or LUN becomes visible. When VxVM detects that the

device is online, VxVM automatically recovers the volume components on the involved LUN. VxVM resynchronizes the plex and the mirror becomes available.
See "Automatic plex reattachment" on page 716.

- If the automatic reattachment feature is disabled, you need to reattach the plexes manually. You may also need to manually reattach the plexes for devices that are not automatically reattached. For example, VxVM does not automatically reattach plexes on site-consistent volumes.
  See "Reattaching a plex manually" on page 717.

Automatic plex reattachment

When a mirror plex encounters irrecoverable errors, Veritas Volume Manager (VxVM) detaches the plex from the mirrored volume. By default, VxVM automatically reattaches the affected mirror plexes when the underlying failed disk or LUN becomes visible. When VxVM detects that the device is online, the VxVM volume components on the involved LUN are automatically recovered, and the mirrors become usable.

VxVM uses the DMP failed LUN probing to detect when the device has come online. The timing for a reattach depends on the `dmp_restore_interval`, which is a tunable parameter. The number of LUNs that have reconnected may also affect the time required before the plex is reattached.

VxVM does not automatically reattach plexes on site-consistent volumes.

When VxVM is installed or the system reboots, VxVM starts the `vxattachd` daemon. The `vxattachd` daemon handles automatic reattachment for both plexes and sites. The`vxattachd` daemon also initiates the resynchronization process for a plex. After a plex is successfully reattached, `vxattachd` notifies root.

To disable automatic plex attachment, remove `vxattachd` from the start up scripts. Disabling `vxattachd` disables the automatic reattachment feature for both plexes and sites.

In a Cluster Volume Manager (CVM) the following considerations apply:

- If the global detach policy is set, a storage failure from any node causes all plexes on that storage to be detached globally. When the storage is connected back to any node, the `vxattachd` daemon triggers reattaching the plexes on the master node only.

- The automatic reattachment functionality is local to a node. When enabled on a node, all of the disk groups imported on the node are monitored. If the automatic reattachment functionality is disabled on a master node, the feature is disable on all shared disk groups and private disk groups imported on the master node.

- The vxattachd daemon listens for "dmpnode online" events using vxnotify to trigger its operation. Therefore, an automatic reattachment is not triggered if the dmpnode online event is not generated when vxattachd is running. The following are typical examples:
  - Storage is reconnected before vxattachd is started; for example, during reboot.
  - In CVM, with active/passive arrays, if all nodes cannot agree on a common path to an array controller, a plex can get detached due to I/O failure. In these cases, the dmpnode will not get disabled. Therefore, after the connections are restored, a dmpnode online event is not generated and automatic plex reattachment is not triggered.

These CVM considerations also apply to automatic site reattachment.

See "Automatic site reattachment" on page 329.

Reattaching a plex manually

This section describes how to reattach plexes manually if automatic reattachment feature is disabled. This procedure may also be required for devices that are not automatically reattached. For example, VxVM does not automatically reattach plexes on site-consistent volumes.

When a disk has been repaired or replaced and is again ready for use, the plexes must be put back online (plex state set to ACTIVE). To set the plexes to ACTIVE, use one of the following procedures depending on the state of the volume.

- If the volume is currently ENABLED, use the following command to reattach the plex:

  ```
  # vxplex [-g diskgroup] att volume plex ...
  ```

  For example, for a plex named vol01-02 on a volume named vol01 in the disk group, mydg, use the following command:

  ```
  # vxplex -g mydg att vol01 vol01-02
  ```

  As when returning an OFFLINE plex to ACTIVE, this command starts to recover the contents of the plex and, after the recovery is complete, sets the plex utility state to ACTIVE.

- If the volume is not in use (not ENABLED), use the following command to re-enable the plex for use:

  ```
  # vxmend [-g diskgroup] on plex
  ```

  For example, to re-enable a plex named vol01-02 in the disk group, mydg, enter:

```
# vxmend -g mydg on vol01-02
```

In this case, the state of `vol01-02` is set to `STALE`. When the volume is next started, the data on the plex is revived from another plex, and incorporated into the volume with its state set to `ACTIVE`.

If the `vxinfo` command shows that the volume is unstartable, set one of the plexes to `CLEAN` using the following command:

```
# vxmend [-g diskgroup] fix clean plex
```

Start the volume using the following command:

```
# vxvol [-g diskgroup] start volume
```

See the *Veritas InfoScale Troubleshooting Guide*.

Plex synchronization

Each plex or mirror of a volume is a complete copy of the data. When a plex is attached to a volume, the data in the plex must be synchronized with the data in the other plexes in the volume. The plex that is attached may be a new mirror or a formerly attached plex. A new mirror must be fully synchronized. A formerly attached plex only requires the changes that were applied since the plex was detached.

The following operations trigger a plex synchronization:

- Moving or copying a subdisk with the `vxsd` command. The operation creates a temporary plex that is synchronized with the original subdisk.
- Adding a mirror with the `vxassist mirror` command.
- Creating a volume with a mirror with the `vxassist make` command.
- Manually reattaching a plex with the `vxplex att` command.
- Recovering a volume with the `vxrecover` command.
- Adding a mirror to a snapshot with the `vxsnap addmir` command.
- Reattaching or restoring a snapshot with the `vxsnap` command.

Plex synchronization can be a long-running operation, depending on the size of the volume and the amount of data that needs to be synchronized. Veritas Volume Manager provides several features to improve the efficiency of synchronizing the plexes.

- FastResync
  If the FastResync feature is enabled, VxVM maintains a FastResync map on the volume. VxVM uses the FastResync map to apply only the updates that the

mirror has missed. This behavior provides an efficient way to resynchronize the plexes.

- SmartMove

  The SmartMove™ feature reduces the time and I/O required to attach or reattach a plex to a VxVM volume with a mounted VxFS file system. The SmartMove feature uses the VxFS information to detect free extents and avoid copying them.

  When the SmartMove feature is on, less I/O is sent through the host, through the storage network and to the disks or LUNs. The SmartMove feature can be used for faster plex creation and faster array migrations.

- Recovery for synchronization tasks

  In this release, VxVM tracks the plex synchronization for the following commands: `vxplex att`, `vxassist mirror`, `vxsnap addmir`, `vxsnap reattach`, and `vxsnap restore`. If the system crashes or the `vxconfigd` daemon fails, VxVM provides automatic recovery for the synchronization task. When the system is recovered, VxVM restarts the synchronization from the point where it failed. The synchronization occurs in the background, so the volume is available without delay.

Decommissioning storage

This section describes how you remove disks and volumes from VxVM.

Removing a volume

If a volume is inactive or its contents have been archived, you may no longer need it. In that case, you can remove the volume and free up the disk space for other uses.

To remove a volume

1. Remove all references to the volume by application programs, including shells, that are running on the system.
2. If the volume is mounted as a file system, unmount it with the following command:

   ```
   # umount /dev/vx/dsk/diskgroup/volume
   ```

3. If the volume is listed in the `/etc/fstab` file, edit this file and remove its entry. For more information about the format of this file and how you can modify it, see your operating system documentation.

4 Stop all activity by VxVM on the volume with the following command:

```
# vxvol [-g diskgroup] stop volume
```

5 Remove the volume using the `vxassist` command as follows:

```
# vxassist [-g diskgroup] remove volume volume
```

You can also use the `vxedit` command to remove the volume as follows:

```
# vxedit [-g diskgroup] [-r] [-f] rm volume
```

The `-r` option to `vxedit` indicates recursive removal. This command removes all the plexes that are associated with the volume and all subdisks that are associated with the plexes. The `-f` option to `vxedit` forces removal. If the volume is still enabled, you must specify this option.

Removing a disk from VxVM control

After removing a disk from a disk group, you can permanently remove it from Veritas Volume Manager control.

Warning: The `vxdiskunsetup` command removes a disk from Veritas Volume Manager control by erasing the VxVM metadata on the disk. To prevent data loss, any data on the disk should first be evacuated from the disk. The `vxdiskunsetup` command should only be used by a system administrator who is trained and knowledgeable about Veritas Volume Manager.

To remove a disk from VxVM control

◆ Type the following command:

```
# /usr/lib/vxvm/bin/vxdiskunsetup sdx
```

See the `vxdiskunsetup`(1m) manual page.

About shredding data

When you decommission a disk that contained sensitive data, you may need to destroy any remaining data on the disk. Simply deleting the data may not adequately protect the confidential and secure data. In addition to deleting the data, you want to prevent the possibility that hackers can recover any information that is stored on the disks. Regulatory standards require that the confidential and secure data is sanitized or erased using a method such as overwriting the data with a digital pattern. Veritas Volume Manager (VxVM) provides the disk shred operation, which overwrites all of the addressable blocks with a digital pattern in one, three, or seven passes.

Caution: All data in the volume will be lost when you shred it. Make sure that the information has been backed up onto another storage medium and verified, or that it is no longer needed.

VxVM provides the ability to shred the data on the disk to minimize the chance that the data is recoverable. When you specify the disk shred operation, VxVM shreds the entire disk, including any existing disk labels. After the shred operation, VxVM writes a new empty label on the disk to prevent the disk from going to the error state. The VxVM shred operation provides the following methods of overwriting a disk:

- One-pass algorithm

  VxVM overwrites the disk with a randomly-selected digital pattern. This option takes the least amount of time. The default type is the one-pass algorithm.
- Three-pass algorithm

  VxVM overwrites the disk a total of three times. In the first pass, VxVM overwrites the data with a pre-selected digital pattern. The second time, VxVM overwrites the data with the binary complement of the pattern. In the last pass, VxVM overwrites the disk with a randomly-selected digital pattern.
- Seven-pass algorithm

  VxVM overwrites the disk a total of seven times. In each pass, VxVM overwrites the data with a randomly-selected digital pattern or with the binary complement of the previous pattern.

VxVM does not currently support shredding of thin-reclaimable LUNs. If you attempt to start the shred operation on a thin-reclaimable disk, VxVM displays a warning message and skips the disk.

You can skip the shred operation for disks that contain only encrypted volumes.

Shredding a VxVM disk

When you decommission a Veritas Volume Manager (VxVM) disk that contains sensitive data, VxVM provides the ability to shred the data on the disk.

Note the following requirements:

- VxVM does not shred a disk that is in use by VxVM on this system or in a shared disk group.

- VxVM does not currently support shredding of thin-reclaimable LUNs. If you attempt to start the shred operation on a thin-reclaimable disk, VxVM displays a warning message and skips the disk.
- VxVM does not shred a disk that is not a VxVM disk.
- VxVM does not shred a disk that is mounted.
- Veritas does not recommend shredding solid state drives (SSDs). To shred SSD devices, use the shred operation with the force (-f) option.

See “About shredding data” on page 720.

Caution: All data on the disk will be lost when you shred the disk. Make sure that the information has been backed up onto another storage medium and verified, or that it is no longer needed.

To shred a VxVM disk

1 To shred the disk:

```
# /etc/vx/bin/vxdiskunsetup [-Cf] -o shred[=1|3|7] disk...
```

Where:

The force option (-f) permits you to shred Solid State Drives (SSDs).

1, 3 and 7 are the shred options corresponding to the number of passes. The default number of passes is 1.

disk... represents one or more disk names. If you specify multiple disk names, the `vxdiskunsetup` command processes them sequentially, one at a time.

For example:

```
# /etc/vx/bin/vxdiskunsetup -o shred=3 hds9970v0_14
disk_shred: Shredding disk hds9970v0_14 with type 3
disk_shred: Disk raw size 2097807360 bytes
disk_shred: Writing 32010 (65536 byte size) pages and 0 bytes
to disk
disk_shred: Wipe Pass 0: Pattern 0x3e
disk_shred: Wipe Pass 1: Pattern 0xca
disk_shred: Wipe Pass 2: Pattern 0xe2
disk_shred: Shred passed random verify of 131072 bytes at
offset 160903168
```

The `vxdiskunsetup shred` command sets up a new task.

2 You can monitor the progress of the shred operation with the `vxtask` command.

For example:

```
# vxtask list
TASKID PTID TYPE/STATE  PCT    PROGRESS
   203   -  DISKSHRED/R 90.16% 0/12291840/11081728 DISKSHRED
nodg nodg
```

You can pause, abort, or resume the shred task. You cannot throttle the shred task.

See `vxtask(1m)`

3 If the disk shred operation fails, the disk may go into an error state with no label.

See “Failed disk shred operation results in a disk with no label” on page 724.

Failed disk shred operation results in a disk with no label

The disk shred operation destroys the label for the disk and recreates the label. If the shred operation aborts in the middle or the system crashes, the disk might go in an error state with no label.

To correct the error state of the disk

1. Create a new label manually or reinitialize the disk under VxVM using the following command:

   ```
   # /etc/vx/bin/vxdisksetup -i disk
   ```

2. Start the shred operation. If the disk shows as a non-VxVM disk, reinitialize the disk with the vxdisksetup command in step 1, then restart the shred operation.

   ```
   # /etc/vx/bin/vxdiskunsetup [-Cf] -o shred[=1|3|7] disk...
   ```

Removing and replacing disks

A replacement disk should have the same disk geometry as the disk that failed. That is, the replacement disk should have the same bytes per sector, sectors per track, tracks per cylinder and sectors per cylinder, same number of cylinders, and the same number of accessible cylinders.

Note: You may need to run commands that are specific to the operating system or disk array before removing a physical disk.

If failures are starting to occur on a disk, but the disk has not yet failed completely, you can replace the disk. This involves detaching the failed or failing disk from its disk group, followed by replacing the failed or failing disk with a new one. Replacing the disk can be postponed until a later date if necessary.

If removing a disk causes a volume to be disabled, you can restart the volume so that you can restore its data from a backup.

See the *Veritas InfoScale Troubleshooting Guide*.

To replace a disk

1. Select `Remove a disk for replacement` from the `vxdiskadm` main menu.
2. At the following prompt, enter the name of the disk to be replaced (or enter `list` for a list of disks):

   ```
   Enter disk name [<disk>,list,q,?] mydg02
   ```

3 When you select a disk to remove for replacement, all volumes that are affected by the operation are displayed, for example:

```
VxVM NOTICE V-5-2-371 The following volumes will lose mirrors
as a result of this operation:

home src

No data on these volumes will be lost.

The following volumes are in use, and will be disabled as a
result of this operation:

mkting

Any applications using these volumes will fail future
accesses. These volumes will require restoration from backup.

Are you sure you want do this? [y,n,q,?] (default: n)
```

To remove the disk, causing the named volumes to be disabled and data to be lost when the disk is replaced, enter `y` or press Return.

To abandon removal of the disk, and back up or move the data associated with the volumes that would otherwise be disabled, enter `n` or `q` and press Return.

For example, to move the volume `mkting` to a disk other than `mydg02`, use the following command.

The ! character is a special character in some shells. The following example shows how to escape it in a bash shell.

```
# vxassist move mkting \!mydg02
```

After backing up or moving the data in the volumes, start again from step 1.

4. At the following prompt, either select the device name of the replacement disk (from the list provided), press Return to choose the default disk, or enter `none` if you are going to replace the physical disk:

   ```
   The following devices are available as replacements:
   sdb

   You can choose one of these disks now, to replace mydg02.
   Select none if you do not wish to select a replacement disk.

   Choose a device, or select none
   [<device>,none,q,?] (default: sdb)
   ```

   Do not choose the old disk drive as a replacement even though it appears in the selection list. If necessary, you can choose to initialize a new disk.

   You can enter `none` if you intend to replace the physical disk.

   See "Replacing a failed or removed disk" on page 727.

5. If you chose to replace the disk in step 4, press Return at the following prompt to confirm this:

   ```
   VxVM NOTICE V-5-2-285 Requested operation is to remove mydg02
   from group mydg. The removed disk will be replaced with disk device
   sdb.  Continue with operation? [y,n,q,?] (default: y)
   ```

   `vxdiskadm` displays the following messages to indicate that the original disk is being removed:

   ```
   VxVM NOTICE V-5-2-265 Removal of disk mydg02 completed
   successfully.
   VxVM NOTICE V-5-2-260 Proceeding to replace mydg02 with device
   sdb.
   ```

6. You can now choose whether the disk is to be formatted as a CDS disk that is portable between different operating systems, or as a non-portable sliced or simple disk:

   ```
   Enter the desired format [cdsdisk,sliced,simple,q,?]
   (default: cdsdisk)
   ```

   Enter the format that is appropriate for your needs. In most cases, this is the default format, `cdsdisk`.

7. At the following prompt, vxdiskadm asks if you want to use the default private region size of 65536 blocks (32 MB). Press Return to confirm that you want to use the default value, or enter a different value. (The maximum value that you can specify is 524288 blocks.)

   ```
   Enter desired private region length [<privlen>,q,?]
   (default: 65536)
   ```

8. If one of more mirror plexes were moved from the disk, you are now prompted whether FastResync should be used to resynchronize the plexes:

   ```
   Use FMR for plex resync? [y,n,q,?] (default: n) y
   vxdiskadm displays the following success message:
   VxVM NOTICE V-5-2-158 Disk replacement completed successfully.
   ```

9. At the following prompt, indicate whether you want to remove another disk (y) or return to the vxdiskadm main menu (n):

   ```
   Remove another disk? [y,n,q,?] (default: n)
   ```

   It is possible to move hot-relocate subdisks back to a replacement disk.

   See "Configuring hot-relocation to use only spare disks" on page 608.

Replacing a failed or removed disk

The following procedure describes how to replace a failed or removed disk.

To specify a disk that has replaced a failed or removed disk

1. Select Replace a failed or removed disk from the vxdiskadm main menu.
2. At the following prompt, enter the name of the disk to be replaced (or enter list for a list of disks):

   ```
   Select a removed or failed disk [<disk>,list,q,?] mydg02
   ```

3. The `vxdiskadm` program displays the device names of the disk devices available for use as replacement disks. Your system may use a device name that differs from the examples. Enter the device name of the disk or press Return to select the default device:

   ```
   The following devices are available as replacements:
   sdb sdk

   You can choose one of these disks to replace mydg02.
   Choose "none" to initialize another disk to replace mydg02.

   Choose a device, or select "none"
   [<device>,none,q,?] (default: sdb)
   ```

4. Depending on whether the replacement disk was previously initialized, perform the appropriate step from the following:

   - If the disk has not previously been initialized, press Return at the following prompt to replace the disk:

     ```
     VxVM INFO V-5-2-378 The requested operation is to initialize
     disk device sdb and to then use that device to
     replace the removed or failed disk mydg02 in disk group mydg.
     Continue with operation? [y,n,q,?] (default: y)
     ```

   - If the disk has already been initialized, press Return at the following prompt to replace the disk:

     ```
     VxVM INFO V-5-2-382 The requested operation is to use the
     initialized device sdb to replace the removed or
     failed disk mydg02 in disk group mydg.
     Continue with operation? [y,n,q,?] (default: y)
     ```

5. You can now choose whether the disk is to be formatted as a CDS disk that is portable between different operating systems, or as a non-portable sliced or simple disk:

   ```
   Enter the desired format [cdsdisk,sliced,simple,q,?]
   (default: cdsdisk)
   ```

   Enter the format that is appropriate for your needs. In most cases, this is the default format, `cdsdisk`.

6 At the following prompt, `vxdiskadm` asks if you want to use the default private region size of 65536 blocks (32 MB). Press Return to confirm that you want to use the default value, or enter a different value. (The maximum value that you can specify is 524288 blocks.)

```
Enter desired private region length [<privlen>,q,?]
(default: 65536)
```

7 The `vxdiskadm` program then proceeds to replace the disk, and returns the following message on success:

```
VxVM NOTICE V-5-2-158 Disk replacement completed successfully.
```

At the following prompt, indicate whether you want to replace another disk (`y`) or return to the `vxdiskadm` main menu (`n`):

```
Replace another disk? [y,n,q,?] (default: n)
```

Rootability

This chapter includes the following topics:

Root Disk Encapsulation (RDE) is not supported

Root Disk Encapsulation (RDE) on Linux Distribution is not supported from 7.3.1. release onwards.

Encapsulating a disk

Warning: Encapsulating a disk requires that the system be rebooted several times. Schedule performance of this procedure for a time when this does not inconvenience users.

This section describes how to encapsulate a disk for use in VxVM. Encapsulation preserves any existing data on the disk when the disk is placed under VxVM control.

A root disk can be encapsulated and brought under VxVM control. However, there are restrictions on the layout and configuration of root disks that can be encapsulated.

See "Restrictions on using rootability with Linux" on page 738.

See "Rootability" on page 737.

Use the `format` or `fdisk` commands to obtain a printout of the root disk partition table before you encapsulate a root disk. For more information, see the appropriate manual pages. You may need this information should you subsequently need to recreate the original root disk.

You cannot grow or shrink any volume (`rootvol`, `usrvol`, `varvol`, `optvol`, `swapvol`, and so on) that is associated with an encapsulated root disk. This is because these volumes map to physical partitions on the disk, and these partitions must be contiguous.

Disks with `msdos` disk labels can be encapsulated as `auto:sliced` disks provided that they have at least one spare primary partition that can be allocated to the public region, and one spare primary or logical partition that can be allocated to the private region.

Disks with `sun` disk labels can be encapsulated as `auto:sliced` disks provided that they have at least two spare slices that can be allocated to the public and private regions.

Extensible Firmware Interface (EFI) disks with gpt (GUID Partition Table) labels can be encapsulated as `auto:sliced` disks provided that they have at least two spare slices that can be allocated to the public and private regions.

The entry in the partition table for the public region does not require any additional space on the disk. Instead it is used to represent (or encapsulate) the disk space that is used by the existing partitions.

Unlike the public region, the partition for the private region requires a small amount of space at the beginning or end of the disk that does not belong to any existing partition or slice. By default, the space required for the private region is 32MB, which is rounded up to the nearest whole number of cylinders. On most modern disks, one cylinder is usually sufficient.

To encapsulate a disk for use in VxVM

1 Before encapsulating a root disk, set the device naming scheme used by VxVM to be persistent.

```
# vxddladm set namingscheme={osn|ebn} persistence=yes
```

For example, to use persistent naming with enclosure-based naming:

```
# vxddladm  set namingscheme=ebn persistence=yes
```

2 Select `Encapsulate one or more disks` from the `vxdiskadm` main menu.

Your system may use device names that differ from the examples shown here.

At the following prompt, enter the disk device name for the disks to be encapsulated:

```
Select disk devices to encapsulate:
[<pattern-list>,all,list,q,?] device name
```

The *pattern-list* can be a single disk, or a series of disks. If *pattern-list* consists of multiple items, those items must be separated by white space.

If you do not know the address (device name) of the disk to be encapsulated, enter `l` or `list` at the prompt for a complete listing of available disks.

3 To continue the operation, enter `y` (or press Return) at the following prompt:

```
Here is the disk selected. Output format: [Device]
device name

Continue operation? [y,n,q,?] (default: y) y
```

4 Select the disk group to which the disk is to be added at the following prompt:

```
You can choose to add this disk to an existing disk group or to
a new disk group. To create a new disk group, select a disk
group name that does not yet exist.

Which disk group [<group>,list,q,?]
```

5 At the following prompt, either press Return to accept the default disk name or enter a disk name:

```
Use a default disk name for the disk? [y,n,q,?] (default: y)
```

6. To continue with the operation, enter `y` (or press Return) at the following prompt:

   ```
   The selected disks will be encapsulated and added to the
   disk group name disk group with default disk names.

   device name

   Continue with operation? [y,n,q,?] (default: y) y
   ```

7. To confirm that encapsulation should proceed, enter `y` (or press Return) at the following prompt:

   ```
   The following disk has been selected for encapsulation.
    Output format: [Device]

   device name

   Continue with encapsulation? [y,n,q,?] (default: y) y
   ```

   A message similar to the following confirms that the disk is being encapsulated for use in VxVM:

   ```
   The disk device device name will be encapsulated and added to
   the disk group diskgroup with the disk name diskgroup01.
   ```

8. For non-root disks, you can now choose whether the disk is to be formatted as a CDS disk that is portable between different operating systems, or as a non-portable sliced disk:

   ```
   Enter the desired format [cdsdisk,sliced,simple,q,?]
   (default: cdsdisk)
   ```

   Enter the format that is appropriate for your needs. In most cases, this is the default format, `cdsdisk`. Note that only the `sliced` format is suitable for use with root, boot or swap disks.

9. At the following prompt, `vxdiskadm` asks if you want to use the default private region size of 65536 blocks (32MB). Press Return to confirm that you want to use the default value, or enter a different value. (The maximum value that you can specify is 524288 blocks.)

   ```
   Enter desired private region length [<privlen>,q,?]
   (default: 65536)
   ```

10 If you entered `cdsdisk` as the format in step 8, you are prompted for the action to be taken if the disk cannot be converted this format:

```
Do you want to use sliced as the format should cdsdisk
fail? [y,n,q,?] (default: y)
```

If you enter `y`, and it is not possible to encapsulate the disk as a CDS disk, it is encapsulated as a sliced disk. Otherwise, the encapsulation fails.

11 `vxdiskadm` then proceeds to encapsulate the disks. You should now reboot your system at the earliest possible opportunity, for example by running this command:

```
# shutdown -r now
```

The `/etc/fstab` file is updated to include the volume devices that are used to mount any encapsulated file systems. You may need to update any other references in backup scripts, databases, or manually created swap devices. The original `/etc/fstab` file is saved as `/etc/fstab.b4vxvm`

12 At the following prompt, indicate whether you want to encapsulate more disks (`y`) or return to the `vxdiskadm` main menu (`n`):

```
Encapsulate other disks? [y,n,q,?] (default: n) n
```

The default layout that is used to encapsulate disks can be changed.

Failure of disk encapsulation

Under some circumstances, encapsulation of a disk can fail because there is not enough free space available on the disk to accommodate the private region. If there is insufficient free space , the encapsulation process ends abruptly with an error message similar to the following:

```
VxVM ERROR V-5-2-338 The encapsulation operation failed with the
following error:
It is not possible to encapsulate device, for the following
reason:
<VxVM vxslicer ERROR V-5-1-1108 Unsupported disk layout.>
```

One solution is to configure the disk with the `nopriv` format.

See “Using nopriv disks for encapsulation” on page 735.

Using nopriv disks for encapsulation

Encapsulation converts existing partitions on a specified disk to volumes. If any partitions contain file systems, their /etc/fstab entries are modified so the file systems are mounted on volumes instead.

Disk encapsulation requires that enough free space be available on the disk (by default, 32 megabytes) for storing the private region that VxVM uses for disk identification and configuration information. This free space cannot be included in any other partitions.

See the vxencap(1M) manual page.

You can encapsulate a disk that does not have space available for the VxVM private region partition by using the vxdisk utility. To do this, configure the disk as a nopriv device that does not have a private region.

The drawback with using nopriv devices is that VxVM cannot track changes in the address or controller of the disk. Normally, VxVM uses identifying information stored in the private region on the physical disk to track changes in the location of a physical disk. Because nopriv devices do not have private regions and have no identifying information stored on the physical disk, tracking cannot occur.

One use of nopriv devices is to encapsulate a disk so that you can use VxVM to move data off the disk. When space has been made available on the disk, remove the nopriv device, and encapsulate the disk as a standard disk device.

A disk group cannot be formed entirely from nopriv devices. This is because nopriv devices do not provide space for storing disk group configuration information. Configuration information must be stored on at least one disk in the disk group.

Creating a nopriv disk for encapsulation

Warning: Do not use nopriv disks to encapsulate a root disk. If insufficient free space exists on the root disk for the private region, part of the swap area can be used instead.

To create a nopriv disk for encapsulation

1. If it does not exist already, set up a partition on the disk for the area that you want to access using VxVM.
2. Use the following command to map a VM disk to the partition:

   ```
   # vxdisk define partition-device type=nopriv
   ```

   where *partition-device* is the basename of the device in the `/dev/dsk` directory.

   For example, to map partition `3` of disk device `sdc`, use the following command:

   ```
   # vxdisk define sdc3 type=nopriv
   ```

Creating volumes for other partitions on a nopriv disk

To create volumes for other partitions on a nopriv disk

1. Add the partition to a disk group.
2. Determine where the partition resides within the encapsulated partition.
3. If no data is to be preserved on the partition, use `vxassist` to create a volume with the required length.

   Warning: By default, `vxassist` re-initializes the data area of a volume that it creates. If there is data to be preserved on the partition, do not use `vxassist`. Instead, create the volume with `vxmake` and start the volume with the command `vxvol init active`.

Device name format changes in RHEL 7 environments after encapsulation

With RHEL 7 and supported RHEL compatible distributions, the format of volumes in the `/etc/fstab` file after root disk encapsulation has changed.

Table 33-1 lists the changes in RHEL 7 and supported RHEL compatible distribution environments.

Table 33-1 Volume formats changes in RHEL 7 and supported RHEL compatible distribution environments

Before RHEL 7	With RHEL 7
Volume format: `/dev/vx/dsk/bootdg/<volume>`	Volume format: `/dev/vx_dsk_bootdg_<volume>`

Contents of `/etc/fstab` file where the rootdisk has two partitions, namely, `/` and `swap`:
`# cat /etc/fstab`
 	
`/dev/vx/dsk/bootdg/rootvol \`
`/    ext4 defaults 1 1`
`/dev/vx/dsk/bootdg/swapvol \`
`swap swap defaults 0 0`
 	
`#NOTE: volume rootvol (/) \`
`encapsulated partition sda1`
`#NOTE: volume swapvol (swap) \`
`encapsulated partition sda2`
	Contents of `/etc/fstab` file where the rootdisk has two partitions, namely, `/` and `swap`:
	`# cat /etc/fstab`
	
	`/dev/vx_dsk_bootdg_rootvol \`
	`/    ext4 defaults 1 1`
	`/dev/vx_dsk_bootdg_swapvol \`
	`swap swap defaults 0 0`
	
	`#NOTE: volume rootvol (/) \`
	`encapsulated partition sda1`
	`#NOTE: volume swapvol (swap) \`
	`encapsulated partition sda2`

Note: Though the format of the device names in the `/etc/fstab` has changed, there is no change in the output of the `mount` utility. The `mount` utility still displays the mounted volumes in the old format.

Rootability

VxVM can place various files from the root file system, `swap` device, and other file systems on the root disk under VxVM control. This is called rootability. The root disk (that is, the disk containing the root file system) can be put under VxVM control through the process of encapsulation.

Encapsulation converts existing partitions on that disk to volumes. Once under VxVM control, the `root` and `swap` devices appear as volumes and provide the same characteristics as other VxVM volumes. A volume that is configured for use as a swap area is referred to as a swap volume, and a volume that contains the root file system is referred to as a root volume.

Note: Only encapsulate your root disk if you also intend to mirror it. There is no benefit in root-disk encapsulation for its own sake.

You can mirror the `rootvol`, and `swapvol` volumes, as well as other parts of the root disk that are required for a successful boot of the system (for example, `/usr`). This provides complete redundancy and recovery capability in the event of disk failure. Without mirroring, the loss of the `root`, `swap`, or `usr` partition prevents the system from being booted from surviving disks.

Mirroring disk drives that are critical to booting ensures that no single disk failure renders the system unusable. A suggested configuration is to mirror the critical disk onto another available disk (using the `vxdiskadm` command). If the disk containing `root` and `swap` partitions fails, the system can be rebooted from a disk containing mirrors of these partitions.

Recovering a system after the failure of an encapsulated root disk requires the application of special procedures.

See the *Veritas InfoScale Troubleshooting Guide*.

Restrictions on using rootability with Linux

Bootable root disks with `msdos` disk labels can contain up to four primary partitions: `/dev/sdx1` through `/dev/sdx4` for SCSI disks, and `/dev/hdx1` through `/dev/hdx4` for IDE disks. If more than four partitions are required, a primary partition can be configured as an extended partition that contains up to 11 logical partitions (`/dev/sdx5` through`/dev/sdx15`) for SCSI disks and 12 logical partitions (`/dev/hdx5` through`/dev/sdx16`) for IDE disks.

Note: Extensible Firmware Interface (EFI) disks with GUID Partition Table (GPT) labels are not supported for root encapsulation.

To encapsulate a root disk, VxVM requires one unused primary partition entry to represent the public region, plus one unused primary partition or one unused logical partition for the private region.

The entry in the partition table for the public region does not require any additional space on the disk. Instead it is used to represent (or encapsulate) the disk space that is used by the existing partitions.

Unlike the public region, the partition for the private region requires a relatively small amount of disk space. By default, the space required for the private region is 32MB, which is rounded up to the nearest whole number of cylinders. On most modern disks, one cylinder is usually sufficient.

To summarize, the requirements for the partition layout of a root disk that can be encapsulated are:

- One unused primary partition entry for the public region.
- Free disk space or a swap partition, from which space can be allocated to the private region. If the free space or swap partition is not located within an extended partition, one unused primary partition entry is required for the private region. Otherwise, one unused logical partition entry is required.

The following error message is displayed by the vxencap or vxdiskadm commands if you attempt to encapsulate a root disk that does not have the required layout:

```
Cannot find appropriate partition layout to allocate space
for VxVM public/private partitions.
```

The following sections show examples of root disk layouts for which encapsulation is either supported or not supported.

- See “Sample supported root disk layouts for encapsulation” on page 740.
- See “Sample unsupported root disk layouts for encapsulation” on page 743.

Note the following additional important restrictions on using rootability with Linux:

- Root disk encapsulation is only supported for devices with standard SCSI or IDE interfaces. It is not supported for most devices with vendor-proprietary interfaces, except the COMPAQ SMART and SMARTII controllers, which use device names of the form /dev/ida/cXdXpX and /dev/cciss/cXdXpX.
- Root disk encapsulation is only supported for disks with msdos or sun labels. It is not supported for disks with gpt labels.
- The root, boot, and swap partitions must be on the same disk.
- Either the GRUB or the LILO boot loader must be used as the boot loader for SCSI and IDE disks.
- The menu entries in the boot loader configuration file must be valid.
- The boot loader configuration file must not be edited during the root encapsulation process.
- The /boot partition must be on the first disk as seen by the BIOS, and this partition must be a primary partition.

  Some systems cannot be configured to ignore local disks. The local disk needs to be removed when encapsulating. Multi-pathing configuration changes (for multiple HBA systems) can have the same effect. VxVM supports only those systems where the initial bootstrap installation configuration has not been changed for root encapsulation.

- The boot loader must be located in the master boot record (MBR) on the root disk or any root disk mirror.
- If the GRUB boot loader is used, the `root` device location of the `/boot` directory must be set to the first disk drive, `sd0` or `hd0`, to allow encapsulation of the root disk.
- If the LILO or ELILO boot loader is used, do not use the `FALLBACK`, `LOCK` or `-R` options after encapsulating the root disk.

> **Warning:** Using the `FALLBACK`, `LOCK` or `-R` options with `LILO` may render your system unbootable because `LILO` does not understand the layout of VxVM volumes.

- Booting from an encapsulated root disk which is connected only to the secondary controller in an A/P (Active/Passive) array is not supported.
- The default Red Hat installation layout is not valid for implementing rootability. If you change the layout of your root disk, ensure that the root disk is still bootable before attempting to encapsulate it.
  See "Example 1: unsupported root disk layouts for encapsulation" on page 743.
- Do not allocate volumes from the root disk after it has been encapsulated. Doing so may destroy partition information that is stored on the disk.
- The device naming scheme must be set to persistent.

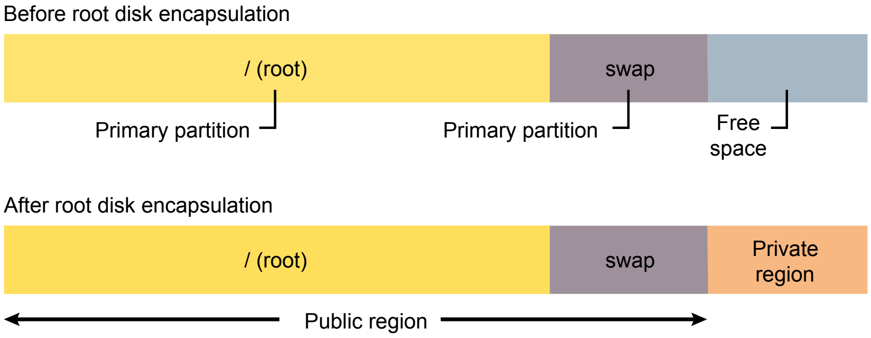
Sample supported root disk layouts for encapsulation

The following examples show root disk layouts that support encapsulation.

Example 1: supported root disk layouts for encapsulation

Figure 33-1 shows an example of a supported layout with `root` and `swap` configured on two primary partitions, and some existing free space on the disk.

Figure 33-1 Root and swap configured on two primary partitions, and free space on the disk

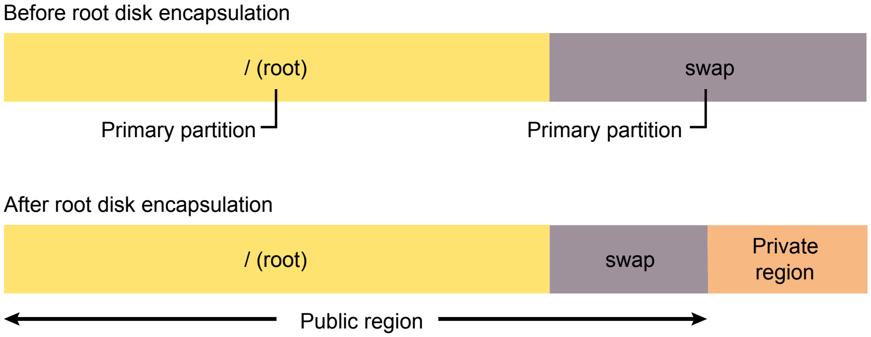


Two primary partitions are in use by `/` and `swap`. There are two unused primary partitions, and free space exists on the disk that can be assigned to a primary partition for the private region.

Example 2: supported root disk layouts for encapsulation

Figure 33-2 shows an example of a supported layout with `root` and `swap` configured on two primary partitions, and no existing free space on the disk.

Figure 33-2 Root and swap configured on two primary partitions, and no free space

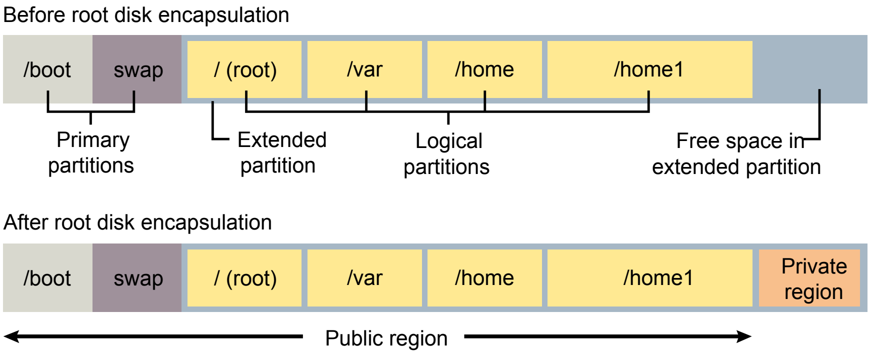


Two primary partitions are in use by `/` and `swap`. There are two unused primary partitions, and the private region can be allocated to a new primary partition by taking space from the end of the swap partition.

Example 3: supported root disk layouts for encapsulation

Figure 33-3 shows an example of a supported layout with `boot` and `swap` configured on two primary partitions, and some existing free space in the extended partition.

Figure 33-3 Boot and swap configured on two primary partitions, and free space in the extended partition

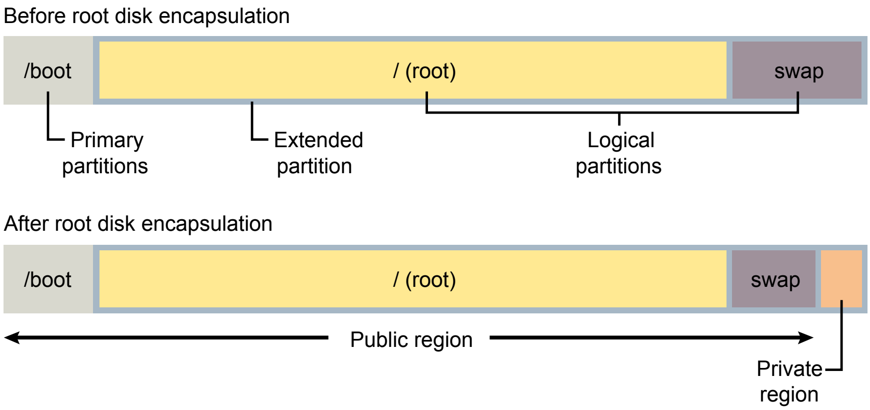


Three primary partitions are in use by `/boot`, `swap` and an extended partition that contains four file systems including `root`. There is free space at the end of the extended primary partition that can be used to create a new logical partition for the private region.

Example 4: supported root disk layouts for encapsulation

Figure 33-4 shows an example of a supported layout with `boot` configured on a primary partition, and `root` and `swap` configured in the extended partition.

Figure 33-4 Boot configured on a primary partition, and root and swap configured in the extended partition



Two primary partitions are in use by `/boot` and an extended partition that contains the `root` file system and swap area. A new logical partition can be created for the private region by taking space from the end of the swap partition.

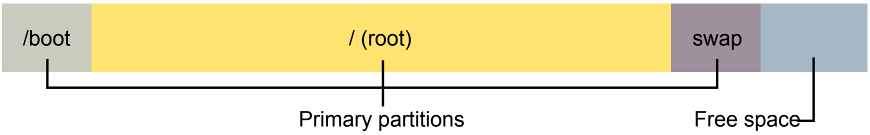
Sample unsupported root disk layouts for encapsulation

The following examples show root disk layouts that do not support encapsulation.

Example 1: unsupported root disk layouts for encapsulation

Figure 33-5 shows an example of an unsupported layout with `boot`, `swap` and `root` configured on three primary partitions, and some existing free space on the disk.

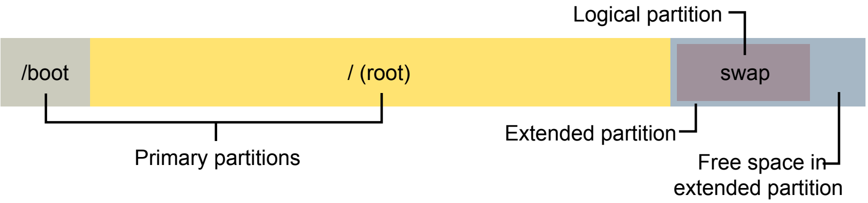
Figure 33-5 Boot, swap and root configured on three primary partitions, and free space on the disk



This layout, which is similar to the default Red Hat layout, cannot be encapsulated because only one spare primary partition is available, and neither the swap partition nor the free space lie within an extended partition.

Figure 33-6 shows a workaround by configuring the swap partition or free space as an extended partition, and moving the swap area to a logical partition (leaving enough space for a logical partition to hold the private region).

Figure 33-6 Workaround by reconfiguring swap as a logical partition

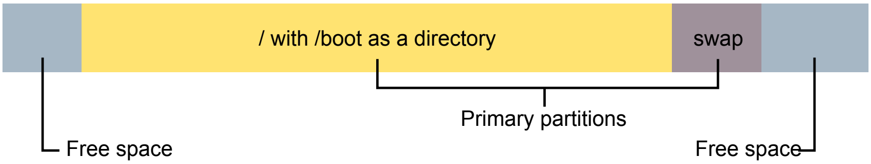


The original swap partition should be deleted. After reconfiguration, this root disk can be encapsulated.

See "Example 3: supported root disk layouts for encapsulation" on page 742.

Figure 33-7 shows another possible workaround by recreating `/boot` as a directory under `/`, deleting the `/boot` partition, and reconfiguring LILO or GRUB to use the new`/boot` location.

Figure 33-7 Workaround by reconfiguring /boot as a directory



Warning: If the start of the root file system does not lie within the first 1024 cylinders, moving `/boot` may render your system unbootable.

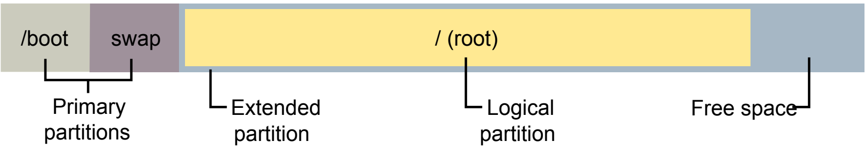
After reconfiguration, this root disk can be encapsulated.

See "Example 1: supported root disk layouts for encapsulation" on page 740.

Example 2: unsupported root disk layouts for encapsulation

Figure 33-8 shows an example of an unsupported layout with `boot` and `swap` configured on two primary partitions, and no existing free space in the extended partition.

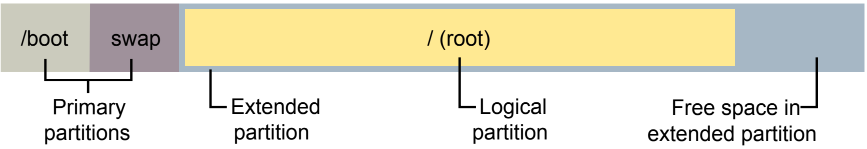
Figure 33-8 Boot and swap configured on two primary partitions, and no free space in the extended partition



This layout cannot be encapsulated because only one spare primary partition is available, and neither the swap partition nor the free space lie within the extended partition.

Figure 33-9 shows a simple workaround that uses a partition configuration tool to grow the extended partition into the free space on the disk.

Figure 33-9 Workaround by growing the extended partition



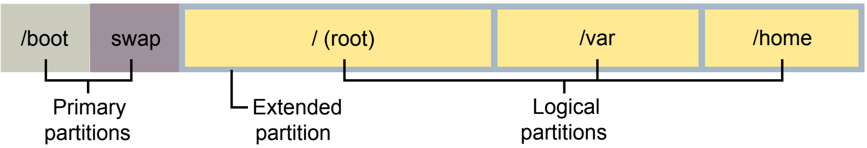
Care should be taken to preserve the boundaries of the logical partition that contains the `root` file system. After reconfiguration, this root disk can be encapsulated.

See "Example 3: supported root disk layouts for encapsulation" on page 742.

Example 3: unsupported root disk layouts for encapsulation

Figure 33-10 shows an example of an unsupported layout with `boot` and `swap` configured on two primary partitions, and no existing free space on the disk.

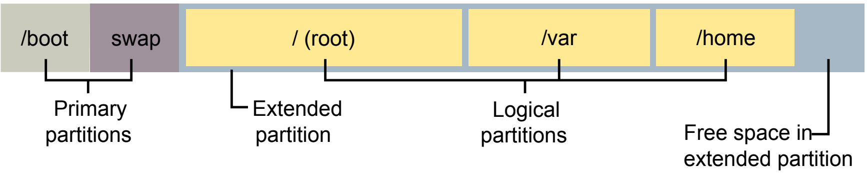
Figure 33-10 Boot and swap configured on two primary partitions, and no free space



This layout cannot be encapsulated because only one spare primary partition is available, the swap partition does not lie in the extended partition, and there is no free space in the extended partition for an additional logical partition.

Figure 33-11 shows a possible workaround by shrinking one or more of the existing file systems and the corresponding logical partitions.

Figure 33-11 Workaround by shrinking existing logical partitions



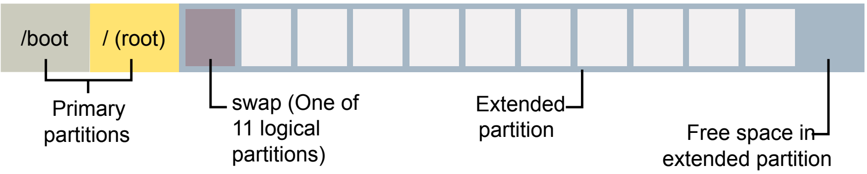
Shrinking existing logical partitions frees up space in the extended partition for the private region. After reconfiguration, this root disk can be encapsulated.

See “Example 3: supported root disk layouts for encapsulation” on page 742.

Example 4: unsupported root disk layouts for encapsulation

Figure 33-12 shows an example of an unsupported layout with `boot` and `root` configured on two primary partitions, and no more available logical partitions.

Figure 33-12 Boot and swap configured on two primary partitions, and no more available logical partitions


/boot
/ (root)
Primary partitions
swap (One of 11 logical partitions)
Extended partition
Free space in extended partition

If this layout exists on a SCSI disk, it cannot be encapsulated because only one spare primary partition is available, and even though swap is configured on a logical partition and there is free space in the extended partition, no more logical partitions can be created. The same problem arises with IDE disks when 12 logical partitions have been created.

A suggested workaround is to evacuate any data from one of the existing logical partitions, and then delete this logical partition. This makes one logical partition available for use by the private region. The root disk can then be encapsulated.

See “Example 3: supported root disk layouts for encapsulation” on page 742.

See “Example 4: supported root disk layouts for encapsulation” on page 742.

Booting root volumes

When the operating system is booted, the root file system and swap area must be available for use before the vxconfigd daemon can load the VxVM configuration or start any volumes. During system startup, the operating system must see the rootvol and swapvol volumes as regular partitions so that it can access them as ordinary disk partitions.

Due to this restriction, each of the rootvol and swapvol plexes must be created from contiguous space on a disk that is mapped to a single partition. It is not possible to stripe, concatenate or span the plex of a rootvol or swapvol volume that is used for booting. Any mirrors of these plexes that are potentially bootable also cannot be striped, concatenated or spanned.

For information on how to configure your system BIOS to boot from a disk other than the default boot disk, refer to the documentation from your hardware vendor.

Boot-time volume restrictions

Volumes on the root disk differ from other volumes in that they have very specific restrictions on their configuration:

- The root volume (rootvol) must exist in the default disk group, bootdg. Although other volumes named rootvol can be created in disk groups other than bootdg, only the volume rootvol in bootdg can be used to boot the system.
- The rootvol and swapvol volumes always have minor device numbers 0 and 1 respectively. Other volumes on the root disk do not have specific minor device numbers.
- Restricted mirrors of volumes on the root disk device have overlay partitions created for them. An overlay partition is one that exactly includes the disk space occupied by the restricted mirror. During boot, before the rootvol, varvol, usrvol and swapvol volumes are fully configured, the default volume configuration uses the overlay partition to access the data on the disk.
- Although it is possible to add a striped mirror to a rootvol device for performance reasons, you cannot stripe the primary plex or any mirrors of rootvol that may be needed for system recovery or booting purposes if the primary plex fails.
- rootvol and swapvol cannot be spanned or contain a primary plex with multiple noncontiguous subdisks. You cannot grow or shrink any volume associated with an encapsulated boot disk (rootvol, usrvol, varvol, optvol, swapvol, and so on) because these map to a physical underlying partition on the disk and must be contiguous. A workaround is to unencapsulate the boot disk, repartition the boot disk as desired (growing or shrinking partitions as needed), and then re-encapsulating.

- When mirroring parts of the boot disk, the disk being mirrored to must be large enough to hold the data on the original plex, or mirroring may not work.
- The volumes on the root disk cannot use dirty region logging (DRL).

In addition to these requirements, it is a good idea to have at least one contiguous, (cylinder-aligned if appropriate) mirror for each of the volumes for `root`, `usr`, `var`, `opt` and `swap`. This makes it easier to convert these from volumes back to regular disk partitions (during an operating system upgrade, for example).

Creating redundancy for the root disk

You can create an active backup of the root disk, in case of a single disk failure. Use the `vxrootadm` command to create a mirror of the booted root disk, and other volumes in the root disk group.

To create a back-up root disk

◆ Create a mirror with the `vxrootadm addmirror` command.

```
# vxrootadm [-v] [-Y] addmirror targetdisk
```

Creating an archived back-up root disk for disaster recovery

In addition to having an active backup of the root disk, you can keep an archived back-up copy of the bootable root disk. Use the `vxrootadm` command to create a snapshot of the booted root disk, which creates a mirror and breaks it off into a separate disk group.

To create an archived back-up root disk

1 Add a disk to the booted root disk group.

2 Create a snapshot of the booted root disk.

```
# vxrootadm [-v] mksnap targetdisk targetdg
```

3 Archive the back-up root disk group for disaster recovery.

Encapsulating and mirroring the root disk

VxVM lets you mirror the root volume and other areas needed for booting onto another disk. This makes it possible to recover from failure of your `root` disk by replacing it with one of its mirrors.

Use the `fdisk` or `sfdisk` commands to obtain a printout of the `root` disk partition table before you encapsulate the root disk. For more information, see the appropriate manual pages. You may need this information should you subsequently need to recreate the original root disk.

See the *Veritas InfoScale Troubleshooting Guide*.

See "Restrictions on using rootability with Linux" on page 738.

You can use the `vxdiskadm` command to encapsulate the root disk.

See "Encapsulating a disk" on page 730.

You can also use the `vxencap` command, as shown in this example where the root disk is `sda`:

```
# vxencap -c -g diskgroup rootdisk=sda
```

where *diskgroup* must be the name of the current boot disk group. If no boot disk group currently exists, one is created with the specified name. The name `bootdg` is reserved as an alias for the name of the boot disk group, and cannot be used. You must reboot the system for the changes to take effect.

Both the `vxdiskadm` and `vxencap` procedures for encapsulating the root disk also update the `/etc/fstab` file and the boot loader configuration file (`/boot/grub/menu.lst` or `/etc/grub.conf` (as appropriate for the platform) for GRUB or `/etc/lilo.conf` for LILO):

- Entries are changed in `/etc/fstab` for the `rootvol`, `swapvol` and other volumes on the encapsulated root disk.
- A special entry, `vxvm_root`, is added to the boot loader configuration file to allow the system to boot from an encapsulated root disk.

The contents of the original `/etc/fstab` and boot loader configuration files are saved in the files `/etc/fstab.b4vxvm`, `/boot/grub/menu.lst.b4vxvm` or `/etc/grub.conf.b4vxvm` for GRUB, and `/etc/lilo.conf.b4vxvm` for LILO.

Warning: When modifying the `/etc/fstab` and the boot loader configuration files, take care not to corrupt the entries that have been added by VxVM. This can prevent your system from booting correctly.

To mirror the root disk onto another disk after encapsulation

1. Choose a disk to use for the mirror that is at least as large as the existing `root` disk, whose geometry is seen by Linux to be the same as the existing root disk, and which is not already in use by VxVM or any other subsystem (such as a mounted partition or swap area). The disk should be visible to the Basic Input Output System (BIOS) and to the bootloader of the operating system.
2. Select `Mirror Volumes on a Disk` from the `vxdiskadm` main menu to create a mirror of the root disk. (These automatically invoke the `vxrootmir` command if the mirroring operation is performed on the root disk.)

   The disk that is used for the root mirror must not be under Volume Manager control already.

3 Alternatively, to mirror all file systems on the root disk, run the following command:

```
# vxrootmir mirror_da_name
   mirror_dm_name
```

mirror_da_name is the disk access name of the disk that is to mirror the root disk, and *mirror_dm_name* is the disk media name that you want to assign to the mirror disk. The alternate `root` disk is configured to allow the system to be booted from it in the event that the primary root disk fails. For example, to mirror the root disk, `sda`, onto disk `sdb`, and give this the disk name `rootmir`, you would use the following command:

```
# vxrootmir sdb rootmir
```

The operations to set up the root disk mirror take some time to complete.

The following is example output from the `vxprint` command after the root disk has been encapsulated and its mirror has been created (the `TUTIL0` and `PUTIL0` fields and the subdisk records are omitted for clarity):

```
Disk group: rootdg

TY NAME          ASSOC   KSTATE   LENGTH    PLOFFS   STATE ...
dg rootdg        rootdg  -        -         -        -

dm rootdisk      sda     -        16450497  -        -
dm rootmir       sdb     -        16450497  -        -

v  rootvol       root    ENABLED  12337857  -        ACTIVE
pl mirrootvol-01 rootvol ENABLED  12337857  -        ACTIVE
pl rootvol-01    rootvol ENABLED  12337857  -        ACTIVE

v  swapvol       swap    ENABLED  4112640   -        ACTIVE
pl mirswapvol-01 swapvol ENABLED  4112640   -        ACTIVE
pl swapvol-01    swapvol ENABLED  4112640   -        ACTIVE
```

Allocation of METADATA Subdisks During Root Disk Encapsulation

`METADATA` subdisks are created during root disk encapsulation to protect partitioning information. These subdisks are deleted automatically when a root disk is unencapsulated.

The following example `fdisk` output shows the original partition table for a system's root disk:

```
# fdisk -ul /dev/hda
Disk /dev/hda: 255 heads, 63 sectors, 2431 cylinders
Units = sectors of 1 * 512 bytes

   Device Boot    Start        End   Blocks    Id  System
/dev/hda1             63   2104514   1052226   83  Linux
/dev/hda2        2104515   6297479   2096482+  83  Linux
/dev/hda3        6329610  39054014  16362202+   5  Extended
/dev/hda5        6329673  10522574   2096451   83  Linux
/dev/hda6       10522638  14715539   2096451   83  Linux
/dev/hda7       14715603  18908504   2096451   83  Linux
/dev/hda8       18908568  23101469   2096451   83  Linux
/dev/hda9       23101533  25205984   1052226   82  Linux swap
```

Notice that there is a gap between start of the extended partition (`hda3`) and the start of the first logical partition (`hda5`). For the logical partitions (`hda5` through `hda9`), there are also gaps between the end of one logical partition and the start of the next logical partition. These gaps contain metadata for partition information. Because these metadata regions lie inside the public region, VxVM allocates subdisks over them to prevent accidental allocation of this space to volumes.

After the root disk has been encapsulated, the output from the `vxprint` command appears similar to the following:

```
Disk group: rootdg

TY NAME            ASSOC       KSTATE   LENGTH   PLOFFS  STATE   TUTIL0  PUTIL0
dg rootdg          rootdg      -        -        -       -       -       -
dm disk01          sdh         -        17765181 -       -       -       -
dm rootdisk        hda         -        39053952 -       -       -       -

sd meta-rootdisk05 -           ENABLED  63       -       -       -       METADATA
sd meta-rootdisk06 -           ENABLED  63       -       -       -       METADATA
sd meta-rootdisk07 -           ENABLED  63       -       -       -       METADATA
sd meta-rootdisk08 -           ENABLED  63       -       -       -       METADATA
sd meta-rootdisk09 -           ENABLED  63       -       -       -       METADATA
sd meta-rootdisk10 -           ENABLED  63       -       -       -       METADATA
sd rootdiskPriv     -          ENABLED  2049     -       -       -       PRIVATE

v bootvol           fsgen      ENABLED  2104452  -       ACTIVE  -       -
pl bootvol-01       bootvol    ENABLED  2104452  -       ACTIVE  -       -
```

```
sd rootdisk-07      bootvol-01 ENABLED  2104452  0       -       -       -

v  homevol          fsgen      ENABLED  4192902  -       ACTIVE  -       -
pl homevol-01       homevol    ENABLED  4192902  -       ACTIVE  -       -
sd rootdisk-05      homevol-01 ENABLED  4192902  0       -       -       -

v  optvol           fsgen      ENABLED  4192902  -       ACTIVE  -       -
pl optvol-01        optvol     ENABLED  4192902  -       ACTIVE  -       -
sd rootdisk-04      optvol-01  ENABLED  4192902  0       -       -       -

v  rootvol          root       ENABLED  4192902  -       ACTIVE  -       -
pl rootvol-01       rootvol    ENABLED  4192902  -       ACTIVE  -       -
sd rootdisk-02      rootvol-01 ENABLED  4192902  0       -       -       -

v  swapvol          swap       ENABLED  2104452  -       ACTIVE  -       -
pl swapvol-01       swapvol    ENABLED  2104452  -       ACTIVE  -       -
sd rootdisk-01      swapvol-01 ENABLED  2104452  0       -       -       -

v  usrvol           fsgen      ENABLED  4192965  -       ACTIVE  -       -
pl usrvol-01        usrvol     ENABLED  4192965  -       ACTIVE  -       -
sd rootdisk-06      usrvol-01  ENABLED  4192965  0       -       -       -

v  varvol           fsgen      ENABLED  4192902  -       ACTIVE  -       -
pl varvol-01        varvol     ENABLED  4192902  -       ACTIVE  -       -
sd rootdisk-03      varvol-01  ENABLED  4192902  0       -       -       -
```

The new partition table for the root disk appears similar to the following:

```
# fdisk -ul /dev/hda
Disk /dev/hda: 255 heads, 63 sectors, 2431 cylinders
Units = sectors of 1 * 512 bytes

    Device Boot    Start     End    Blocks  Id  System
/dev/hda1             63 2104514   1052226  83  Linux
/dev/hda2        2104515 6297479   2096482+ 83  Linux
/dev/hda3        6329610 39054014 16362202+  5  Extended
/dev/hda4             63 39054014 19526976  7e  Unknown
/dev/hda5        6329673 10522574 2096451   83  Linux
/dev/hda6       10522638 14715539 2096451   83  Linux
/dev/hda7       14715603 18908504 2096451   83  Linux
/dev/hda8       18908568 23101469 2096451   83  Linux
/dev/hda9       23101533 25205984 1052226   82  Linux swap
/dev/hda10      39051966 39054014 1024+     7f  Unknown
```

In this example, primary partition `hda4` and logical partition `hda10` have been created to represent the VxVM public and private regions respectively.

Upgrading the kernel on a root encapsulated system

OS vendors often release maintenance patches to their products to address security issues and other minor product defects. They may require customers to regularly apply these patches to conform with maintenance contracts or to be eligible for vendor support. Prior to this release, it was not possible to install a kernel patch or upgrade on a root encapsulated system: it was necessary to unencapsulate the system, apply the upgrade, then reencapsulate the root disk. It is now possible to upgrade the OS kernel on a root encapsulated system.

Note: The procedures in this section only apply to minor kernel upgrades or patches. These procedures do not apply to a full upgrade of the Linux operating system.

To upgrade the OS kernel on a root encapsulated system

1. Apply the minor upgrade or patch to the system.
2. After applying the upgrade, run the commands:

   ```
   # . /etc/vx/modinst-vxvm
   ```

   ```
   # upgrade_encapped_root
   ```

   The above commands determine if the kernel upgrade can be applied to the encapsulated system. If the upgrade is successful, the command displays the following message:

   ```
   # upgrade_encapped_root
   The VxVM root encapsulation upgrade has succeeded.
   Please reboot the machine to load the new kernel.
   ```

   After the next reboot, the system restarts with the patched kernel and a VxVM encapsulated root volume.

Some patches may be completely incompatible with the installed version of VxVM. In this case the script fails, with the following message:

```
# upgrade_encapped_root
FATAL ERROR: Unencapsulate the root disk manually.
VxVM cannot re-encapsulate the upgraded system.
```

The upgrade script saves a system configuration file that can be used to boot the system with the previous configuration. If the upgrade fails, follow the steps to restore the previous configuration.

Note: The exact steps may vary depending on the operating system.

To restore the previous configuration

1. Interrupt the GRuB bootloader at bootstrap time by pressing the space bar.

   The system displays a series of potential boot configurations, named after the various installed kernel versions and VxVM root encapsulation versions.

   For example:

   ```
   Red Hat Enterprise Linux Server (2.6.18-53.el5)
   Red Hat Enterprise Linux Server (2.6.18-8.el5)
   vxvm_root_backup
   vxvm_root
   ```

2. Select the `vxvm_root_backup` option to boot the previous kernel version with the VxVM encapsulated root disk.

To upgrade the OS kernel on a root encapsulated system using manual steps

1. If the upgrade script fails, you can manually unencapsulate the root disk to allow it to boot.

   See “Unencapsulating the root disk” on page 757.

2. Upgrade the kernel and reboot the system.

3. If the reboot succeeds, you can re-encapsulate and remirror the root disk.

   See “Encapsulating and mirroring the root disk” on page 748.

   However, after the next reboot, VxVM may not be able to run correctly, making all VxVM volumes unavailable. To restore the VxVM volumes, you must remove the kernel upgrade, as follows:

   ```
   # rpm -e upgrade_kernel_package_name
   ```

   For example:

   ```
   # rpm -e kernel-2.6.18-53.el5
   ```

Administering an encapsulated boot disk

The vxrootadm command lets you make a bootable snapshot of an encapsulated boot disk.

The vxrootadm command has the following format:

```
# vxrootadm [-v] [-g dg] [-s srcdisk] ... keyword arg ...
```

The mksnap keyword must have the following format:

```
# vxrootadm -s srcdisk mksnap destdisk newdg
```

vxrootadm includes the following options:

vxrootadm [-v] [-D]	These are verbose and debug message options and are optional.
vxrootadm [-g dg]	The disk group argument is optional.
vxrootadm -s *srcdisk*	Specifies the source disk.

See the vxrootadm(1m) manual page.

Creating a snapshot of an encapsulated boot disk

To create a snapshot of an encapsulated boot disk, use the vxrootadm command. The target disk for the snapshot must be as large (or bigger) than the source disk (boot disk). You must use a new disk group name to associate the target disk.

To create a snapshot of an encapsulated boot disk

- Enter the following command:

  ```
  # vxrootadm -s srcdisk [-g dg] mksnap destdisk newdg
  ```

  For example:

  ```
  # vxrootadm -s disk_0 -g rootdg mksnap disk_1 snapdg
  ```

  In this example, disk_0 is the encapsulated boot disk, and rootdg is the associate boot disk group. disk_1 is the target disk, and snapdg is the new disk group name

Unencapsulating the root disk

You can use the vxunroot utility to remove rootability support from a system. This makes root, swap, home and other file systems on the root disk directly accessible through disk partitions, instead of through volume devices.

The vxunroot utility also makes the necessary configuration changes to allow the system to boot without any dependency on VxVM.

Only the volumes that were present on the root disk when it was encapsulated can be unencapsulated using vxunroot. Before running vxunroot, evacuate all other volumes that were created on the root disk after it was encapsulated.

Do not remove the plexes on the root disk that correspond to the original disk partitions.

Warning: This procedure requires a reboot of the system.

To remove rootability from a system

1 Use the vxplex command to remove all the plexes of the volumes rootvol, swapvol, usr, var, opt and home on the disks other than the root disk.

For example, the following command removes the plexes mirrootvol-01 and mirswapvol-01 that are configured on the disk rootmir:

```
# vxplex -g bootdg -o rm dis mirrootvol-01 mirswapvol-01
```

2 Run the vxunroot utility:

```
# vxunroot
```

vxunroot does not perform any conversion to disk partitions if any plexes remain on other disks.

If the device naming scheme has changed since the root disk was encapsulated, the vxunroot command fails with the following error:

```
VxVM vxunroot ERROR V-5-2-4101 The root disk name does not match
the name of the original disk that was encapsulated.
```

If this message displays, use the vxddladm assign names command to regenerate the persistent device name for the encapsulated root disk, then retry the vxunroot command.

See “Regenerating persistent device names” on page 289.

Chapter 34

Quotas

This chapter includes the following topics:

About Veritas File System quota limits

Veritas File System (VxFS) supports user and group quotas. The quota system limits the use of two principal resources of a file system: files and data blocks. For each of these resources, you can assign quotas to individual users and groups to limit their usage.

You can set the following kinds of limits for each of the two resources:

hard limit — An absolute limit that cannot be exceeded under any circumstances.

soft limit — Must be lower than the hard limit, and can be exceeded, but only for a limited time. The time limit can be configured on a per-file system basis only. The VxFS default limit is seven days.

Soft limits are typically used when a user must run an application that could generate large temporary files. In this case, you can allow the user to exceed the quota limit for a limited time. No allocations are allowed after the expiration of the time limit.

Use the `vxedquota` command to set limits.

See “Using Veritas File System quotas” on page 761.

Although file and data block limits can be set individually for each user and group, the time limits apply to the file system as a whole. The quota limit information is associated with user and group IDs and is stored in a user or group quota file.

See "About quota files on Veritas File System" on page 759.

The quota soft limit can be exceeded when VxFS preallocates space to a file.

See "About extent attributes" on page 185.

About quota files on Veritas File System

A quotas file (named `quotas`) must exist in the root directory of a file system for any of the quota commands to work. For group quotas to work, there must be a `quotas.grp` or `quotas.grp.64` file. The files in the file system's mount point are referred to as the external quotas file. VxFS also maintains an internal quotas file for its own use.

The quota administration commands read and write to the external quotas file to obtain or change usage limits. VxFS uses the internal file to maintain counts of data blocks and inodes used by each user. When quotas are turned on, the quota limits are copied from the external quotas file into the internal quotas file. While quotas are on, all the changes in the usage information and changes to quotas are registered in the internal quotas file. When quotas are turned off, the contents of the internal quotas file are copied into the external quotas file so that all data between the two files is synchronized.

VxFS supports group quotas in addition to user quotas. Just as user quotas limit file system resource (disk blocks and the number of inodes) usage on individual users, group quotas specify and limit resource usage on a group basis. As with user quotas, group quotas provide a soft and hard limit for file system resources. If both user and group quotas are enabled, resource utilization is based on the most restrictive of the two limits for a given user.

To distinguish between group and user quotas, VxFS quota commands use a `-g` and `-u` option. The default is user quotas if neither option is specified. One exception to this rule is when you specify the `-o quota` option as a `mount` command option. In this case, both user and group quotas are enabled. Support for group quotas also requires a separate group quotas file. The VxFS group quota file is named `quotas.grp` or `quotas.grp.64`. The VxFS user quotas file is named `quotas` or `quotas.64`. This name was used to distinguish it from the `quotas.user` file used by other file systems under the operating system.

About Veritas File System quota commands

Note: The `quotacheck` command is an exception—VxFS does not support an equivalent command.

See "About quota checking with Veritas File System" on page 761.

Quota support for various file systems is implemented using the generic code provided by the Linux kernel. However, VxFS does not use this generic interface. VxFS instead supports a similar set of commands that work only for VxFS file systems.

VxFS supports the following quota-related commands:

`vxedquota`	Edits quota limits for users and groups. The limit changes made by `vxedquota` are reflected both in the internal quotas file and the external quotas file.
`vxrepquota`	Provides a summary of quotas and disk usage.
`vxquot`	Provides file ownership and usage summaries.
`vxquota`	Views quota limits and usage.
`vxquotaon`	Turns quotas on for a mounted VxFS file system.
`vxquotaoff`	Turns quotas off for a mounted VxFS file system.

The `vxquota`, `vxrepquota`, `vxquot`, and `vxedquota` commands support the `-H` option for human friendly input and output. When the `-H` option is used, the storage size is displayed in the following human−friendly units: bytes (B), kilobytes (KB), megabytes (MB), gigabytes (GB), terabyte (TB), petabytes (PB), and exabytes (EB). The quota soft and hard limits, quota usage, and the total storage consumed by a specific user or group or all users or groups can be obtained in human-friendly units using the `-H` option.

In addition to these commands, the VxFS `mount` command supports a special mount option (`-o quota|usrquota|grpquota`), which can be used to turn on quotas at mount time. You can also selectively enable or disable user or group quotas on a VxFS file system during remount or on a mounted file system.

For additional information on the quota commands, see the `vxedquota`(1M), `vxrepquota`(1M), `vxquot`(1M), `vxquota`(1M), `vxquotaon`(1M), and `vxquotaoff`(1M) manual pages.

Note: When VxFS file systems are exported via NFS, the VxFS quota commands on the NFS client cannot query or edit quotas. You can use the VxFS quota commands on the server to query or edit quotas.

About quota checking with Veritas File System

The standard practice with most quota implementations is to mount all file systems and then run a quota check on each one. The quota check reads all the inodes on disk and calculates the usage for each user and group. This can be time consuming, and because the file system is mounted, the usage can change while quotacheck is running.

VxFS does not support a `quotacheck` command. With VxFS, quota checking is performed automatically (if necessary) at the time quotas are turned on. A quota check is necessary if the file system has changed with respect to the usage information as recorded in the internal quotas file. This happens only if the file system was written with quotas turned off, or if there was structural damage to the file system that required a full file system check.

See the `fsck_vxfs`(1M) manual page.

A quota check generally reads information for each inode on disk and rebuilds the internal quotas file. It is possible that while quotas were not on, quota limits were changed by the system administrator. These changes are stored in the external quotas file. As part of enabling quotas processing, quota limits are read from the external quotas file into the internal quotas file.

Using Veritas File System quotas

The Veritas File System (VxFS) quota commands are used to perform the following quota functions:

- Turning on Veritas File System quotas
- Turning on Veritas File System quotas at mount time
- Editing Veritas File System quotas
- Modifying Veritas File System quota time limits
- Viewing Veritas File System disk quotas and usage
- Displaying blocks owned by users or groups
- Turning off Veritas File System quotas
- Support for 64-bit Quotas

Turning on Veritas File System quotas

To use the quota functionality on a file system, quotas must be turned on. You can turn quotas on at mount time or after a file system is mounted.

Note: Before turning on quotas, the root directory of the file system must contain a file for user quotas named `quotas`, and a file for group quotas named `quotas.grp` owned by root.

64-bit quota is supported on DLV>=10. The quota file names are `quotas.64` and `quotas.grp.64`.

To turn on quotas

1 To turn on user and group quotas for a VxFS file system, enter:

```
# vxquotaon /mount_point
```

2 To turn on only user quotas for a VxFS file system, enter:

```
# vxquotaon -u /mount_point
```

3 To turn on only group quotas for a VxFS file system, enter:

```
# vxquotaon -g /mount_point
```

Turning on Veritas File System quotas at mount time

Quotas can be turned on with the `mount` command when you mount a file system.

To turn on quotas at mount time

1 To turn on user or group quotas for a file system at mount time, enter:

```
# mount -t vxfs -o quota special /mount_point
```

where *special* is aVxFS block special device.

2 To turn on only user quotas, enter:

```
# mount -t vxfs -o usrquota special /mount_point
```

3 To turn on only group quotas, enter:

```
# mount -t vxfs -o grpquota special /mount_point
```

Note: "*special*" signifies a VxFS block special device.

Editing Veritas File System quotas

You can set up user and group quotas using the vxedquota command. You must have superuser privileges to edit quotas.

vxedquota creates a temporary file for the given user; this file contains on-disk quotas for each mounted file system that has a quotas file. It is not necessary that quotas be turned on for vxedquota to work. However, the quota limits are applicable only after quotas are turned on for a given file system.

To edit quotas

1 Specify the -u option to edit the quotas of one or more users specified by *username*:

```
# vxedquota [-u] username
```

Editing the quotas of one or more users is the default behavior if the -u option is not specified.

2 Specify the -g option to edit the quotas of one or more groups specified by *groupname*:

```
# vxedquota -g groupname
```

Modifying Veritas File System quota time limits

The soft and hard limits can be modified or assigned values with the vxedquota command. For any user or group, usage can never exceed the hard limit after quotas are turned on.

Modified time limits apply to the entire file system and cannot be set selectively for each user or group.

To modify time limits

1 Specify the -t option to modify time limits for any user:

```
# vxedquota [-u] -t
```

2 Specify the -g and -t options to modify time limits for any group:

```
# vxedquota -g -t
```

Viewing Veritas File System disk quotas and usage

Use the vxquota command to view a user's or group's disk quotas and usage on VxFS file systems.

To display disk quotas and usage

1 To display a user's quotas and disk usage on all mounted VxFS file systems where the quotas file exists, enter:

```
# vxquota -v [-u] username
```

2 To display a group's quotas and disk usage on all mounted VxFS file systems where the quotas.grp file exists, enter:

```
# vxquota -v -g groupname
```

Displaying blocks owned by users or groups

Use the vxquot command to display the number of blocks owned by each user or group in a file system.

To display the number of blocks owned by users or groups

1 To display the number of files and the space owned by each user, enter:

```
# vxquot [-u] -f filesystem
```

2 To display the number of files and the space owned by each group, enter:

```
# vxquot -g -f filesystem
```

Turning off Veritas File System quotas

Use the vxquotaoff command to turn off quotas.

To turn off quotas

1 To turn off quotas for a mounted file system, enter:

```
# vxquotaoff /mount_point
```

2 To turn off only user quotas for a VxFS file system, enter:

```
# vxquotaoff -u /mount_point
```

3 To turn off only group quotas for a VxFS file system, enter:

```
# vxquotaoff -g /mount_point
```

Support for 64-bit Quotas

64-bit quotas are supported on DLV>=10. To use this feature the existing file systems need to be upgraded to DLV>=10 using the `vxupgrade` command.

See `vxupgrade`(1M) man page.

As for 32-bit quotas, this continues to be supported on disk layout Version 9 or earlier. The same quota commands can be used for both 32-bit and 64-bit quotas.

As for 64-bit quotas, there are two new quotas files. For group quotas the file name is `quotas.grp.64` and for user quotas the file name is `quotas.64`. These files will be created on each file system after the disk layout version upgrade is completed.

Chapter 35

File Change Log

This chapter includes the following topics:

About Veritas File System File Change Log

The VxFS File Change Log (FCL) tracks changes to files and directories in a file system.

Applications that typically use the FCL are usually required to:

- scan an entire file system or a subset
- discover changes since the last scan

These applications may include: backup utilities, webcrawlers, search engines, and replication programs.

Note: The FCL tracks when the data has changed and records the change type, but does not track the actual data changes. It is the responsibility of the application to examine the files to determine the changed data.

FCL functionality is a separately licensable feature.

About the Veritas File System File Change Log file

File Change Log records file system changes such as creates, links, unlinks, renaming, data appended, data overwritten, data truncated, extended attribute modifications, holes punched, and miscellaneous file property updates.

FCL stores changes in a sparse file in the file system namespace. The FCL file is located in `mount_point/lost+found/changelog`. The FCL file behaves like a regular file, but some operations are prohibited. The standard system calls `open`(2), `lseek`(2), `read`(2) and `close`(2) can access the data in the FCL, while the `write`(2), `mmap`(2) and `rename`(2) calls are not allowed.

Warning: Although some standard system calls are currently supported, the FCL file might be pulled out of the namespace in future VxFS release and these system calls may no longer work. It is recommended that all new applications be developed using the programmatic interface.

The FCL log file contains both the information about the FCL, which is stored in the FCL superblock, and the changes to files and directories in the file system, which is stored as FCL records.

See "Veritas File System File Change Log programmatic interface" on page 770.

In the 4.1 release, the structure of the File Change Log file was exposed through the `/opt/VRTS/include/sys/fs/fcl.h` header file. In this release, the internal structure of the FCL file is opaque. The recommended mechanism to access the FCL is through the API described by the `/opt/VRTSfssdk/7.0.0.000/include/vxfsutil.h` header file.

The `/opt/VRTS/include/sys/fs/fcl.h` header file is included in this release to ensure that applications accessing the FCL with the 4.1 header file do not break. New applications should use the new FCL API described in `/opt/VRTSfssdk/7.0.0.000/include/vxfsutil.h`. Existing applications should also be modified to use the new FCL API.

To provide backward compatibility for the existing applications, this release supports multiple FCL versions. Users have the flexibility of specifying the FCL version for new FCLs. The default FCL version is 4.

See the `fcladm`(1M) man page.

Veritas File System File Change Log administrative interface

The File Change Log (FCL) can be set up and tuned through the `fcladm` and `vxtunefs` VxFS administrative commands.

See the `fcladm`(1M) and `vxtunefs`(1M) manual pages.

The FCL keywords for `fcladm` are as follows:

`clear` Disables the recording of the audit, open, close, and statistical events after it has been set.

`dump` Creates a regular file image of the FCL file that can be downloaded to an off-host processing system. This file has a different format than the FCL file.

`on` Activates the FCL on a mounted file system. VxFS 5.0 and later releases support either FCL Versions 3 or 4. If no version is specified, the default is Version 4. Use `fcladm on` to specify the version.

`print` Prints the contents of the FCL file starting from the specified offset.

`restore` Restores the FCL file from the regular file image of the FCL file created by the `dump` keyword.

`rm` Removes the FCL file. You must first deactivate the FCL with the `off` keyword, before you can remove the FCL file.

`set` Enables the recording of events specified by the 'eventlist' option.

See the `fcladm`(1M) manual page.

`state` Writes the current state of the FCL to the standard output.

`sync` Brings the FCL to a stable state by flushing the associated data of an FCL recording interval.

The FCL tunable parameters for `vxtunefs` are as follows:

`fcl_keeptime` Specifies the duration in seconds that FCL records stay in the FCL file before they can be purged. The first records to be purged are the oldest ones, which are located at the beginning of the file. Additionally, records at the beginning of the file can be purged if allocation to the FCL file exceeds `fcl_maxalloc` bytes. The default value of `fcl_keeptime` is 0. If the `fcl_maxalloc` parameter is set, records are purged from the FCL file if the amount of space allocated to the FCL file exceeds `fcl_maxalloc`. This is true even if the elapsed time the records have been in the log is less than the value of `fcl_keeptime`.

`fcl_maxalloc` Specifies the maximum number of spaces in bytes to be allocated to the FCL file. When the space allocated exceeds `fcl_maxalloc`, a hole is punched at the beginning of the file. As a result, records are purged and the first valid offset (`fc_foff`) is updated. In addition, `fcl_maxalloc` may be violated if the oldest record has not reached `fcl_keeptime`.

The minimum value of `fcl_maxalloc` is 4 MB. The default value is `fs_size`/33.

`fcl_winterval` Specifies the time in seconds that must elapse before the FCL records an overwrite, extending write, or a truncate. This helps to reduce the number of repetitive records in the FCL. The `fcl_winterval` timeout is per inode. If an inode happens to go out of cache and returns, its write interval is reset. As a result, there could be more than one write record for that file in the same write interval. The default value is 3600 seconds.

`fcl_ointerval` The time interval in seconds within which subsequent opens of a file do not produce an additional FCL record. This helps to reduce the number of repetitive records logged in the FCL file. If the tracking of access information is also enabled, a subsequent file open even within the `fcl_ointerval` may produce a record, if it is opened by a different user. Similarly, if the inode is bumped out of cache, this may also produce more than one record within the same open interval.

The default value is 600 sec.

Either or both `fcl_maxalloc` and `fcl_keeptime` must be set to activate the FCL feature. The following are examples of using the `fcladm` command.

To activate FCL for a mounted file system, type the following:

```
# fcladm on mount_point
```

To deactivate the FCL for a mounted file system, type the following:

```
# fcladm off mount_point
```

To remove the FCL file for a mounted file system, on which FCL must be turned off, type the following:

```
# fcladm rm mount_point
```

To obtain the current FCL state for a mounted file system, type the following:

```
# fcladm state mount_point
```

To enable tracking of the file opens along with access information with each event in the FCL, type the following:

```
# fcladm set fileopen,accessinfo mount_point
```

To stop tracking file I/O statistics in the FCL, type the following:

```
# fcladm clear filestats mount_point
```

Print the on-disk FCL super-block in text format to obtain information about the FCL file by using offset 0. Because the FCL on-disk super-block occupies the first block of the FCL file, the first and last valid offsets into the FCL file can be determined by reading the FCL super-block and checking the `fc_foff` field. Enter:

```
# fcladm print 0 mount_point
```

To print the contents of the FCL in text format, of which the offset used must be 32-byte aligned, enter:

```
# fcladm print offset mount_point
```

Veritas File System File Change Log programmatic interface

VxFS provides an enhanced API to simplify reading and parsing the FCL file in two ways:

Simplified reading	The API simplifies user tasks by reducing additional code needed to parse FCL file entries. In 4.1, to obtain event information such as a remove or link, the user was required to write additional code to get the name of the removed or linked file. In this release, the API allows the user to directly read an assembled record. The API also allows the user to specify a filter to indicate a subset of the event records of interest.

Backward compatibility	Providing API access for the FCL feature allows backward compatibility for applications. The API allows applications to parse the FCL file independent of the FCL layout changes. Even if the hidden disk layout of the FCL changes, the API automatically translates the returned data to match the expected output record. As a result, the user does not need to modify or recompile the application due to changes in the on-disk FCL layout.

The following sample code fragment reads the FCL superblock, checks that the state of the FCL is `VX_FCLS_ON`, issues a call to `vxfs_fcl_sync` to obtain a finishing offset to read to, determines the first valid offset in the FCL file, then reads the entries in 8K chunks from this offset. The section process fcl entries is what an application developer must supply to process the entries in the FCL file.

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/fcntl.h>
#include <errno.h>
#include <fcl.h>
#include <vxfsutil.h>
#define FCL_READSZ 8192
char* fclname = "/mnt/lost+found/changelog";
int read_fcl(fclname) char* fclname;
{
    struct fcl_sb fclsb;
    uint64_t off, lastoff;
    size_t size;
    char buf[FCL_READSZ], *bufp = buf;
    int fd;
    int err = 0;
    if ((fd = open(fclname, O_RDONLY)) < 0) {
        return ENOENT;
    }
    if ((off = lseek(fd, 0, SEEK_SET)) != 0) {
        close(fd);
        return EIO;
    }
    size = read(fd, &fclsb, sizeof (struct fcl_sb));
    if (size < 0) {
        close(fd);
        return EIO;
    }
    if (fclsb.fc_state == VX_FCLS_OFF) {
        close(fd);
        return 0;
    }
    if (err = vxfs_fcl_sync(fclname, &lastoff)) {
        close(fd);
        return err;
    }
    if ((off = lseek(fd, off_t, uint64_t)) != uint64_t) {
        close(fd);
        return EIO;
    }
    while (off < lastoff) {
        if ((size = read(fd, bufp, FCL_READSZ)) <= 0) {
            close(fd);
            return errno;
        }
        /* process fcl entries */
        off += size;
    }
    close(fd);
    return 0;
}
```

Summary of Veritas File System File Change Log API functions

The following is a brief summary of File Change Log API functions:

`vxfs_fcl_close()`	Closes the FCL file and cleans up resources associated with the handle.
`vxfs_fcl_cookie()`	Returns an opaque structure that embeds the current FCL activation time and the current offset. This cookie can be saved and later passed to `vxfs_fcl_seek()` function to continue reading from where the application last stopped.
`vxfs_fcl_getinfo()`	Returns information such as the state and version of the FCL file.
`vxfs_fcl_open()`	Opens the FCL file and returns a handle that can be used for further operations.

`vxfs_fcl_read`() Reads FCL records of interest into a buffer specified by the user.

`vxfs_fcl_seek`() Extracts data from the specified cookie and then seeks to the specified offset.

`vxfs_fcl_seektime`() Seeks to the first record in the FCL after the specified time.

Support for protection against ransomware

This chapter includes the following topics:

About support for protection against ransomware

InfoScale provides several VxFS features to ensure compliance with the security and governance requirements of critical data. The following sections describe these features and their usage.

Write Once, Read Many (WORM) storage

InfoScale provides the facility to create a WORM storage system that you can use to retain critical data for a certain period and to ensure that the data cannot be modified. When a file is committed to WORM storage, the data in the file can be read but cannot be overwritten or erased within the specified retention period. Thus, committing a file to WORM storage prevents any accidental or intentional erasure of its data. The retention period for a WORM-enabled file specifies the duration within which the file cannot be deleted after it is committed to the WORM storage.

The storage system allows the file to be deleted only after the retention period has expired.

You can enable WORM and set a different retention period for every individual file. However, per-file WORM enablement is supported only on WORM-enabled file systems, so ensure that the file system on the server is WORM-enabled. You can identify whether a file system is WORM-enabled by using one of the following commands:

- To confirm WORM-enablement of a file system using a device path

  ```
  # mkfs -t vxfs -m absolutePathOfVolume
  ```

- To confirm WORM-enablement of a file system using a mount point

  ```
  # /opt/VRTS/bin/fsadm nameOfMountPoint
  ```

The output of these commands displays the worm keyword to specify that the file system is WORM-enabled.

When you WORM-enable a file, you specify the retention period. Accordingly, the access time of the file is internally set to a date in the future. The maximum value allowed for the retention period is up to 2038-01-19.

Once enabled, you cannot disable WORM for a specific file system. The file system remains WORM-enabled forever.

You can WORM-enable a file system in one of the following ways:

- Use the `-o worm` option while creating the file system.
- Use the `fsadm` command to enable WORM for an existing file system that is mounted.

You can WORM-enable a file in one of the following ways:

- Write a utility to WORM-enable a file by using VxFS APIs.
- Set the access time (`atime`) of a file to the required retention time and then set the read-only attribute of the file.

To WORM-enable a file by manually setting its access time

1. Change the access time of the file so that it has the same value as the retention period, by running the following command:

   ```
   # touch -at YYYYMMDDhhmm.ss nameOfFile
   ```

   For example, if a file named `foo` must be retained till 14th July 2035 10:37:42 pm, run:

   ```
   # touch -at 203507141037.42 foo
   ```

2. Mark the file as read-only by changing its permissions.

   For example, to make the `foo` file read-only, run:

   ```
   # chmod -w foo
   ```

When these steps are completed successfully, `foo` becomes WORM-enabled, with 14th July 2035 10:37:42 pm as its retention period.

Secure clock

System clocks can be easily manipulated. To ensure that the WORM retention period is calculated accurately regardless of the system clock value, InfoScale uses an alternate mechanism. A secure clock or a compliance clock (CompClock) is a software clock that gets generated automatically for a WORM-enabled file system and is controlled by VxFS. The secure clock is stored in the file system to make it persistent across restarts. It is internal and cannot be read or altered manually. All the operations on a WORM-enabled file are based on the secure clock and not on the system clock. For example, for WORM-enabled files, InfoScale uses this clock to check whether the retention period is expired.

A lag may occur between a secure clock and the system clock in certain situations, like downtime for scheduled or unscheduled maintenance, or when the cluster is offline. To address such situations, the secure clock is programed to auto-correct and catch up with the system clock. The auto-correct mechanism increments the secure clock by half a second after every minute to eventually match the system clock. Thus, the secure clock is adjusted for a maximum of 12 minutes per day, which is approximately three days per year. These adjustments are not made if the secure clock matches the system clock.

To enable the auto-correct mechanism, either use the `autocorr` option while creating a new file system or use the `fsadm` utility in case of an existing file system.

Audit logging

An audit log records events in the life cycle of a WORM-enabled file, from the instant the WORM flag is set on a file until the file is deleted. You can use the `-aulog` option of the `mkfs` command or the `fsadm` command to enable audit logging for a new file system or an existing file system respectively. Once enabled, you cannot disable audit logging for a file system. InfoScale creates an audit log file for a file system only if audit logging is enabled for that file system. The audit logging feature is supported with DLV 16 and later.

An audit log is stored in a non-readable and non-editable format. Only the users with administrator privileges can view or read the audit log by using the `vxauditlogadm` utility. A record in audit log is retained until the corresponding WORM-enabled file is deleted.

The audit log file records the events similar but not limited to the following events on WORM-enabled objects (files and non-modifiable checkpoints):

- Failure events on the WORM-enabled objects
- Types of events that occur on an object, for example:
  - Setting the WORM flag
  - Extending the retention period
  - Deleting the WORM-enabled object
  - Attempting to modify the WORM-enabled object
- Information on who (process or user) triggers the event and when

Non-modifiable storage checkpoints

The Storage Checkpoint feature of VxFS lets you create a persistent image of a file system at a specific time. You can modify the data on a standard storage checkpoint by mounting the checkpoint in the read-write (RW) mode. To protect data from being modified due to ransomware attacks, InfoScale lets you create non-modifiable checkpoints, which can be mounted only in the read-only (RO) mode.

You can also specify a retention period for a WORM-enabled checkpoint as a date before which the checkpoint cannot be deleted. During this period, the checkpoint data is retained as-is for security or for governance purposes. After the retention period expires, you can delete the checkpoint. If audit logging is enabled, records of the create, rename, mount, and delete operations for non-modifiable checkpoints are maintained.

Note: All the retention period limitations that are applicable to WORM-enabled files are also applicable to WORM-enabled checkpoints. However, a removable checkpoint is not deleted unless its retention period has expired, and the ENOSPC condition may occur as a result.

Post-upgrade tasks to enable the use of non-modifiable checkpoints

After you upgrade the `VRTSvxfs` package to 8.0, first use the `vxupgrade` command with the appropriate options to upgrade the any mounted file systems to DLV 17. If a file system is not already WORM-enabled, use the `fsadm` command with the appropriate options to enable the feature.

Creating non-modifiable checkpoints

Non-modifiable checkpoints can be created only on WORM-enabled or soft WORM-enabled VxFS file systems and are therefore also referred to as WORM-enabled or soft WORM-enabled checkpoints.

To create a non-modifiable storage checkpoint

1. Create a WORM-enabled file system.

   Sample command and output:

   ```
   # mkfs -t vxfs -o worm /dev/vx/dsk/testdg/vol1
   ```

   ```
   version 17 layout
   20971520 sectors, 10485760 blocks of size 1024, log size 16384 blocks
   rcq size 1024 blocks
   largefiles supported
   maxlink supported
   WORM supported
   maxts supported
   ```

2. Mount the file system.

   Sample command and output:

   ```
   # mount -t vxfs /dev/vx/dsk/testdg/vol1 /mnt1
   ```

3 Verify that the file system is WORM-enabled.

Sample command and output:

```
# /opt/VRTS/bin/fsadm /mnt

largefiles,maxlink,worm,maxts
```

The presence of the **worm** string in the output indicates that the file system is WORM-enabled.

4 Create a non-modifiable storage checkpoint.

The extended syntax of the `fsckptadm` command is:

```
# /opt/VRTS/bin/fsckptadm [-v[r | R]u[n | w[-t <yyyy-mm-dd>]]]
create <checkpoint_name> [-S <signature_value>] [-D <data_policy>]
[-M <metadata_policy>] <mountpoint_name>
```

The minimum required parameters are:

```
# /opt/VRTS/bin/fsckptadm -w create <checkpoint_name>
<mountpoint_name>
```

Sample command and output:

```
# /opt/VRTS/bin/fsckptadm -w create clone1 /mnt1
```

Alternatively, you can also create non-modifiable storage checkpoints on multiple file systems simultaneously by using the `createall` option.

Sample commands and output:

```
# /opt/VRTS/bin/fsckptadm -w createall clone4 /mnt2 /mnt3

# /opt/VRTS/bin/fsckptadm list /mnt2 /mnt3

/mnt2
clone4:
    ctime            = Thu 24 Jun 2021 03:54:38 PM IST
    mtime            = Thu 24 Jun 2021 03:54:38 PM IST
    Retention period = 0
    flags            = largefiles, worm
/mnt3
clone4:
    ctime            =  Thu 24 Jun 2021 03:54:38 PM IST
    mtime            =  Thu 24 Jun 2021 03:54:38 PM IST
    Retention period =  0
    flags            = largefiles, worm
```

5 Verify that the checkpoint is WORM-enabled.

Sample command and output:

```
# /opt/VRTS/bin/fsckptadm list /mnt1
```

```
/mnt1
clone1:
ctime            = Thu 24 Jun 2021 03:32:13 PM IST
mtime            = Thu 24 Jun 2021 03:32:13 PM IST
Retention period = 0
flags            = largefiles, worm
```

6 Verify that the WORM-enabled checkpoint can be mounted in the RO mode.

Sample commands and output:

```
# mount -t vxfs -o ro,ckpt=clone1 /dev/vx/dsk/testdg/vol1:clone1
/mnt2
```

```
# mount | grep clone1
```

```
/dev/vx/dsk/testdg/vol1:clone1 on /mnt2 type vxclonefs
(ro,seclabel,largefiles,ckpt=clone1,ioerror=mwdisable)
```

7 Verify that the WORM-enabled checkpoint cannot be mounted in the RW mode.

Sample command and output:

```
# mount -t vxfs -o rw,ckpt=clone1 /dev/vx/dsk/testdg/vol1:clone1
/mnt3
```

```
UX:vxfs mount.vxfs: ERROR: V-3-23731: mount failed.
```

Setting retention periods for non-modifiable checkpoints

If you do not specify a retention period, a WORM-enabled checkpoint can be deleted at any time.

To create a non-modifiable storage checkpoint with a retention period

1. Set a retention period at the time of the WORM-enabled checkpoint creation.

   Sample command:

   ```
   # /opt/VRTS/bin/fsckptadm -w -t 2022-05-01 create clone5 /mnt1
   ```

2. Verify that the retention period is set for the WORM-enabled checkpoint.

   Sample command and output:

   ```
   # /opt/VRTS/bin/fsckptadm list /mnt1
   ```

   ```
   /mnt1
   clone5:
       ctime            = Thu 24 Jun 2021 04:04:39 PM IST
       mtime            = Thu 24 Jun 2021 04:04:39 PM IST
       Retention period = Sun 01 May 2022 12:00:00 AM IST
       flags            = largefiles, worm
   ```

To create a non-modifiable storage checkpoint with the same retention period for multiple file systems

1. Alternatively, you can set a common retention period at the time of WORM-enabled checkpoint creation on multiple filesystems simultaneously.

   Sample command:

   ```
   # /opt/VRTS/bin/fsckptadm -w -t 2022-05-01 createall clone6 /mnt2
   /mnt3
   ```

2. Verify that the same retention period is set for all the WORM-enabled checkpoints.

   Sample command and output:

   ```
   # /opt/VRTS/bin/fsckptadm list /mnt2 /mnt3

   /mnt2
   clone6:
       ctime            = Thu 24 Jun 2021 04:07:33 PM IST
       mtime            = Thu 24 Jun 2021 04:07:33 PM IST
       Retention period = Sun 01 May 2022 12:00:00 AM IST
       flags            = largefiles, worm
   /mnt3
   clone6:
       ctime            = Thu 24 Jun 2021 04:07:33 PM IST
       mtime            = Thu 24 Jun 2021 04:07:33 PM IST
       Retention period = Sun 01 May 2022 12:00:00 AM IST
       flags            = largefiles, worm
   ```

To set a retention period for an existing WORM-enabled checkpoint

- ◆ Specify the date until which an existing WORM-enabled checkpoint should be retained.

  Sample command and output:

  ```
  # /opt/VRTS/bin/fsckptadm -v setretention 2023-02-08 clone5 /mnt1

  clone5:
      ctime            = Thu 24 Jun 2021 04:04:39 PM IST
      mtime            = Thu 24 Jun 2021 04:04:39 PM IST
      Retention period = Wed 08 Feb 2023 12:00:00 AM IST
      flags            = largefiles, worm
  ```

To extend the retention period of an existing WORM-enabled checkpoint

- Specify a date beyond the one that is currently assigned to a WORM-enabled checkpoint to extend its retention period.

  Sample command and output:

  ```
  # /opt/VRTS/bin/fsckptadm -v setretention 2050-02-08 clone5 /mnt1

  clone5:
      ctime            = Thu 24 Jun 2021 04:04:39 PM IST
      mtime            = Thu 24 Jun 2021 04:04:39 PM IST
      Retention period = Tue 08 Feb 2050 12:00:00 AM IST
      flags            = largefiles, worm
  ```

Note: The maximum supported retention timestamp on a WORM-enabled checkpoint is 2038-01-19. However, if MAXTS is enabled on the file system, the maximum supported timestamp is 2106-02-07 (UINT_MAX).

Making an existing checkpoint non-modifiable

InfoScale lets you make existing storage checkpoints non-modifiable and also specify retention periods for them.

To make an existing checkpoint non-modifiable

1 Set the worm flag for the checkpoint.

Sample command:

```
# /opt/VRTS/bin/fsckptadm set worm clone7 /mnt1
```

2 Optionally, set a retention period for the checkpoint.

Sample command:

```
# /opt/VRTS/bin/fsckptadm setretention 2050-02-08 clone7 /mnt1
```

Note: To set a retention period for an existing checkpoint, it must be explicitly WORM-enabled.

3 Verify the changes that you made to the checkpoint.

Sample commands and output:

```
# /opt/VRTS/bin/fsckptadm list clone7 /mnt1
```

```
# /opt/VRTS/bin/fsckptadm list /mnt1
```

```
/mnt1
clone7:
    ctime            = Thu 24 Jun 2021 04:23:08 PM IST
    mtime            = Thu 24 Jun 2021 04:23:08 PM IST
    Retention period = Tue 08 Feb 2050 12:00:00 AM IST
    flags            = largefiles, worm
```

Deletion of non-modifiable checkpoints

The `fsckptadm remove` command fails to remove a checkpoint and returns a "permission denied" error if the retention period has not yet expired. When an ENOSPC condition occurs, VxFS checks for the presence of the removable flag as well as the retention period on storage checkpoints. If the removable flag is set on a WORM-enabled checkpoint and if its retention period has expired, the checkpoint is removed.

Compatibility of WORM flag with relevant checkpoint operations

Operation and the minimum required parameters for the corresponding commands	Compatibility description
create `fsckptadm [-vn[r | R]uwt] create <checkpoint_name>`	■ The creation of checkpoints with the WORM flag and a retention period is supported with the NOMOUNT flag. The `-wt` options work with the `-u` option of the `fsckptadm create` command. ■ The creation of checkpoints with the WORM flag is supported with the REMOVE flag. A checkpoint that has the REMOVE flag set is not removed unless its retention period has expired. The `-wt` options work with the `-r` option of the `fsckptadm create` command. ■ The creation of checkpoints with the WORM flag is not supported with the NODATA flag. Modification of either the data or the metadata by external processes is not permitted on WORM-enabled checkpoints. The `-wt` options do not work with the `-n` option of the `fsckptadm create` command.
createall `fsckptadm [-vn[r | R]u] createall <checkpoint_name> [-S <signature_value>] [-D <data_policy>] [-M <metadata_policy>] <mountpoint_name>`	■ The creation of checkpoints on multiple file systems simultaneously with the WORM flag and a retention period is supported with the NOMOUNT flag. The `-wt` options work with the `-u` option of the `fsckptadm createall` command. ■ The creation of checkpoints on multiple file systems simultaneously with the WORM flag is supported with the REMOVE flag. A checkpoint that has the REMOVE flag set is not removed unless its retention period has expired. The `-wt` options work with the `-r` option of the `fsckptadm createall` command. ■ The creation of checkpoints on multiple file systems simultaneously with the WORM flag is not supported with NODATA flag. The `-wt` option does not work with the `-n` option of the `fsckptadm createall` command.
set `fsckptadm [-vsf] set <nodata|remove|nomount|worm> <checkpoint_name> <mountpoint_name>`	■ The NOMOUNT and the REMOVE flags are set on checkpoints that have the WORM flag set and vice versa. ■ The NODATA flag is not set on checkpoints that have the WORM flag set and vice versa.
clear `fsckptadm [-v] clear <remove|nomount> <checkpoint_name> <mountpoint_name>`	Once the WORM flag is set on a checkpoint, it cannot be cleared.

Operation and the minimum required parameters for the corresponding commands	Compatibility description
rename `fsckptadm [-v] rename <old-checkpoint_name> <new_checkpoint_name> <mountpoint_name>`	The rename operation is not supported on WORM-enabled checkpoints.
remove `fsckptadm [-vsf] remove <checkpoint_name> <mountpoint_name>`	The remove operation fails on WORM-enabled checkpoints until the retention period is in effect. If a retention period is not set on a WORM-enabled checkpoint, or if its retention period has expired, it can be deleted.

Restrictions and limitations on the promote operation for checkpoints

- The WORM capability of a checkpoint can be lost if it is mounted in the RW mode. Therefore, the promote operation is not allowed on WORM-enabled checkpoints.
- Similarly, the `fsckpt restore` command is not allowed on such checkpoints.
- During the promote operation of a regular checkpoint, any intermediate WORM-enabled checkpoint can be deleted if a retention period is not set for it or if its retention period has expired.

  For example, consider a file system, FS1, that has checkpoints C1, C2, and C3, where C3 is the latest created checkpoint. C1 and C2 have their respective retention periods set, but no retention period is set for C3 and a promote operation is performed on C3. In this scenario, C1 and C2 get deleted, which counters their retention period setting. Therefore, before you run a promote operation, make sure that either one of the following conditions are satisfied:
  - None of the intermediate checkpoints have retention periods set.
  - The retention periods of all the intermediate checkpoints have expired.

Restrictions and limitations on the mount operations

- Checkpoints that have the VX_FSET_WORM flag set are only allowed to be mounted in the RO mode.
- The `protected=on` mount option is not supported with WORM-enabled checkpoints.

- When the `automount` option is used with the RW mode, the WORM-enabled checkpoints are mounted in the RO mode internally, while the rest are mounted in the RW mode.

Soft WORM storage

InfoScale provides an option that lets you enable the soft WORM setting on file systems, either at the time of creation or by modifying them later. Later, a root user can reduce or remove the retention periods of files and storage checkpoints that are created on the file system. Soft WORM supports all the semantics of WORM. Once enabled, you cannot disable the soft WORM setting of a file system. However, you can upgrade it to use the WORM setting—also referred to as the compliance mode—so as to block the reduction or removal of its retention period. The retention period of soft WORM-enabled files and checkpoints can be reduced or removed, but their data or metadata cannot be modified, and they cannot be mounted in the RW mode.

If audit logging is enabled, records of the create, rename, mount, and delete operations of all soft WORM-enabled entities are maintained.

Post-upgrade tasks to enable the use of soft WORM-enabled storage

After you upgrade the `VRTSvxfs` package to 8.0, first use the `vxupgrade` command with the appropriate options to upgrade any mounted file systems to DLV 17. Use the `fsadm` command with the appropriate options to enable the feature.

Creating WORM-enabled entities

The following procedure uses examples to describe how soft WORM-enabled file systems and files can be created. Storage checkpoints can also be similarly created.

To create a soft WORM-enabled file

1. Soft WORM-enable and existing file system using the command:

   ```
   # /opt/VRTS/bin/fsadm -o softworm <path_of_mountpoint>
   ```

   Alternatively, create a new soft WORM-enabled file system.

   Sample command and output:

   ```
   # mkfs -t vxfs -o softworm /dev/vx/rdsk/testdg/vol1

   version 17 layout
   125829120 sectors, 62914560 blocks of size 1024, log size 65536 blocks
   rcq size 4096 blocks
   largefiles supported
   maxlink supported
   SOFTWORM supported
   maxts supported
   ```

2. Mount the file system.

   Sample command:

   ```
   # mount -t vxfs /dev/vx/dsk/testdg/vol1 /mnt1
   ```

3. Verify that the file system is soft WORM-enabled.

   ```
   # /opt/VRTS/bin/fsadm /mnt1

   largefiles,maxlink,softworm,maxts
   ```

   The presence of the **softworm** string in the output indicates that the file system is WORM-enabled.

4. Create a file.

   Sample command and output:

   ```
   # dd if=/dev/urandom of=/mnt1/a1 bs=1024 count=10

   10+0 records in
   10+0 records out
   10240 bytes (10 kB, 10 KiB) copied, 0.000653719 s, 15.7 MB/s
   ```

5. Set a retention period on the file by performing the following tasks sequentially:

   - Change the access time of the file so that it has the same value as the retention period, by using the touch command.

     ```
     # touch -at YYYYMMDDhhmm.ss <file_name>
     ```

For example, if a file named `a1` must be retained till 27th June 2036 09:49:50 pm, run:

```
# touch -at 203606270949.50 /mnt1/a1
```

- Mark the file as read-only by changing its permissions.
  For example, to make the `a1` file read-only, run:

  ```
  # chmod -w /mnt1/a1
  ```

6. Reduce the retention period of the file.

   For example, if a file named `a1` must be retained till 27th June 2033 09:49:50 pm, run:

   ```
   # touch -at "203306270949.50" /mnt1/a1
   ```

7. Optionally, upgrade the file from soft WORM-enabled to WORM-enabled.

   Sample command:

   ```
   # /opt/VRTS/bin/fsadm -o worm <path_of_mountpoint>
   ```

   Note: You cannot change a WORM-enabled file to a soft WORM-enabled file.

Limitations

All the limitations that are applicable to WORM-enabled entities are also applicable to soft WORM-enabled entities, except that the root user can reduce their individual retention periods. Additionally, truncation or deletion of soft WORM-related audit logs is not yet supported.

Section 10

Reference

Appendix A

Reverse path name lookup

This appendix includes the following topics:

- About reverse path name lookup

About reverse path name lookup

The reverse path name lookup feature obtains the full path name of a file or directory from the inode number of that file or directory. The inode number is provided as an argument to the `vxlsino` administrative command, or the `vxfs_inotopath_gen`(3) application programming interface library function.

The reverse path name lookup feature can be useful for a variety of applications, such as for clients of the VxFS File Change Log feature, in backup and restore utilities, and for replication products. Typically, these applications store information by inode numbers because a path name for a file or directory can be very long, thus the need for an easy method of obtaining a path name.

An inode is a unique identification number for each file in a file system. An inode contains the data and metadata associated with that file, but does not include the file name to which the inode corresponds. It is therefore relatively difficult to determine the name of a file from an inode number. The `ncheck` command provides a mechanism for obtaining a file name from an inode identifier by scanning each directory in the file system, but this process can take a long period of time. The VxFS reverse path name lookup feature obtains path names relatively quickly.

Note: Because symbolic links do not constitute a path to the file, the reverse path name lookup feature cannot track symbolic links to files.

Because of the possibility of errors with processes renaming or unlinking and creating new files, it is advisable to perform a `lookup` or `open` with the path name and verify that the inode number matches the path names obtained.

See the `vxlsino`(1M), `vxfs_inotopath_gen`(3), and `vxfs_inotopath`(3) manual pages.

Tunable parameters

This appendix includes the following topics:

About tuning Storage Foundation

Storage Foundation (SF) is widely used in a range of environments where performance plays a critical role. SF has a number of tunable parameters and configuration options that are meant to enable customization of the stack for the particular environment and workload in which SF is used. This guide helps administrators understand how some of these options affect performance, and provides guidelines for tuning the options.

This guide focuses on the important tunable parameters for SF. For good performance for a workload, you might also need to tune other entities of the stack, such as the operating system, database, storage network, or disk array. While this guide discusses some of these parameters briefly, you should refer to the documentation from the entity's vendor for more information on how to tune these entities.

Tuning the VxFS file system

This section describes the following kernel tunable parameters in VxFS:

Tuning inode table size

VxFS caches inodes in an inode table. The tunable for VxFS to determine the number of entries in its inode table is `vxfs_ninode`.

VxFS uses the value of vxfs_ninode in `/etc/modprobe.conf` as the number of entries in the VxFS inode table. By default, the file system uses a value of `vxfs_ninode`, which is computed based on system memory size. To increase the value, make the following change in `/etc/modprobe.conf` and reboot:

```
options vxfs vxfs_ninode=new_value
```

The new parameters take affect after a reboot or after the VxFS module is unloaded and reloaded. The VxFS module can be loaded using the `modprobe` command or automatically when a file system is mounted.

See the `modprobe`(8) manual page.

Note: New parameters in the `/etc/modprobe.conf` file are not read by the `insmod vxfs` command.

Tuning performance optimization of inode allocation

The `delicache_enable` tunable parameter specifies whether performance optimization of inode allocation and inode reuse during a new file creation is turned on or off. The `delicache_enable` tunable is also supported for cluster file systems. You can specify the following values for`delicache_enable`:

- 0 – Disables delicache optimization.
- 1 – Enables delicache optimization.

The default value of `delicache_enable` is 1.

Tuning file system parallel direct I/O

OnVxFS, each iovec is performed synchronously for the readv(2) call and writev(2) call. For both readv(2) and writev(2), the Single Unix Specification states, "The readv/writev() function shall always fill an area completely before proceeding to the next." However, for direct I/O, Linux ignores this requirement and submits a number of iovecs in parallel before waiting for completion. Veritas File System (VxFS) performs parallel direct I/O for both reads and writes, which improves VxFS performance. This support for parallel direct I/O can be enabled by setting the VxFS module load tunable vx_parallel_dio.

To enable parallel direct I/O, make the following change in the /etc/modprobe.conf file and reboot the system:

```
options vxfs vx_parallel_dio=1
```

Partitioned directories

You can enable or disable the partitioned directories feature by setting the pdir_enable tunable. Specifying a value of 1 enables partitioned directories, while specifying a value of 0 disables partitioned directories.

The default value is 0.

You can set the pdir_threshold tunable to specify the threshold value in terms of directory size in bytes beyond which VxFS will partition a directory if you enabled partitioned directories. The default value is 32768.

The -d option of the fsadm command removes empty hidden directories from partitioned directories. If you disabled partitioned directories, the fsadm -d command also converts partitioned directories to regular directories.

The partitioned directories feature operates only on disk layout Version 8 or later file systems.

Warning: If the directories are huge, conversion between partitioned directories and regular directories or vice versa needs some time. If you enable the feature when the root directory already contains a large number of files, the conversion can occur at file system mount time, and can cause the mount to take a long time. Veritas recommends that the conversion is performed when directories are slightly or not populated.

Veritas Volume Manager maximum I/O size

When using VxFS with Veritas Volume Manager (VxVM), VxVM by default breaks up I/O requests larger than 256K. When using striping, to optimize performance, the file system issues I/O requests that are up to a full stripe in size. If the stripe size is larger than 256K, those requests are broken up.

To avoid undesirable I/O breakup, you can increase the maximum I/O size by changing the value of the `vol_maxio` parameter in the `/etc/modprobe.conf` file.

Native asynchronous I/O with cloned processes

You can enable or disable native asynchronous I/O with cloned processes by setting the `vx_allow_cloned_naio` tunable. Specifying a value of 1 enables native asynchronous I/O with cloned processes, while specifying a value of 0 disables native asynchronous I/O with cloned processes. The default value is 0.

Processes that are cloned by using the `CLONE_VM` flag share an address space with their parent. When such threads issue native asynchronous I/O by using the `io_submit`() call, the system can panic if those threads return and exit before the I/O completes. You can avoid this issue by setting the `vx_allow_cloned_naio` tunable to 0, which causes such threads to issue the I/O synchronously.

Well-behaved applications that do not have threads exiting with pending asynchronous I/O do not have this restriction. When using such applications, such as Sybase, you can set the `vx_allow_cloned_naio` tunable to 1, which avoids the performance impact of such threads having asynchronous I/O become synchronous.

DMP tunable parameters

DMP provides various parameters that you can use to tune your environment.

Table B-1 shows the DMP parameters that can be tuned. You can set a tunable parameter online, without a reboot.

Table B-1 DMP parameters that are tunable

Parameter	Description
`dmp_cache_open`	If this parameter is set to `on`, the first open of a device is cached. This caching enhances the performance of device discovery by minimizing the overhead that is caused by subsequent opens on the device. If this parameter is set to `off`, caching is not performed. The default value is `on`.

Table B-1 DMP parameters that are tunable *(continued)*

Parameter	Description
`dmp_daemon_count`	The number of kernel threads that are available for servicing path error handling, path restoration, and other DMP administrative tasks. The default number of threads is 10.
`dmp_delayq_interval`	How long DMP should wait before retrying I/O after an array fails over to a standby path. Some disk arrays are not capable of accepting I/O requests immediately after failover. The default value is 15 seconds.
`dmp_display_alua_states`	For ALUA arrays, this tunable displays the asymmetric access state instead of PRIMARY or SECONDARY state in the PATH-TYPE[M] column. The asymmetric access state can be: ■ Active/Optimized ■ Active/Non-optimized ■ Standby ■ Unavailable ■ TransitionInProgress ■ Offline The default tunable value is on.
`dmp_fast_recovery`	Whether DMP should try to obtain SCSI error information directly from the HBA interface. Setting the value to `on` can potentially provide faster error recovery, if the HBA interface supports the error enquiry feature. If this parameter is set to `off`, the HBA interface is not used. The default setting is `on`.

Table B-1 DMP parameters that are tunable *(continued)*

Parameter	Description
`dmp_health_time`	DMP detects intermittently failing paths, and prevents I/O requests from being sent on them. The value of `dmp_health_time` represents the time in seconds for which a path must stay healthy. If a path's state changes back from enabled to disabled within this time period, DMP marks the path as intermittently failing, and does not re-enable the path for I/O until `dmp_path_age` seconds elapse. The default value is 60 seconds. A value of 0 prevents DMP from detecting intermittently failing paths.
`dmp_log_level`	The level of detail that is displayed for DMP console messages. The following level values are defined: 1 — Displays all DMP log messages that are critical. 2 — Displays level 1 messages plus messages that relate to path or disk addition or removal, SCSI errors, IO errors and DMP node migration. 3 — Displays level 1 and 2 messages plus messages that relate to path throttling, suspect path, idle path and insane path logic. 4 — Displays level 1, 2 and 3 messages plus messages that relate to setting or changing attributes on a path and tunable related changes. 5 or higher — Displays level 1, 2, 3 and 4 messages plus more verbose messages. The default value is 1.
`dmp_low_impact_probe`	Determines if the path probing by restore daemon is optimized or not. Set it to `on` to enable optimization and `off` to disable. Path probing is optimized only when restore policy is check_disabled or during check_disabled phase of check_periodic policy. The default value is `on`.

Table B-1 DMP parameters that are tunable *(continued)*

Parameter	Description
`dmp_lun_retry_timeout`	Specifies a retry period for handling transient errors that are not handled by the HBA and the SCSI driver. Specify the time in seconds. In general, no such special handling is required. Therefore, the default value of the `dmp_lun_retry_timeout` tunable parameter is 30. When all paths to a disk fail, DMP fails the I/Os to the application. The paths are checked for connectivity only once. In special cases when DMP needs to handle the transient errors, configure DMP to delay failing the I/Os to the application for a short interval. Set the `dmp_lun_retry_timeout` tunable parameter to a non-zero value to specify the interval. If all of the paths to the LUN fail and I/Os need to be serviced, then DMP probes the paths every five seconds for the specified interval. If the paths are restored within the interval, DMP detects this and retries the I/Os. DMP does not fail I/Os to a disk with all failed paths until the specified `dmp_lun_retry_timeout` interval or until the I/O succeeds on one of the paths, whichever happens first.
`dmp_monitor_fabric`	Determines if DMP should register for HBA events from SNIA HAB APIs. These events improve the failover performance by proactively avoiding the I/O paths that have impending failure. The default setting is `off` for releases before 5.0 that have been patched to support this DDL feature. The default setting is `on` for 5.0 and later releases.
`dmp_monitor_ownership`	Determines whether the ownership monitoring is enabled for ALUA arrays. When this tunable is set to on, DMP polls the devices for LUN ownership changes. The polling interval is specified by the dmp_restore_interval tunable. The default value is `on`. When the dmp_monitor_ownership tunable is `off`, DMP does not poll the devices for LUN ownership changes.

Table B-1 DMP parameters that are tunable *(continued)*

Parameter	Description
`dmp_native_support`	Determines whether DMP will do multi-pathing for native devices. Set the tunable to `on` to have DMP do multi-pathing for native devices. When Dynamic Multi-Pathing is installed as a component of Storage Foundation, the default value is `off`. When Dynamic Multi-Pathing is installed as a stand-alone product, the default value is `on`.
`dmp_path_age`	The time for which an intermittently failing path needs to be monitored as healthy before DMP again tries to schedule I/O requests on it. The default value is 300 seconds. A value of 0 prevents DMP from detecting intermittently failing paths.
`dmp_pathswitch_blks_shift`	The default number of contiguous I/O blocks that are sent along a DMP path to an array before switching to the next available path. The value is expressed as the integer exponent of a power of 2; for example 9 represents 512 blocks. The default value is 9. In this case, 512 blocks (256k) of contiguous I/O are sent over a DMP path before switching. For intelligent disk arrays with internal data caches, better throughput may be obtained by increasing the value of this tunable. For example, for the HDS 9960 A/A array, the optimal value is between 15 and 17 for an I/O activity pattern that consists mostly of sequential reads or writes. This parameter only affects the behavior of the `balanced` I/O policy. A value of 0 disables multi-pathing for the policy unless the `vxdmpadm` command is used to specify a different partition size for an array. See “Specifying the I/O policy” on page 240.

Table B-1 DMP parameters that are tunable *(continued)*

Parameter	Description
`dmp_probe_idle_lun`	If DMP statistics gathering is enabled, set this tunable to `on` (default) to have the DMP path restoration thread probe idle LUNs. Set this tunable to `off` to turn off this feature. (Idle LUNs are VM disks on which no I/O requests are scheduled.) The value of this tunable is only interpreted when DMP statistics gathering is enabled. Turning off statistics gathering also disables idle LUN probing. The default value is `on`.
`dmp_probe_threshold`	If the dmp_low_impact_probe is turned `on`, dmp_probe_threshold determines the number of paths to probe before deciding on changing the state of other paths in the same subpath failover group. The default value is 5.
`dmp_restore_cycles`	If the DMP restore policy is `check_periodic`, the number of cycles after which the `check_all` policy is called. The default value is 10. See "Configuring DMP path restoration policies" on page 254.
`dmp_restore_interval`	The interval attribute specifies how often the path restoration thread examines the paths. Specify the time in seconds. The default value is 300. The value of this tunable can also be set using the `vxdmpadm start restore` command. See "Configuring DMP path restoration policies" on page 254.

Table B-1 DMP parameters that are tunable *(continued)*

Parameter	Description
`dmp_restore_policy`	The DMP restore policy, which can be set to one of the following values: ■ `check_all` ■ `check_alternate` ■ `check_disabled` ■ `check_periodic` The default value is `check_disabled` The value of this tunable can also be set using the `vxdmpadm start restore` command. See "Configuring DMP path restoration policies" on page 254.
`dmp_restore_state`	If this parameter is set to `enabled`, it enables the path restoration thread to be started. See "Configuring DMP path restoration policies" on page 254. If this parameter is set to `disabled`, it stops and disables the path restoration thread. If this parameter is set to `stopped`, it stops the path restoration thread until the next device discovery cycle. The default is `enabled`. See "Stopping the DMP path restoration thread" on page 255.
`dmp_scsi_timeout`	Determines the timeout value to be set for any SCSI command that is sent via DMP. If the HBA does not receive a response for a SCSI command that it has sent to the device within the timeout period, the SCSI command is returned with a failure error code. The default value is 20 seconds.
`dmp_sfg_threshold`	Determines the minimum number of paths that should be failed in a failover group before DMP starts suspecting other paths in the same failover group. The value of 0 disables the failover logic based on subpath failover groups. The default value is 1.

Table B-1 DMP parameters that are tunable *(continued)*

Parameter	Description
`dmp_stat_interval`	The time interval between gathering DMP statistics. The default and minimum value are 1 second.

Methods to change Dynamic Multi-Pathing tunable parameters

Dynamic Multi-Pathing (DMP) provides a variety of parameters that you can use to tune your configuration.

See "DMP tunable parameters" on page 796.

Change the DMP tunable parameters with one of the following methods:

Use the `vxdmpadm settune` command to display or modify the values.	See "Changing the values of DMP parameters with the vxdmpadm settune command line" on page 803.
Use the template method of the `vxdmpadm` command.	See "About tuning Dynamic Multi-Pathing (DMP) with templates" on page 803.

Changing the values of DMP parameters with the vxdmpadm settune command line

To set a DMP timable parameter, use the following command:

```
# vxdmpadm settune dmp_tunable=value
```

To display the values of the DMP tunable parameters, use the following command:

```
# vxdmpadm gettune [dmp_tunable]
```

You can also use the template method to view or change DMP tunable parameters.

See "About tuning Dynamic Multi-Pathing (DMP) with templates" on page 803.

About tuning Dynamic Multi-Pathing (DMP) with templates

Dynamic Multi-Pathing has multiple tunable parameters and attributes that you can configure for optimal performance. DMP provides a template method to update several tunable parameters and attributes with a single operation. The template represents a full or partial DMP configuration, showing the values of the parameters and attributes of the host.

To view and work with the tunable parameters, you can dump the configuration values of the DMP tunable parameters to a file. Edit the parameters and attributes, if required. Then, load the template file to a host to update all of the values in a single operation.

You can load the configuration file to the same host, or to another similar host. The template method is useful for the following scenarios:

- Configure multiple similar hosts with the optimal performance tuning values.

  Configure one host for optimal performance. After you have configured the host, dump the tunable parameters and attributes to a template file. You can then load the template file to another host with similar requirements. Veritas recommends that the hosts that use the same configuration template have the same operating system and similar I/O requirements.

- Define multiple specialized templates to handle different I/O load requirements.

  When the load changes on a host, you can load a different template for the best performance. This strategy is appropriate for predictable, temporary changes in the I/O load. As the system administrator, after you define the system's I/O load behavior, you can customize tuning templates for particular loads. You can then automate the tuning, since there is a single load command that you can use in scripts or cron jobs.

At any time, you can reset the configuration, which reverts the values of the tunable parameters and attributes to the DMP default values.

You can manage the DMP configuration file with the `vxdmpadm config` commands.

See the `vxdmpadm`(1m) man page.

DMP tuning templates

The template mechanism enables you to tune DMP parameters and attributes by dumping the configuration values to a file, or to standard output.

DMP supports tuning the following types of information with template files:

- DMP tunable parameters.
- DMP attributes defined for an enclosure, array name, or array type.
- Veritas naming scheme parameters.

The template file is divided into sections, as follows:

DMP Tunables	Applied to all enclosures and arrays.

Namingscheme	Applied to all enclosures and arrays.
Arraytype	Use to customize array types. Applied to all of the enclosures of the specified array type.
Arrayname	Use if particular arrays need customization; that is, if the tunables vary from those applied for the array type. Attributes in this section are applied to all of the enclosures of the specified array name.
Enclosurename	Applied to the enclosures of the specified Cab serial number and array name. Use if particular enclosures need customization; that is, if the tunables vary from those applied for the array type and array name.

Loading is atomic for the section. DMP loads each section only if all of the attributes in the section are valid. When all sections have been processed, DMP reports the list of errors and warns the user. DMP does not support a partial rollback. DMP verifies the tunables and attributes during the load process. However, Veritas recommends that you check the configuration template file before you attempt to load the file. Make any required corrections until the configuration file validates correctly.

The attributes are given priority in the following order when a template is loaded:

Enclosure Section > Array Name Section > Array Type Section

If all enclosures of the same array type need the same settings, then remove the corresponding array name and enclosure name sections from the template. Define the settings only in the array type section. If some of the enclosures or array names need customized settings, retain the attribute sections for the array names or enclosures. You can remove the entries for the enclosures or the array names if they use the same settings that are defined for the array type.

When you dump a configuration file from a host, that host may contain some arrays which are not visible on the other hosts. When you load the template to a target host that does not include the enclosure, array type, or array name, DMP ignores the sections.

You may not want to apply settings to non-shared arrays or some host-specific arrays on the target hosts. Be sure to define an enclosure section for each of those arrays in the template. When you load the template file to the target host, the enclosure section determines the settings. Otherwise, DMP applies the settings from the respective array name or array type sections.

Example DMP tuning template

This section shows an example of a DMP tuning template.

```
DMP Tunables
        dmp_cache_open=on
        dmp_daemon_count=10
        dmp_delayq_interval=15
        dmp_restore_state=enabled
        dmp_fast_recovery=on
        dmp_health_time=60
        dmp_log_level=1
        dmp_low_impact_probe=on
        dmp_lun_retry_timeout=30
        dmp_path_age=300
        dmp_pathswitch_blks_shift=9
        dmp_probe_idle_lun=on
        dmp_probe_threshold=5
        dmp_restore_cycles=10
        dmp_restore_interval=300
        dmp_restore_policy=check_disabled
        dmp_retry_count=5
        dmp_scsi_timeout=20
        dmp_sfg_threshold=1
        dmp_stat_interval=1
        dmp_monitor_ownership=on
        dmp_monitor_fabric=on
        dmp_native_support=off
Namingscheme
        namingscheme=ebn
        persistence=yes
        lowercase=yes
        use_avid=yes
Arraytype
        arraytype=CLR-A/PF
        iopolicy=minimumq
        partitionsize=512
        recoveryoption=nothrottle
        recoveryoption=timebound iotimeout=300
        redundancy=0
Arraytype
        arraytype=ALUA
        iopolicy=adaptive
        partitionsize=512
        use_all_paths=no
        recoveryoption=nothrottle
        recoveryoption=timebound iotimeout=300
        redundancy=0
Arraytype
        arraytype=Disk
        iopolicy=minimumq
        partitionsize=512
        recoveryoption=nothrottle
        recoveryoption=timebound iotimeout=300
        redundancy=0
Arrayname
        arrayname=EMC_CLARiiON
        iopolicy=minimumq
        partitionsize=512
        recoveryoption=nothrottle
        recoveryoption=timebound iotimeout=300
        redundancy=0
Arrayname
        arrayname=EVA4K6K
        iopolicy=adaptive
        partitionsize=512
        use_all_paths=no
        recoveryoption=nothrottle
        recoveryoption=timebound iotimeout=300
        redundancy=0
Arrayname
        arrayname=Disk
        iopolicy=minimumq
        partitionsize=512
        recoveryoption=nothrottle
        recoveryoption=timebound iotimeout=300
        redundancy=0
Enclosure
        serial=CK200051900278
        arrayname=EMC_CLARiiON
        arraytype=CLR-A/PF
        iopolicy=minimumq
        partitionsize=512
        recoveryoption=nothrottle
        recoveryoption=timebound iotimeout=300
        redundancy=0
        dmp_lun_retry_timeout=30
```

```
Enclosure
        serial=50001FE1500A8F00
        arrayname=EVA4K6K
        arraytype=ALUA
        iopolicy=adaptive
        partitionsize=512
        use_all_paths=no
        recoveryoption=nothrottle
        recoveryoption=timebound iotimeout=300
        redundancy=0
        dmp_lun_retry_timeout=30
Enclosure
        serial=50001FE1500BB690
        arrayname=EVA4K6K
        arraytype=ALUA
        iopolicy=adaptive
        partitionsize=512
        use_all_paths=no
        recoveryoption=nothrottle
        recoveryoption=timebound iotimeout=300
        redundancy=0
        dmp_lun_retry_timeout=30
Enclosure
        serial=DISKS
        arrayname=Disk
        arraytype=Disk
        iopolicy=minimumq
        partitionsize=512
        recoveryoption=nothrottle
        recoveryoption=timebound iotimeout=300
        redundancy=0
        dmp_lun_retry_timeout=30
```

Tuning a DMP host with a configuration attribute template

You can use a template file to upload a series of changes to the DMP configuration to the same host or to another similar host.

Veritas recommends that you load the DMP template to a host that is similar to the host that was the source of the tunable values.

To configure DMP on a host with a template

1. Dump the contents of the current host configuration to a file.

   ```
   # vxdmpadm config dump file=filename
   ```

2. Edit the file to make any required changes to the tunable parameters in the template.

   The target host may include non-shared arrays or host-specific arrays. To avoid updating these with settings from the array name or array type, define an enclosure section for each of those arrays in the template. When you load the template file to the target host, the enclosure section determines the settings. Otherwise, DMP applies the settings from the respective array name or array type sections.

3. Validate the values of the DMP tunable parameters.

   ```
   # vxdmpadm config check file=filename
   ```

   DMP displays no output if the configuration check is successful. If the file contains errors, DMP displays the errors. Make any required corrections until the configuration file is valid. For example, you may see errors such as the following:

   ```
   VxVM vxdmpadm ERROR V-5-1-0 Template file 'error.file' contains
   following errors:

   Line No: 22  'dmp_daemon_count' can not be set to 0 or less
   Line No: 44  Specified value for 'dmp_health_time' contains
   non-digits
   Line No: 64  Specified value for 'dmp_path_age' is beyond
   the limit of its value
   Line No: 76  'dmp_probe_idle_lun' can be set to either on or off
   Line No: 281 Unknown arraytype
   ```

4. Load the file to the target host.

   ```
   # vxdmpadm config load file=filename
   ```

   During the loading process, DMP validates each section of the template. DMP loads all valid sections. DMP does not load any section that contains errors.

Managing the DMP configuration files

You can display the name of the template file most recently loaded to the host. The information includes the date and time when DMP loaded the template file.

To display the name of the template file that the host currently uses

-
  ```
  # vxdmpadm config show

  TEMPLATE_FILE      DATE              TIME
  ==============================================
  /tmp/myconfig      Feb 09, 2011      11:28:59
  ```

Resetting the DMP tunable parameters and attributes to the default values

DMP maintains the default values for the DMP tunable parameters and attributes. At any time, you can restore the default values to the host. Any changes that you applied to the host with template files are discarded.

To reset the DMP tunables to the default values

- Use the following command:

  ```
  # vxdmpadm config reset
  ```

DMP tunable parameters and attributes that are supported for templates

DMP supports tuning the following tunable parameters and attributes with a configuration template.

DMP tunable parameters	See “DMP tunable parameters” on page 796.
DMP attributes defined for an enclosure, array name, or array type.	■ iopolicy ■ partitionsize ■ use_all_paths ■ recoveryoption attributes (retrycount or iotimeout) ■ redundancy ■ dmp_lun_retry_timeout
Naming scheme attributes:	■ naming scheme ■ persistence ■ lowercase ■ use_avid

The following tunable parameters are NOT supported with templates:

- OS tunables

- TPD mode
- Failover attributes of enclosures (failovermode)

Tunable parameters for VxVM

Veritas Volume Manager (VxVM) has several parameters that you can use to tune the environment. The VxVM tunable parameters comprise several components.

The VxVM components for tunable parameters are as follows:

- basevm
  Parameters to tune the core functionality of VxVM.
  See "Tunable parameters for core VxVM" on page 811.
- cvm
  Parameters to tune Cluster Volume Manager (CVM).
  See "Tunable parameters for CVM" on page 823.
- fmr
  Parameters to tune the FlashSnap functionality (FMR).
  See "Tunable parameters for FlashSnap (FMR)" on page 818.
- vvr
  Parameters to tune Veritas Volume Replicator (VVR).
  See "Tunable parameters for VVR" on page 824.

Tunable parameters for core VxVM

Table B-2 lists the kernel tunable parameters for VxVM.

You can tune the parameters using the `vxtune` command or the operating system method, unless otherwise noted.

Table B-2 Kernel tunable parameters for core VxVM

Parameter	Description
`vol_checkpt_default`	The interval at which utilities performing recoveries or resynchronization operations load the current offset into the kernel as a checkpoint. A system failure during such operations does not require a full recovery, but can continue from the last reached checkpoint. The default value is 20480 sectors (10MB). Increasing this size reduces the overhead of checkpoints on recovery operations at the expense of additional recovery following a system failure during a recovery.
`vol_default_iodelay`	The count in clock ticks for which utilities pause if they have been directed to reduce the frequency of issuing I/O requests, but have not been given a specific delay time. This tunable is used by utilities performing operations such as resynchronizing mirrors or rebuilding RAID-5 columns. The default value is 50 ticks. Increasing this value results in slower recovery operations and consequently lower system impact while recoveries are being performed.
`vol_max_adminio_poolsz`	The maximum size of the memory pool that is used for administrative I/O operations. VxVM uses this pool when throttling administrative I/O. The default value is 128MB. The maximum size must not be greater than the value of the `voliomem_maxpool_sz` parameter.
`vol_max_vol`	This parameter cannot be tuned with the `vxtune` command. The maximum number of volumes that can be created on the system. The minimum permitted value is 1. The maximum permitted value is the maximum number of minor numbers representable on the system. The default value is 65534.

Table B-2 Kernel tunable parameters for core VxVM *(continued)*

Parameter	Description
`vol_maxio`	The maximum size of logical I/O operations that can be performed without breaking up the request. I/O requests to VxVM that are larger than this value are broken up and performed synchronously. Physical I/O requests are broken up based on the capabilities of the disk device and are unaffected by changes to this maximum logical request limit. The default value is 2048 sectors (1 MB). The value of `voliomem_maxpool_sz` must be at least 10 times greater than the value of `vol_maxio`. If DRL sequential logging is configured, the value of `voldrl_min_regionsz` must be set to at least half the value of `vol_maxio`. The maximum limit for `vol_maxio` is 20% of the smaller of physical memory or kernel virtual memory. It is inadvisable to go over this limit.
`vol_maxioctl`	The maximum size of data that can be passed into VxVM via an `ioctl` call. Increasing this limit allows larger operations to be performed. Decreasing the limit is not generally recommended, because some utilities depend upon performing operations of a certain size and can fail unexpectedly if they issue oversized `ioctl` requests. The default value is 32768 bytes (32KB).
`vol_maxparallelio`	The number of I/O operations that the `vxconfigd` daemon is permitted to request from the kernel in a single `VOL_VOLDIO_READ` per `VOL_VOLDIO_WRITE` `ioctl` call. The default value is 256. This value should not be changed.

Table B-2 Kernel tunable parameters for core VxVM *(continued)*

Parameter	Description
`vol_maxspecialio`	The maximum size of an I/O request that can be issued by an `ioctl` call. Although the `ioctl` request itself can be small, it can request that a large I/O request be performed. This tunable limits the size of these I/O requests. If necessary, a request that exceeds this value can be failed, or the request can be broken up and performed synchronously. The default value is 2048 sectors. Raising this limit can cause difficulties if the size of an I/O request causes the process to take more memory or kernel virtual mapping space than exists and thus deadlock. The maximum limit for this tunable is 20% of the smaller of physical memory or kernel virtual memory. It is inadvisable to go over this limit, because deadlock is likely to occur. If stripes are larger than the value of this tunable, full stripe I/O requests are broken up, which prevents full-stripe read/writes. This throttles the volume I/O throughput for sequential I/O or larger I/O requests. This tunable limits the size of an I/O request at a higher level in VxVM than the level of an individual disk. For example, for an 8 by 64KB stripe, a value of 256KB only allows I/O requests that use half the disks in the stripe; thus, it cuts potential throughput in half. If you have more columns or you have used a larger interleave factor, then your relative performance is worse. This tunable must be set, as a minimum, to the size of your largest stripe (RAID-0 or RAID-5).
`vol_stats_enable`	Enables or disables the I/O stat collection for Veritas Volume manager objects. The default value is 1, since this functionality is enabled by default.
`vol_subdisk_num`	The maximum number of subdisks that can be attached to a single plex. The default value of this tunable is 4096.

Table B-2 Kernel tunable parameters for core VxVM *(continued)*

Parameter	Description
`voliomem_chunk_size`	The granularity of memory chunks used by VxVM when allocating or releasing system memory. A larger granularity reduces CPU overhead by allowing VxVM to retain hold of a larger amount of memory. The value of this tunable parameter depends on the page size of the system. You cannot specify a value larger than the default value. If you change the value, VxVM aligns the values to the page size when the system reboots. The default value is 32 KB for 512 Byte page size.
`voliomem_maxpool_sz`	The maximum memory requested from the system by VxVM for internal purposes. This tunable has a direct impact on the performance of VxVM as it prevents one I/O operation from using all the memory in the system. VxVM allocates two pools that can grow up to this size, one for RAID-5 and one for mirrored volumes. Additional pools are allocated if instant (Copy On Write) snapshots are present. A write request to a RAID-5 volume that is greater than one fourth of the pool size is broken up and performed in chunks of one tenth of the pool size. A write request to a mirrored volume that is greater than the pool size is broken up and performed in chunks of the pool size. The default value is 134217728 (128MB). The value of `voliomem_maxpool_sz` must be greater than the value of `volraid_minpool_size`. The value of `voliomem_maxpool_sz` must be at least 10 times greater than the value of `vol_maxio`.

Table B-2 Kernel tunable parameters for core VxVM *(continued)*

Parameter	Description
`voliot_errbuf_dflt`	The default size of the buffer maintained for error tracing events. This buffer is allocated at driver load time and is not adjustable for size while VxVM is running. The default value is 16384 bytes (16KB). Increasing this buffer can provide storage for more error events at the expense of system memory. Decreasing the size of the buffer can result in an error not being detected via the tracing device. Applications that depend on error tracing to perform some responsive action are dependent on this buffer.
`voliot_iobuf_default`	The default size for the creation of a tracing buffer in the absence of any other specification of desired kernel buffer size as part of the trace `ioctl`. The default value is 8192 bytes (8 KB). If trace data is often being lost due to this buffer size being too small, then this value can be increased.
`voliot_iobuf_limit`	The upper limit to the size of memory that can be used for storing tracing buffers in the kernel. Tracing buffers are used by the VxVM kernel to store the tracing event records. As trace buffers are requested to be stored in the kernel, the memory for them is drawn from this pool. Increasing this size can allow additional tracing to be performed at the expense of system memory usage. Setting this value to a size greater than can readily be accommodated on the system is inadvisable. The default value is 131072 bytes (128 KB).
`voliot_iobuf_max`	The maximum buffer size that can be used for a single trace buffer. Requests of a buffer larger than this size are silently truncated to this size. A request for a maximal buffer size from the tracing interface results (subject to limits of usage) in a buffer of this size. The default value is 65536 bytes (64 KB). Increasing this buffer can provide for larger traces to be taken without loss for very heavily used volumes. Do not increase this value above the value for the `voliot_iobuf_limit` tunable value.

Table B-2 Kernel tunable parameters for core VxVM *(continued)*

Parameter	Description
`voliot_max_open`	The maximum number of tracing channels that can be open simultaneously. Tracing channels are clone entry points into the tracing device driver. Each `vxtrace` process running on a system consumes a single trace channel. The default number of channels is 32. The allocation of each channel takes up approximately 20 bytes even when the channel is not in use.
`volraid_minpool_size`	This parameter cannot be tuned with the `vxtune` command. The initial amount of memory that is requested from the system by VxVM for RAID-5 operations. The maximum size of this memory pool is limited by the value of `voliomem_maxpool_sz`. The default value is 8192 sectors (4MB).
`volraid_rsrtransmax`	The maximum number of transient reconstruct operations that can be performed in parallel for RAID-5. A transient reconstruct operation is one that occurs on a non-degraded RAID-5 volume that has not been predicted. Limiting the number of these operations that can occur simultaneously removes the possibility of flooding the system with many reconstruct operations, and so reduces the risk of causing memory starvation. The default value is 1. Increasing this size improves the initial performance on the system when a failure first occurs and before a detach of a failing object is performed, but can lead to memory starvation.
`autostartvolumes`	Turns on or off the automatic volume recovery. When set to on, VxVM automatically recovers and starts disabled volumes when you import, join, move, or split a disk group. When set to off, turns off this behavior. The default value is on.
`fssmartmovethreshold`	The threshold for an individual file system, in percentage full. After this threshold is reached, the SmartMove feature is not used. The default value is 100.

Table B-2 Kernel tunable parameters for core VxVM *(continued)*

Parameter	Description
`reclaim_on_delete_start_time`	The time of day when the reclamation begins on a thin LUN, after a volume using that LUN is deleted. Specified in 24 hour time (HH:MM). The default value is 22:10.
`reclaim_on_delete_wait_period`	The number of days to wait before starting to reclaim space on a thin LUN, after a volume using that LUN is deleted. Specified as an integer from −1 to 366, where −1 specifies immediately and 366 specifies never. The default value is 1.
`usefssmartmove`	The state of the SmartMove feature. Valid values are: ■ thinonly − use for Thin disks only. ■ all − use for all disks. ■ none − turn off the SmartMove feature. The default value is all.

Tunable parameters for FlashSnap (FMR)

Table B-3 lists the kernel tunable parameters for FlashSnap. The `vxtune` command classifies these parameters under the FMR component.

You can tune the parameters using the `vxtune` command or the operating system method, unless otherwise noted.

Table B-3 Kernel tunable parameters for FlashSnap (FMR)

Parameter	Description
`vol_fmr_logsz`	The maximum size in kilobytes of the bitmap that Non-Persistent FastResync uses to track changed blocks in a volume. The number of blocks in a volume that are mapped to each bit in the bitmap depends on the size of the volume, and this value changes if the size of the volume is changed. For example, if the volume size is 1 gigabyte and the system block size is 512 bytes, a value for this tunable of 4 yields a map that contains 16,384 bits, each bit representing one region of 128 blocks. The larger the bitmap size, the fewer the number of blocks that are mapped to each bit. This can reduce the amount of reading and writing required on resynchronization, at the expense of requiring more non-pageable kernel memory for the bitmap. Additionally, on clustered systems, a larger bitmap size increases the latency in I/O performance, and it also increases the load on the private network between the cluster members. This is because every other member of the cluster must be informed each time a bit in the map is marked. Since the region size must be the same on all nodes in a cluster for a shared volume, the value of this tunable on the master node overrides the tunable values on the slave nodes, if these values are different. Because the value of a shared volume can change, the value of this tunable is retained for the life of the volume. In configurations which have thousands of mirrors with attached snapshot plexes, the total memory overhead can represent a significantly higher overhead in memory consumption than is usual for VxVM. The default value is 4 KB. The maximum and minimum permitted values are 1 KB and 8 KB. Specify a value to `vxtune` in kilobytes. **Note:** The value of this tunable does not have any effect on Persistent FastResync.

Table B-3 Kernel tunable parameters for FlashSnap (FMR) *(continued)*

Parameter	Description
`voldrl_dirty_regions`	This parameter applies to enhanced DCO layout (version 30) only. Represents the number of dirty regions to cache before another write to the same region causes a DRL update. A smaller number results in more frequent updates to the DRL, which decreases performance. A larger number results in better I/O performance, but requires that the DRL uses more memory. The default value is 1024.
`voldrl_max_drtregs`	The maximum number of dirty regions that can exist on the system for non-sequential DRL on volumes. A larger value may result in improved system performance at the expense of recovery time. This tunable can be used to regulate the worse-case recovery time for the system following a failure. The default value is 2048.
`voldrl_max_seq_dirty`	The maximum number of dirty regions allowed for sequential DRL. This is useful for volumes that are usually written to sequentially, such as database logs. Limiting the number of dirty regions allows for faster recovery if a crash occurs. The default value is 32.
`voldrl_min_regionsz`	The minimum number of sectors for a dirty region logging (DRL) volume region. With DRL, VxVM logically divides a volume into a set of consecutive regions. Larger region sizes tend to cause the cache hit-ratio for regions to improve. This improves the write performance, but it also prolongs the recovery time. The default value is 512 sectors. If DRL sequential logging is configured, the value of `voldrl_min_regionsz` must be set to at least half the value of `vol_maxio`. Specify the value in sectors.

Table B-3 Kernel tunable parameters for FlashSnap (FMR) *(continued)*

Parameter	Description
`voldrl_volumemax_drtregs`	Maximum per-volume limit on dirty regions for a mirrored volume using traditional DRL. For heavily-used volumes, increase the value of this parameter to improve performance. The default value is 256.
`voldrl_volumemax_drtregs_20`	Maximum per-volume limit on dirty regions for a mirrored volume using version 20 DCO. For heavily-used volumes, increase the value of this parameter to improve performance. The default value is 1024.

Table B-3 Kernel tunable parameters for FlashSnap (FMR) *(continued)*

Parameter	Description
`volpagemod_max_memsz`	The amount of memory that is allocated for caching FastResync and cache object metadata. The memory allocated for this cache is exclusively dedicated and is not available for other processes or applications. The default value is 6144KB (6MB). If cache objects or volumes that are prepared for instant snapshot operations are present on the system, setting the value below 512KB fails. If you do not use the FastResync or DRL features that are implemented using a version 20 DCO volume, you can set the value to 0. If you subsequently decide to enable these features, change the value to an appropriate one. Specify the value in kilobytes. The new value is page-aligned automatically; however the actual value specified is made persistent. Determine the value based on the region size and the number of volumes for which instant snapshots are taken. The paging module size must be at least twice the size required for the largest size volume, as calculated with the following formula: *size_in_KB* = 6 * (*total_volume_size_in_GB*) * (64/*region_size_in_KB*) For example, a single 1TB volume requires around 6MB of paging memory if the region size is 64KB. The minimum value for the tunable parameter is at least twice that, or 12 MB. If there are multiple volumes, all volumes share the same paging module. The maximum requirement is calculated by multiplying the above formula by the number of volumes. However, a more reasonable value depends on the average load to each volume. For example, if only 20% of the data on each volume is updated, the paging module size can be reduced proportionally without compromising the performance. The minimum requirement for the largest volume still must be met. For example, if there are 10 volumes of 1TB each, the initial calculation is 60MB of paging memory. If only 20% of the data is updated, calculate the revised value as 12MB.

Tunable parameters for CVM

Table B-4 lists the kernel tunable parameters for CVM. You can tune the parameters using the `vxtune` command or the operating system method, unless otherwise noted.

Table B-4 Kernel tunable parameters for CVM

`autoreminor`	Turns on or off the automatic reminor functionality. A disk group cannot be imported if the device minor numbers of the disk group or its objects conflict with those of an existing disk group. When autoreminor is on, VxVM automatically assigns new minor numbers to a disk group if VxVM detects a conflict during an import. The disk group is then imported. The default value is on. In some scenarios such as with NFS file systems, assigning new minor numbers may result in issues. In this case, set the tunable parameter to off'. When the autoreminor parameter is set to off', attempting to import a disk group with conflicting minor numbers will fail, even when you specify the force (-f) option. You must manually reminor the disk group before you can import the disk group.
`same_key_for_alldgs`	By default, CVM generates a unique fence key for each of the shared disk groups in the cluster. Some storage arrays have an upper limit on the number of unique keys that can be registered. When persistent reservation is used, CVM might reach the limit. When this tunable is set to on, CVM generates the same key for shared disk groups that it creates or imports. Disk groups that are already imported when the tunable is set must be deported and re-imported before the changed value of the tunable takes effect. The default value is off.
`sharedminorstart`	The starting number in the range used to assign device minor numbers in shared (CVM) disk groups. The default value is 33000.

Table B-4 Kernel tunable parameters for CVM *(continued)*

storage_connectivity	A clusterwide tunable parameter that indicates the CVM tolerance to storage disconnectivity. When the value is asymmetric (the default), CVM enables a node to join the cluster as long as the node has access to all of the disks through another node. Similarly, as long as at least one node can access the disks in a shared disk group, CVM can import the shared disk group. When the value is resilient, CVM requires that a node must have access to all of the disks in a shared disk group before the node can join the CVM cluster. The connectivity is also required before a shared disk group can be imported. With either setting, CVM handles access to the disk group configuration as long as at least one node has access to the disk group configuration.

Tunable parameters for VVR

Table B-5 lists the tunable parameters for VVR.

You can tune the parameters using the vxtune command or the operating system method, unless otherwise noted.

Table B-5 VVR Tunables

Tunable Name	Description
vol_cmpres_enabled	A clusterwide tunable parameter that enables or disables compression globally. The default value is 0, since compression is disabled by default.
vol_cmpres_threads	A per-system tunable that lets you set the number of compression threads on the Primary host or the number of decompression threads on the Secondary host between 1 and 64. The default value is 10. You can tune this setting dependent on your CPU usage.

Table B-5 VVR Tunables *(continued)*

Tunable Name	Description
`vol_dcm_replay_size`	This tunable cannot be changed using the `vxtune` command. The size of the DCM replay blocks. The default value is 256KB.
`vol_max_nmpool_sz`	The amount of buffer space available for requests coming in to the Secondary over the network. The default value is 128MB.
`vol_max_rdback_sz`	The amount of buffer space available for readbacks. The default value is 128MB.
`vol_max_wrspool_sz`	The write ship buffer space, which is the amount of buffer space that can be allocated on the logowner to receive writes sent by the non-logowner. The default value is 64MB.
`vol_min_lowmem_sz`	The minimum buffer space. VVR frees the write if the amount of buffer space available is below this threshold. The default value is 32MB. This value is auto-tunable. The value that you specify is used as an initial value and could change depending on the application write behavior.
`vol_nm_hb_timeout`	The heartbeat timeout value. The default value is 10 seconds.
`vol_rvio_maxpool_sz`	The amount of buffer space that can be allocated within the operating system to handle incoming writes. The default value is 128MB.
`vol_vvr_use_nat`	This tunable cannot be changed using the `vxtune` command. This tunable parameter directs VVR to use the translated address from the received message so that VVR can communicate over a NAT-based firewall. Set this tunable to 1 only if there is a NAT-based firewall in the configuration.

Tunable parameters for hot-relocation in FSS environments

Table B-6 lists the tunable parameters for hot-relocation in FSS environments. You can tune the parameters using the `vxtune` command.

Table B-6 Tunable parameters for hot-relocation in FSS environments

Tunable name	Description
`storage_reloc_timeout`	A cluster-wide tunable that specifies the the amount of time that VxVM waits for the storage to come online before initiating hot-relocation in FSS environments. The default value is 30 minutes.
`node_reloc_timeout`	A cluster-wide tunable that specifies the the amount of time that VxVM waits for the node to come online before initiating hot-relocation in FSS environments. The default value is 120 minutes.

Points to note when changing the values of the VVR tunables

Note the following points when changing the values of the tunables:

- When decreasing the value of the `vol_rvio_maxpool_sz` tunable, all the RVGs on the host must be stopped.
- When decreasing the size of the tunables `vol_max_rdback_sz` and `vol_max_nmpool_sz` pause the RLINKs.

  Note: `vol_max_wrspool_sz` also pauses the RLINKs.

- The `vol_min_lowmem_sz` tunable is auto-tunable; depending on the incoming writes VVR increases or decreases the tunable value.
  Auto-tuning is only supported for the tunable `vol_min_lowmem_sz`.

In a shared disk group environment, you may choose to set only those tunables that are required on each host. However, we recommend that you configure the tunables appropriately even if the tunables are currently not being used. This is because if the logowner changes, then tunables on the new logowner will be used. The following list of tunables are required to be set only on the logowner and not the other hosts:

- `vol_max_rdback_sz`
- `vol_max_nmpool_sz`
- `vol_max_wrspool_sz`
- `vol_dcm_replay_size`
- `vol_nm_hb_timeout`

- vol_vvr_use_nat

The tunable changes that are done using the vxtune command affect only the tunable values on the host on which it is run. Therefore, in a shared disk group environment, you must run the command separately on each host for which you want to change the tunable values.

Methods to change Veritas Volume Manager tunable parameters

Veritas Volume Manager (VxVM) provides a variety of parameters that you can use to tune your configuration.

See "Tunable parameters for VxVM" on page 811.

Change the VxVM tunable parameters with one of the following methods:

Use the vxtune command to display or modify the values of the VxVM tunable parameters.	See "Changing the values of the Veritas Volume Manager tunable parameters using the vxtune command line" on page 827.
Use the template method of the vxtune command.	See "Changing the value of the Veritas Volume Manager tunable parameters using templates" on page 830.

Changing the values of the Veritas Volume Manager tunable parameters using the vxtune command line

Use the vxtune command to display or change the values of the VxVM tunable parameters. The changes are persistent so that the value is retained after subsequent reboots. Before setting a new value, VxVM validates the value to ensure that it is within the permissible range for the tunable. If the value is valid, VxVM updates the tunable. Some tunables require a reboot before the changed value takes effect. VxVM prompts you to reboot the system, if required.

By default, the vxtune command only affects the tunable values on the host on which it is run, except for clusterwide tunable parameters. The clusterwide attribute indicates that the vxtune command sets the value for that tunable on all nodes in the cluster. If the tunable parameter is not clusterwide, you can use the -C option to change the tunable values on all nodes of the cluster. If the -C option is used on a standalone system, the operation fails.

VxVM stores the tunable values in the /etc/vx/vxtunables file.

Caution: The recommended method to change the tunable values is with the `vxtune` command. Do not edit the tunable values directly in the `vxtunables` file.

For most tunables, specify the value of the tunable with a suffix to indicate the units: K, M, or G. If no unit is specified, vxtune assumes the value is bytes.

Note: The default unit for entering a value may differ from the default display unit.

To change the values of the VxVM tunable parameters

1 Find the name and current value of the tunable you want to change. Use the `-l` option to display a description.

```
# vxtune -l
```

The following example shows a truncated output, that shows the format.

```
Tunable             Current Value Default Value Reboot Clusterwide  Description
------------------- ------------- ------------- ------ -----------  -----------
vol_checkpt_default         20480         20480      Y           N  Size of VxVM
                                                                    checkpoints (sectors)
vol_cmpres_enabled              0             0      N           Y  Allow enabling
                                                                    compression for VERITAS
                                                                    Volume Replicator
vol_cmpres_threads             10            10      N           N  Maximum number of
                                                                    compression threads
                                                                    for VERITAS
                                                                    Volume Replicator
vol_default_iodelay            50            50      Y           N  Time to pause between
                                                                    I/O requests from VxVM
                                                                    utilities (10ms units)
vol_fmr_logsz                   4             4      Y           N  Maximum size of bitmap
                                                                    Fast Mirror Resync
                                                                    uses to track changed
                                                                    blocks (KBytes)
vol_max_adminio_poolsz   67108864      67108864      Y           N  Maximum amount of
                                                                    memory used by
                                                                    VxVM admin IO's (bytes)
.
.
.
```

The output displays the default value and the current value. The Reboot field indicates whether or not a reboot is required before the tunable value takes effect. The Clusterwide field indicates whether `vxtune` applies the value to all nodes in the cluster by default.

See the `vxtune`(1M) manual page.

2. Set the new value for a specific tunable. Specify the value with a suffix to indicate the units: K, M, or G. If not unit is specified, the `vxtune` command uses the default unit for the tunable parameter. For most tunables, the default value is bytes. The description in the `vxtune` output displays the default units for each tunable.

   ```
   # vxtune [-C] tunable_name tunable_value
   ```

   For example, to change the value of `vol_cmpres_enabled` to 1, use the following command:

   ```
   # vxtune vol_cmpres_enabled 1
   ```

   If the specified tunable parameter is not clusterwide, use the `-C` option to set its value for all nodes in the cluster.

3. Verify the new value.

   ```
   # vxtune tunable_name
   ```

   For example, to view the changed value for `vol_cmpres_enabled`, use the following command:

   ```
   # vxtune vol_cmpres_enabled
   Tunable              Current Value Default Value Reboot
   -------------------- ------------- ------------- ------
   vol_cmpres_enabled   1             0             N
   ```

   The `vxtune` command changed the value on all nodes in the cluster, because the `vol_cmpres_enabled` tunable parameter is `clusterwide`.

Changing the value of the Veritas Volume Manager tunable parameters using templates

VxVM provides a template method to change the tunable parameters. With this method, you export the tunable parameters to a file, modify the file, then import the file. The tunable template must be strictly of the format that export provides. In case of discrepencies observe that particular value will be discarded.

To change the values of the VxVM tunable parameters using templates

1. Export the tunable parameters and their values to a tunable template file. You can export all of the tunable parameters or specify a component.

   ```
   # vxtune -o export file=file_name [component]
   ```

   For example:

   ```
   # vxtune -o export file=vxvm-tunables
   ```

   ```
   # vxtune -o export file=vvr-tunables vvr
   ```

2. Modify the template as required. You must retain the file format that the export operation provides.

3. Import the tunable template file to the sytem. The import operation only applies valid values. If a value is not valid for a specific parameter, that particular value is discarded.

   ```
   # vxtune -o import file=file_name
   ```

   For example:

   ```
   # vxtune -o import file=vxvm-tunables
   ```

Command reference

This appendix includes the following topics:

Command completion for Veritas commands

Storage Foundation supports command completion for Veritas Volume Manager (VxVM) commands and Dynamic Multi-Pathing (DMP) commands.

In this release, command completion is supported only on the bash shell. The shell must be bash version 2.4 or later.

To use this feature, press **Tab** while entering a supported VxVM or DMP command. The command is completed as far as possible. When there is a choice, the command completion displays the next valid options for the command. Enter one of the displayed values. A value in brackets indicates a user-specified value.

Note: Platform-specific options are not supported with command completion.

By default, you can use the command completion feature by invoking the bash shell on every log in. If you want to permanently enable the command completion, use the following command:

```
# vxdctl cmdcompletion enable
```

The enable command completion creates the `.bash_profile` file, if it is not present.

To permanently disable the command completion, use the following command:

```
# vxdctl cmdcompletion disable
```

See the `vxdctl`(1M) manual page.

The following commands support command completion:

- vxassist
- vxcache
- vxcdsconvert
- vxclustadm
- vxconfigd
- vxdctl
- vxddladm
- vxdg
- vxdisk
- vxdiskadm
- vxdisksetup
- vxdiskunsetup
- vxdmpadm
- vxedit
- vxinstall
- vxplex
- vxprint
- vxreattach
- vxrecover
- vxresize
- vxsd
- vxsnap
- vxstat

- vxtask
- vxtrace
- vxtune
- vxvol
- vxvset

Veritas Volume Manager command reference

Most Veritas Volume Manager (VxVM) commands (excepting daemons, library commands and supporting scripts) are linked to the `/usr/sbin` directory from the `/opt/VRTS/bin` directory. It is recommended that you add the following directories to your PATH environment variable:

- If you are using the Bourne or Korn shell (`sh` or `ksh`), use the commands:

```
$ PATH=$PATH:/usr/sbin:/opt/VRTS/bin:/opt/VRTSvxfs/sbin:\
  /opt/VRTSdbed/bin:/opt/VRTSob/bin
$ MANPATH=/usr/share/man:/opt/VRTS/man:$MANPATH
$ export PATH MANPATH
```

- If you are using a C shell (`csh` or `tcsh`), use the commands:

```
% set path = ( $path /usr/sbin /opt/VRTSvxfs/sbin \
  /opt/VRTSdbed/bin /opt/VRTSob/bin /opt/VRTS/bin )
% setenv MANPATH /usr/share/man:/opt/VRTS/man:$MANPATH
```

VxVM library commands and supporting scripts are located under the `/usr/lib/vxvm` directory hierarchy. You can include these directories in your path if you need to use them on a regular basis.

For detailed information about an individual command, refer to the appropriate manual page in the 1M section.

See “Veritas Volume Manager manual pages” on page 855.

Commands and scripts that are provided to support other commands and scripts, and which are not intended for general use, are not located in `/opt/VRTS/bin` and do not have manual pages.

Commonly-used commands are summarized in the following tables:

- Table C-1 lists commands for obtaining information about objects in VxVM.
- Table C-2 lists commands for administering disks.

- Table C-3 lists commands for creating and administering disk groups.
- Table C-4 lists commands for creating and administering subdisks.
- Table C-5 lists commands for creating and administering plexes.
- Table C-6 lists commands for creating volumes.
- Table C-7 lists commands for administering volumes.
- Table C-8 lists commands for monitoring and controlling tasks in VxVM.

Table C-1 Obtaining information about objects in VxVM

Command	Description
`vxdctl license [init]`	List licensed features of VxVM. The init parameter is required when a license has been added or removed from the host for the new license to take effect.
`vxdisk [-g` *`diskgroup`*`] list [`*`diskname`*`]`	Lists disks under control of VxVM. See "Displaying disk information" on page 282. Example: `# ` **`vxdisk -g mydg list`**
`vxdg list [`*`diskgroup`*`]`	Lists information about disk groups. See "Displaying disk group information" on page 679. Example: `# ` **`vxdg list mydg`**
`vxdg -s list`	Lists information about shared disk groups. Example: `# ` **`vxdg -s list`**
`vxdisk -o alldgs list`	Lists all diskgroups on the disks. The imported diskgroups are shown as standard, and additionally all other diskgroups are listed in single quotes.

Table C-1 Obtaining information about objects in VxVM *(continued)*

Command	Description
`vxdisk -o cluster list`	Provides a global view of all disks, local and shared, in the cluster.
`vxinfo [-g` *diskgroup*`] [`*volume* `...]`	Displays information about the accessibility and usability of volumes. See the *Veritas InfoScale Troubleshooting Guide*. Example: `#` **`vxinfo -g mydg myvol1 \`** **`myvol2`**

`vxprint -hrt [-g` *diskgroup*`] [`*object* `...]`	Prints single-line information about objects in VxVM.
	Example:
	`#` **`vxprint -g mydg myvol1 \`**
	`myvol2`

`vxlist`	Provides a consolidated view of the SF configuration, including information from Veritas Volume Manager (VxVM) and Veritas File System (VxFS). See `vxlist`(1m) manual page.
`vxprint -st [-g` *diskgroup*`] [`*subdisk* `...]`	Displays information about subdisks. Example: `#` **`vxprint -st -g mydg`**
`vxprint -pt [-g` *diskgroup*`] [`*plex* `...]`	Displays information about plexes. Example: `#` **`vxprint -pt -g mydg`**

Table C-2 Administering disks

Command	Description
vxdisk [-o full] reclaim {disk|enclosure|diskgroup}...	Performs storage reclamation on thin provision LUNs.
vxdiskadm	Administers disks in VxVM using a menu-based interface.
vxdiskadd [*devicename* ...]	Adds a disk specified by device name. See “Using vxdiskadd to put a disk under VxVM control” on page 302. Example: # **vxdiskadd sde**
vxedit [-g *diskgroup*] rename \\ *olddisk newdisk*	Renames a disk under control of VxVM. See “Renaming a disk” on page 306. Example: # **vxedit -g mydg rename \** **mydg03 mydg02**
vxedit [-g *diskgroup*] set \\ reserve=on|off *diskname*	Sets aside/does not set aside a disk from use in a disk group. Examples: # **vxedit -g mydg set \** **reserve=on mydg02** # **vxedit -g mydg set \** **reserve=off mydg02**

Table C-2 Administering disks *(continued)*

Command	Description

`vxedit [-g diskgroup] set \`
`nohotuse=on|off diskname`
	Does not/does allow free space on a disk to be used for hot-relocation.
	See "Excluding a disk from hot-relocation use" on page 607.
	See "Making a disk available for hot-relocation use" on page 608.
	Examples:
	`# vxedit -g mydg set \`
	`nohotuse=on mydg03`
	`# vxedit -g mydg set \`
	`nohotuse=off mydg03`

`vxedit [-g diskgroup] set \`
`spare=on|off diskname`
	Adds/removes a disk from the pool of hot-relocation spares.
	See "Marking a disk as a hot-relocation spare" on page 605.
	See "Removing a disk from use as a hot-relocation spare" on page 606.
	Examples:
	`# vxedit -g mydg set \`
	`spare=on mydg04`
	`# vxedit -g mydg set \`
	`spare=off mydg04`

`vxdisk online devicename`	Clears the offline state for a disk device. See `vxdisk`(1M) manual page. Example: `# vxdisk online sde`
`vxdisk offline devicename`	Takes a disk offline. Example: `# vxdisk offline sde`

Table C-2 Administering disks *(continued)*

Command	Description
`vxdg -g diskgroup adddisk diskname`	Adds a disk to a disk group. See "Growing the existing storage by adding a new LUN" on page 118. Example: `# vxdg -g mydg adddisk mydg02`
`vxdg -g diskgroup rmdisk diskname`	Removes a disk from its disk group. See "Removing a disk from a disk group" on page 682. Example: `# vxdg -g mydg rmdisk mydg02`
`vxdisksetup devicename`	Configures a disk for use with VxVM. See "Provisioning new storage" on page 117. Example: `# /etc/vx/bin/vxdisksetup -i enc1_3`
`vxdiskunsetup devicename`	Removes a disk from control of VxVM. See "Removing a disk from a disk group" on page 682. Example: `# vxdiskunsetup sdg`

Table C-3 Creating and administering disk groups

Command	Description
`vxdg [-s] init diskgroup \ [diskname=]devicename`	Creates a disk group using a pre-initialized disk. See "Creating a disk group" on page 681. Example: `# vxdg init mydg \ mydg01=sde`

Table C-3 Creating and administering disk groups *(continued)*

Command	Description
`vxdg -g diskgroup listssbinfo`	Reports conflicting configuration information. See "Handling conflicting configuration copies" on page 703. Example: `# vxdg -g mydg listssbinfo`
`vxdg [-n newname] deport diskgroup`	Deports a disk group and optionally renames it. See "Deporting a disk group" on page 683. Example: `# vxdg -n newdg deport mydg`
`vxdg [-n newname] import diskgroup`	Imports a disk group and optionally renames it. See "Importing a disk group" on page 685. Example: `# vxdg -n newdg import mydg`

`vxdg [-n newname] -s import diskgroup`	Imports a disk group as shared by a cluster, and optionally renames it.
	Example:
	`# vxdg -n newsdg -s import \`
	`mysdg`

Table C-3 Creating and administering disk groups *(continued)*

<table>
<tr><th>Command</th><th>Description</th></tr>
<tr><td>vxdg [-o expand] listmove <i>sourcedg</i> \
<i>targetdg object</i> ...</td><td>Lists the objects potentially affected by moving a disk group.
See “Listing objects potentially affected by a move” on page 648.
Example:
<b>vxdg -o expand listmove \</b>
<b>mydg newdg myvol1</b></td></tr>
<tr><td>vxdg [-o expand] move <i>sourcedg</i> \
<i>targetdg object</i> ...</td><td>Moves objects between disk groups.
See “Moving objects between disk groups” on page 650.
Example:
<b>vxdg -o expand move mydg \</b>
<b>newdg myvol1</b></td></tr>
<tr><td>vxdg [-o expand] split <i>sourcedg</i> \
<i>targetdg object</i> ...</td><td>Splits a disk group and moves the specified objects into the target disk group.
See “Splitting disk groups” on page 653.
Example:
<b>vxdg -o expand split mydg \</b>
<b>newdg myvol2 myvol3</b></td></tr>
<tr><td>vxdg join <i>sourcedg targetdg</i></td><td>Joins two disk groups.
See “Joining disk groups” on page 654.
Example:
<b>vxdg join newdg mydg</b></td></tr>
<tr><td>vxdg -g <i>diskgroup</i> set \
activation=ew|ro|sr|sw|off</td><td>Sets the activation mode of a shared disk group in a cluster.
Example:
<b>vxdg -g mysdg set \</b>
<b>activation=sw</b></td></tr>
</table>

Table C-3 Creating and administering disk groups *(continued)*

Command	Description
`vxrecover -g` *`diskgroup`* `-sb`	Starts all volumes in an imported disk group. See "Moving disk groups between systems" on page 687. Example: `# ` **`vxrecover -g mydg -sb`**
`vxdg destroy` *`diskgroup`*	Destroys a disk group and releases its disks. See "Destroying a disk group" on page 710. Example: `# ` **`vxdg destroy mydg`**

Table C-4 Creating and administering subdisks

Command	Description

`vxmake [-g` *`diskgroup`*`] sd` *`subdisk`* `\`
`diskname,offset,length`
	Creates a subdisk.
	Example:
	`# ` **`vxmake -g mydg sd \`**
	`mydg02-01 mydg02,0,8000`

`vxsd [-g` *`diskgroup`*`] assoc` *`plex`* `\`
`subdisk...`
	Associates subdisks with an existing plex.
	Example:
	`# ` **`vxsd -g mydg assoc home-1 \`**
	`mydg02-01 mydg02-00 \`
	`mydg02-01`

Table C-4 Creating and administering subdisks *(continued)*

Command	Description
vxsd [-g *diskgroup*] assoc *plex* \ *subdisk1:0 ... subdiskM:N-1*	Adds subdisks to the ends of the columns in a striped or RAID-5 volume. Example: # **vxsd -g mydg assoc \ vol01-01 mydg10-01:0 \ mydg11-01:1 mydg12-01:2**
vxsd [-g *diskgroup*] mv *oldsubdisk* \ *newsubdisk ...*	Replaces a subdisk. Example: # **vxsd -g mydg mv mydg01-01 \ mydg02-01**
vxsd [-g *diskgroup*] -s *size* split \ *subdisk sd1 sd2*	Splits a subdisk in two. Example: # **vxsd -g mydg -s 1000m \ split mydg03-02 mydg03-02 \ mydg03-03**
vxsd [-g diskgroup] join \ *sd1 sd2 ... subdisk*	Joins two or more subdisks. Example: # **vxsd -g mydg join \ mydg03-02 mydg03-03 \ mydg03-02**
vxassist [-g *diskgroup*] move \ *volume* \\!*olddisk newdisk*	Relocates subdisks in a volume between disks. Example: # **vxassist -g mydg move \ myvol \\!mydg02 mydg05** **Note:** The ! character is a special character in some shells. This example shows how to escape it in a bash shell.

Table C-4 Creating and administering subdisks *(continued)*

Command	Description
vxunreloc [-g *diskgroup*] *original_disk*	Relocates subdisks to their original disks. See “Moving relocated subdisks using vxunreloc” on page 609. Example: `#` **`vxunreloc -g mydg mydg01`**
vxsd [-g *diskgroup*] dis *subdisk*	Dissociates a subdisk from a plex. Example: `#` **`vxsd -g mydg dis mydg02-01`**
vxedit [-g *diskgroup*] rm *subdisk*	Removes a subdisk. Example: `#` **`vxedit -g mydg rm mydg02-01`**

vxsd [-g *diskgroup*] -o rm dis *subdisk*	Dissociates and removes a subdisk from a plex.
	Example:
	`#` **`vxsd -g mydg -o rm dis \`**
	`mydg02-01`

Table C-5 Creating and administering plexes

Command	Description

vxmake [-g *diskgroup*] plex *plex* \\
sd=*subdisk1*[,*subdisk2*,...]
	Creates a concatenated plex.
	Example:
	`#` **`vxmake -g mydg plex \`**
	`vol01-02 \`
	`sd=mydg02-01,mydg02-02`

Table C-5 Creating and administering plexes *(continued)*

Command	Description

`vxmake [-g diskgroup] plex plex \`
`layout=stripe|raid5 stwidth=W \`
`ncolumn=N \`
`sd=subdisk1[,subdisk2,...]`
	Creates a striped or RAID-5 plex.
	Example:
	`# vxmake -g mydg plex pl-01 \`
	`layout=stripe stwidth=32 \`
	`ncolumn=2 \`
	`sd=mydg01-01,mydg02-01`

`vxplex [-g diskgroup] att volume plex`	Attaches a plex to an existing volume.
	See "Reattaching a plex manually" on page 717.
	Example:
	`# vxplex -g mydg att vol01 \`
	`vol01-02`

`vxplex [-g diskgroup] det plex`	Detaches a plex. Example: `# vxplex -g mydg det vol01-02`
`vxmend [-g diskgroup] off plex`	Takes a plex offline for maintenance. Example: `# vxmend -g mydg off vol02-02`
`vxmend [-g diskgroup] on plex`	Re-enables a plex for use. See "Reattaching a plex manually" on page 717. Example: `# vxmend -g mydg on vol02-02`

`vxplex [-g diskgroup] mv oldplex \`
`newplex`
	Replaces a plex.
	Example:
	`# vxplex -g mydg mv \`
	`vol02-02 vol02-03`

Table C-5 Creating and administering plexes *(continued)*

Command	Description

`vxplex [-g diskgroup] cp volume \`
`newplex`
	Copies a volume onto a plex.
	Example:
	`# vxplex -g mydg cp vol02 \`
	`vol03-01`

`vxmend [-g diskgroup] fix clean plex`	Sets the state of a plex in an unstartable volume to CLEAN.
	See "Reattaching a plex manually" on page 717.
	Example:
	`# vxmend -g mydg fix clean \`
	`vol02-02`

`vxplex [-g diskgroup] -o rm dis plex`	Dissociates and removes a plex from a volume.
	Example:
	`# vxplex -g mydg -o rm dis \`
	`vol03-01`

Table C-6 Creating volumes

Command	Description

`vxassist [-g diskgroup] maxsize \`
`layout=layout [attributes]`
	Displays the maximum size of volume that can be created.
	Example:
	`# vxassist -g mydg maxsize \`
	`layout=raid5 nlog=2`

Table C-6 Creating volumes *(continued)*

Command	Description
vxassist -b [-g *diskgroup*] make \ *volume length* [layout=*layout*] \ [*attributes*]	Creates a volume. See “Creating a volume on specific disks” on page 151. Example: # **vxassist -b -g mydg make \ myvol 20g layout=concat \ mydg01 mydg02**
vxassist -b [-g *diskgroup*] make \ *volume length* layout=mirror \ [nmirror=*N*] [*attributes*]	Creates a mirrored volume. See “Creating a mirrored volume” on page 145. Example: # **vxassist -b -g mydg make \ mymvol 20g layout=mirror \ nmirror=2**
vxassist -b [-g *diskgroup*] make \ *volume length* layout=*layout* \ exclusive=on [*attributes*]	Creates a volume that may be opened exclusively by a single node in a cluster. Example: # **vxassist -b -g mysdg make \ mysmvol 20g layout=mirror \ exclusive=on**
vxassist -b [-g *diskgroup*] make \ *volume length* layout={stripe|raid5} \ [stripeunit=*W*] [ncol=*N*] \ [*attributes*]	Creates a striped or RAID-5 volume. See “Creating a striped volume” on page 147. See “Creating a RAID-5 volume” on page 149. Example: # **vxassist -b -g mydg make \ mysvol 20g layout=stripe \ stripeunit=32 ncol=4**

Table C-6 Creating volumes *(continued)*

Command	Description
vxassist -b [-g *diskgroup*] make \ *volume* length layout=mirror \ mirror=ctlr [*attributes*]	Creates a volume with mirrored data plexes on separate controllers. Example: # **vxassist -b -g mydg make \ mymcvol 20g layout=mirror \ mirror=ctlr**
vxassist -g *dg_name* -Q \ make *vol_name volume_size*	Creates a volume with its sector size exported as 4KB to VxFS. Example: # **vxassist -g dg1 -Q make vol1 10g** # **vxassist -g dg1 -Q make srl 1g**
vxmake -b [-g *diskgroup*] \ -U*usage_type* vol *volume* \ [len=*length*] plex=*plex*,...	Creates a volume from existing plexes. Example: # **vxmake -g mydg -Uraid5 \ vol r5vol \ plex=raidplex,raidlog1,\ raidlog2**
vxvol [-g *diskgroup*] start *volume*	Initializes and starts a volume for use. Example: # **vxvol -g mydg start r5vol**
vxvol [-g *diskgroup*] init zero \ *volume*	Initializes and zeros out a volume for use. Example: # **vxvol -g mydg init zero \ myvol**

Table C-7 Administering volumes

Command	Description

`vxassist [-g diskgroup] mirror \`
`volume [attributes]`
	Adds a mirror to a volume.
	See "Adding a mirror to a volume " on page 667.
	Example:
	`# vxassist -g mydg mirror \`
	`myvol mydg10`

`vxassist [-g diskgroup] remove \`
`mirror volume [attributes]`
	Removes a mirror from a volume.
	See "Removing a mirror " on page 670.
	Example:
	`# vxassist -g mydg remove \`
	`mirror myvol \!mydg11`
	Note: The ! character is a special character in some shells. This example shows how to escape it in a bash shell.

`vxassist [-g diskgroup] \`
`{growto|growby} volume length`
	Grows a volume to a specified size or by a specified amount.
	Example:
	`# vxassist -g mydg growby \`
	`myvol 10g`

`vxassist [-g diskgroup] \`
`{shrinkto|shrinkby} volume length`
	Shrinks a volume to a specified size or by a specified amount.
	Example:
	`# vxassist -g mydg shrinkto \`
	`myvol 20g`

Table C-7 Administering volumes *(continued)*

Command	Description
vxresize -b -F vxfs [-g *diskgroup*] \ *volume length diskname* ...	Resizes a volume and the underlying Veritas File System. Example: # **vxresize -b -F vxfs \** **-g mydg myvol 20g mydg10 \** **mydg11**
vxsnap [-g *diskgroup*] prepare *volume* [drl=on|sequential|off]	\Prepares a volume for instant snapshots and for DRL logging. See "Adding an instant snap DCO and DCO volume" on page 392. Example: # **vxsnap -g mydg prepare \** **myvol drl=on**
vxsnap [-g *diskgroup*] make \ source=*volume*\ /newvol=*snapvol*\ [/nmirror=*number*]	Takes a full-sized instant snapshot of a volume by breaking off plexes of the original volume. See "Creating instant snapshots" on page 390. Example: # **vxsnap -g mydg make \** **source=myvol/\** **newvol=mysnpvol/\** **nmirror=2**

Table C-7 Administering volumes *(continued)*

Command	Description
vxsnap [-g *diskgroup*] make \ source=*volume*/snapvol=*snapvol*	Takes a full-sized instant snapshot of a volume using a prepared empty volume. See "Creating a volume for use as a full-sized instant or linked break-off snapshot" on page 396. See "Creating instant snapshots" on page 390. Example: # **vxsnap -g mydg make \** **source=myvol/snapvol=snpvol**
vxmake [-g *diskgroup*] cache \ *cache_object* cachevolname=*volume* \ [regionsize=*size*]	Creates a cache object for use by space-optimized instant snapshots. See "Creating a shared cache object" on page 394. A cache volume must have already been created. After creating the cache object, enable the cache object with the vxcache start command. For example: # **vxassist -g mydg make \** **cvol 1g layout=mirror \** **init=active mydg16 mydg17** # **vxmake -g mydg cache cobj \** **cachevolname=cvol** # **vxcache -g mydg start cobj**
vxsnap [-g *diskgroup*] make \ source=*volume*/newvol=*snapvol*\ /cache=*cache_object*	Takes a space-optimized instant snapshot of a volume. See "Creating instant snapshots" on page 390. Example: # **vxsnap -g mydg make \** **source=myvol/\** **newvol=mysosvol/\** **cache=cobj**

Table C-7 Administering volumes *(continued)*

Command	Description

`vxsnap [-g diskgroup] refresh snapshot`	Refreshes a snapshot from its original volume.
	See "Refreshing an instant space-optimized snapshot" on page 413.
	Example:
	`# vxsnap -g mydg refresh \`
	`mysnpvol`

`vxsnap [-g diskgroup] dis snapshot`	Turns a snapshot into an independent volume. See "Dissociating an instant snapshot" on page 415. Example: `# vxsnap -g mydg dis mysnpvol`

`vxsnap [-g diskgroup] unprepare \`
`volume`
	Removes support for instant snapshots and DRL logging from a volume.
	Example:
	`# vxsnap -g mydg unprepare \`
	`myvol`

`vxassist [-g diskgroup] relayout \`
`volume [layout=layout] \`
`[relayout_options]`
	Performs online relayout of a volume.
	See "Performing online relayout" on page 660.
	Example:
	`# vxassist -g mydg relayout \`
	`vol2 layout=stripe`

Table C-7 Administering volumes *(continued)*

Command	Description

`vxassist [-g diskgroup] relayout \`
`volume layout=raid5 \`
`stripeunit=W \`
`ncol=N`
	Relays out a volume as a RAID-5 volume with stripe width *W* and *N* columns.
	See "Performing online relayout" on page 660.
	Example:
	`# vxassist -g mydg relayout \`
	`vol3 layout=raid5 \`
	`stripeunit=16 ncol=4`

`vxrelayout [-g diskgroup] -o bg \`
`reverse volume`
	Reverses the direction of a paused volume relayout.
	See "Volume sets" on page 91.
	Example:
	`# vxrelayout -g mydg -o bg \`
	`reverse vol3`

`vxassist [-g diskgroup] convert \`
`volume [layout=layout] \`
`[convert_options]`
	Converts between a layered volume and a non-layered volume layout.
	Example:
	`# vxassist -g mydg convert \`
	`vol3 layout=stripe-mirror`

`vxassist [-g diskgroup] remove \`
`volume volume`
	Removes a volume.
	See "Removing a volume" on page 719.
	Example:
	`# vxassist -g mydg remove \`
	`myvol`

Table C-8 Monitoring and controlling tasks

Command	Description

`command [-g diskgroup] -t tasktag \`
`[options] [arguments]`
	Specifies a task tag to a VxVM command.
	See "Specifying task tags" on page 657.
	Example:
	`# vxrecover -g mydg \`
	`-t mytask -b mydg05`

`vxtask [-h] [-g diskgroup] list`	Lists tasks running on a system. See "Using the vxtask command" on page 659. Example: `# vxtask -h -g mydg list`
`vxtask monitor task`	Monitors the progress of a task. See "Using the vxtask command" on page 659. Example: `# vxtask monitor mytask`
`vxtask pause task`	Suspends operation of a task. See "Using the vxtask command" on page 659. Example: `# vxtask pause mytask`
`vxtask -p [-g diskgroup] list`	Lists all paused tasks. See "Using the vxtask command" on page 659. Example: `# vxtask -p -g mydg list`

Table C-8 Monitoring and controlling tasks *(continued)*

Command	Description
`vxtask resume` *`task`*	Resumes a paused task. See "Using the vxtask command" on page 659. Example: `# `**`vxtask resume mytask`**
`vxtask abort` *`task`*	Cancels a task and attempts to reverse its effects. See "Using the vxtask command" on page 659. Example: `# `**`vxtask abort mytask`**

Veritas Volume Manager manual pages

Manual pages are organized into the following sections:

1M	Administrative commands.
4	File formats.

Section 1M — administrative commands

Table C-9 lists the manual pages in section 1M for commands that are used to administer Veritas Volume Manager.

Table C-9 Section 1M manual pages

Name	Description
`vxassist`	Create, relayout, convert, mirror, backup, grow, shrink, delete, and move volumes.
`vxcache`	Administer the cache object for space-optimized snapshots.

Table C-9 Section 1M manual pages *(continued)*

Name	Description
vxcached	Daemon that monitors and resizes cache volumes when required.
vxcdsconvert	Make disks and disk groups portable between systems.
vxclustadm	Start, stop, and reconfigure a cluster.
vxcmdlog	Administer command logging.
vxconfigbackup	Back up disk group configuration.
vxconfigbackupd	Disk group configuration backup daemon.
vxconfigd	Veritas Volume Manager configuration daemon
vxconfigrestore	Restore disk group configuration.
vxdco	Perform operations on version 0 DCO objects and DCO volumes.
vxdctl	Control the volume configuration daemon.
vxddladm	Device Discovery Layer subsystem administration.
vxdefault	Manage the defaults set in /etc/default/vxsf that configure settings such as SmartMove, thin reclamation, automatic starting of volumes, and minor numbers for shared disk groups.
vxdg	Manage Veritas Volume Manager disk groups.
vxdisk	Define and manage Veritas Volume Manager disks.
vxdiskadd	Add one or more disks for use with Veritas Volume Manager.
vxdiskadm	Menu-driven Veritas Volume Manager disk administration.
vxdisksetup	Configure a disk for use with Veritas Volume Manager.

Table C-9 Section 1M manual pages *(continued)*

Name	Description
`vxdiskunsetup`	Deconfigure a disk from use with Veritas Volume Manager.
`vxdmpadm`	DMP subsystem administration.
`vxdmptune`	Display and change values of DMP tunable parameters. This utility is deprecated. Use the `vxdmpadm` utility instead.
`vxedit`	Create, remove, and modify Veritas Volume Manager records.
`vxencap`	Encapsulate partitions on a new disk.
`vxevac`	Evacuate all volumes from a disk.
`vxinfo`	Print accessibility and usability of volumes.
`vxinitrd`	Create initial ramdisk images for preloading VxVM modules.
`vxinstall`	Menu-driven Veritas Volume Manager initial configuration.
`vxintro`	Introduction to the Veritas Volume Manager utilities.
`vxiod`	Start, stop, and report on Veritas Volume Manager kernel I/O threads.
`vxmake`	Create Veritas Volume Manager configuration records.
`vxmemstat`	Display memory statistics for Veritas Volume Manager.
`vxmend`	Mend simple problems in configuration records.
`vxmirror`	Mirror volumes on a disk or control default mirroring.
`vxnotify`	Display Veritas Volume Manager configuration events.
`vxplex`	Perform Veritas Volume Manager operations on plexes.

Table C-9 Section 1M manual pages *(continued)*

Name	Description
`vxprint`	Display records from the Veritas Volume Manager configuration.
`vxr5check`	Verify RAID-5 volume parity.
`vxreattach`	Reattach disk drives that have become accessible again.
`vxrecover`	Perform volume recovery operations.
`vxrelayout`	Convert online storage from one layout to another.
`vxrelocd`	Monitor Veritas Volume Manager for failure events and relocate failed subdisks.
`vxresize`	Change the length of a volume containing a file system.
`vxrootadm`	Grow or take snapshots of the boot disk.
`vxrootmir`	Mirror root disk to an alternate disk.
`vxscsiinq`	Display SCSI inquiry data.
`vxsd`	Perform Veritas Volume Manager operations on subdisks.
`vxsnap`	Enable DRL on a volume, and create and administer instant snapshots.
`vxstat`	Veritas Volume Manager statistics management utility.
`vxtask`	List and administer Veritas Volume Manager tasks.
`vxtrace`	Trace operations on volumes.
`vxtranslog`	Administer transaction logging.
`vxtune`	Adjust Veritas Volume Replicator and Veritas Volume Manager tunables.
`vxunreloc`	Move a hot-relocated subdisk back to its original disk.

Table C-9 Section 1M manual pages *(continued)*

Name	Description
`vxunroot`	Remove Veritas Volume Manager hooks from encapsulated root volumes.
`vxvol`	Perform Veritas Volume Manager operations on volumes.
`vxvoltune`	Display and change values of VxVM tunable parameters. This utility is deprecated. Use the `vxtune` command instead.
`vxvset`	Create and administer volume sets.

Section 4 — file formats

Table C-10 lists the manual pages in section 4 that describe the format of files that are used by Veritas Volume Manager.

Table C-10 Section 4 manual pages

Name	Description
`vol_pattern`	Disk group search specifications.
`vxmake`	`vxmake` description file.

Veritas File System command summary

Symbolic links to all VxFS command executables are installed in the `/opt/VRTS/bin` directory. Add this directory to the end of your `PATH` environment variable to access the commands.

Table C-11 describes the VxFS-specific commands.

Table C-11 VxFS commands

Command	Description
`df`	Reports the number of free disk blocks and inodes for a VxFS file system.
`fcladm`	Administers VxFS File Change Logs.
`ff`	Lists file names and inode information for a VxFS file system.
`fiostat`	Administers file I/O statistics

Table C-11 VxFS commands *(continued)*

Command	Description
`fsadm`	Resizes or defragments a VxFS file system.
`fsapadm`	Administers VxFS allocation policies.
`fscat`	Cats a VxFS file system.
`fscdsadm`	Performs online CDS operations.
`fscdsconv`	Performs offline CDS migration tasks on VxFS file systems.
`fscdstask`	Performs various CDS operations.
`fsck`	Checks and repairs a VxFS file system. Due to a behavioral issue with the Linux `fsck` wrapper, you must run the VxFS `fsck` command, `/opt/VRTS/bin/fsck`, when specifying any option with an equals sign (=) in it. For example: `# /opt/VRTS/bin/fsck -o zapvol=MyVolName /dev/rdsk/c0t0d1s1`
`fsckpt_restore`	Restores file systems from VxFS Storage Checkpoints.
`fsclustadm`	Manages cluster-mounted VxFS file systems.
`fsdb`	Debugs VxFS file systems.
`fsdedupadm`	Administers data deduplication.
`fsfreeze`	Freezes VxFS file systems and executes a user command on the file systems.
`fsmap`	Displays VxFS file system extent information.
`fsppadm`	Administers VxFS placement policies.
`fsppmk`	Creates placement policies.
`fstag`	Creates, deletes, or lists file tags.
`fstyp`	Returns the type of file system on a specified disk partition.
`fsvmap`	Maps volumes of VxFS file systems to files.
`fsvoladm`	Administers VxFS volumes.
`glmconfig`	Configures Group Lock Managers (GLM).
`glmdump`	Reports stuck Group Lock Managers (GLM) locks in a cluster file system.

Table C-11 VxFS commands *(continued)*

Command	Description
glmstat	Group Lock Managers (GLM) statistics gathering utility.
mkdstfs	SmartTier file system creation utility.
mkfs	Constructs a VxFS file system.
mount	Mounts a VxFS file system.
ncheck	Generates path names from inode numbers for a VxFS file system.
setext	Sets extent attributes on a file in a VxFS file system.
vxcompress	Compresses and uncompresses files.
vxdump	Incrementally dumps file systems.
vxedquota	Edits user quotas for a VxFS file system.
vxenablef	Enables specific VxFS features.
vxfilesnap	Makes a copy-on-write copy of a file in a VxFS file system.
vxfsconvert	Converts an unmounted file system to VxFS or upgrades a VxFS disk layout version.
vxfsstat	Displays file system statistics.
vxlsino	Looks up VxFS reverse path names.
vxquot	Displays file system ownership summaries for a VxFS file system.
vxquota	Displays user disk quotas and usage on a VxFS file system.
vxquotaoff vxquotaon	Turns quotas on and off for a VxFS file system.
vxrepquota	Summarizes quotas for a VxFS file system.
vxrestore	Restores a file system incrementally.
vxtunefs	Tunes a VxFS file system.
vxupgrade	Upgrades the disk layout of a mounted VxFS file system.

Veritas File System manual pages

This release includes the following online manual pages as part of the VRTSvxfs rpm. These are installed in the appropriate directories under /opt/VRTS/man (add this to your MANPATH environment variable), but does not update the windex database. To ensure that new VxFS manual pages display correctly, update the windex database after installing VRTSvxfs.

See the catman(1M) manual page.

Table C-12 describes the VxFS-specific section 1 manual pages.

Table C-12 Section 1 manual pages

Section 1	Description
fiostat	Administers file I/O statistics.
fsmap	Displays VxFS file system extent information.
getext	Gets extent attributes for a VxFS file system.
setext	Sets extent attributes on a file in a VxFS file system.
vxcompress	Compresses or uncompresses files.
vxfilesnap	Makes a copy-on-write copy of a file in a VxFS file system.

Table C-13 describes the VxFS-specific section 1M manual pages.

Table C-13 Section 1M manual pages

Section 1M	Description
df_vxfs	Reports the number of free disk blocks and inodes for a VxFS file system.
fcladm	Administers VxFS File Change Logs.
ff_vxfs	Lists file names and inode information for a VxFS file system.
fsadm_vxfs	Resizes or reorganizes a VxFS file system.
fsapadm	Administers VxFS allocation policies.
fscat_vxfs	Cats a VxFS file system.
fscdsadm	Performs online CDS operations.
fscdsconv	Performs offline CDS migration tasks on VxFS file systems.

Table C-13 Section 1M manual pages *(continued)*

Section 1M	Description
`fscdstask`	Performs various CDS operations.
`fsck_vxfs`	Checks and repairs a VxFS file system.
`fsckptadm`	Performs various administrative tasks like creating, deleting, converting, setting, and displaying the quota on a Storage Checkpoint. Quota display can be formatted in a human-friendly way, using the `-H` option.
`fsckpt_restore`	Restores file systems from VxFS Storage Checkpoints.
`fsclustadm`	Manages cluster-mounted VxFS file systems.
`fsdbencap`	Encapsulates databases.
`fsdb_vxfs`	Debugs VxFS file systems.
`fsdedupadm`	Administers data deduplication.
`fsfreeze`	Freezes VxFS file systems and executes a user command on the file systems.
`fsppadm`	Administers VxFS placement policies.
`fstyp_vxfs`	Returns the type of file system on a specified disk partition.
`fsvmap`	Maps volumes of VxFS file systems to files.
`fsvoladm`	Administers VxFS volumes.
`glmconfig`	Configures Group Lock Managers (GLM). This functionality is available only with the Storage Foundation Cluster File System High Availability product.
`glmdump`	Reports stuck Group Lock Managers (GLM) locks in a cluster file system.
`mkdstfs`	SmartTier file system creation utility.
`mkfs_vxfs`	Constructs a VxFS file system.
`mount_vxfs`	Mounts a VxFS file system.
`ncheck_vxfs`	Generates path names from inode numbers for a VxFS file system.
`quot`	Summarizes ownership on a VxFS file system.
`quotacheck_vxfs`	Checks VxFS file system quota consistency.
`vxdiskusg`	Generates VxFS disk accounting data by user ID.

Table C-13 Section 1M manual pages *(continued)*

Section 1M	Description
vxdump	Incrementally dumps file systems.
vxedquota	Edits user quotas for a VxFS file system.
vxenable	Enables specific VxFS features.
vxfsconvert	Converts an unmounted file system to VxFS or upgrades a VxFS disk layout version.
vxfsstat	Displays file system statistics.
vxlsino	Looks up VxFS reverse path names.
vxquot	Displays file system ownership summaries for a VxFS file system.
vxquota	Displays user disk quotas and usage on a VxFS file system.
vxquotaoff vxquotaon	Turns quotas on and off for a VxFS file system.
vxrepquota	Summarizes quotas for a VxFS file system.
vxrestore	Restores a file system incrementally.
vxtunefs	Tunes a VxFS file system.
vxupgrade	Upgrades the disk layout of a mounted VxFS file system.

Table C-14 describes the VxFS-specific section 3 manual pages.

Table C-14 Section 3 manual pages

Section 3	Description
vxfs_ap_alloc2	Allocates an fsap_info2 structure.
vxfs_ap_assign_ckpt	Assigns an allocation policy to file data and metadata in a Storage Checkpoint.
vxfs_ap_assign_ckptchain	Assigns an allocation policy for all of the Storage Checkpoints of a VxFS file system.
vxfs_ap_assign_ckptdef	Assigns a default allocation policy for new Storage Checkpoints of a VxFS file system.
vxfs_ap_assign_file	Assigns an allocation policy for file data and metadata.
vxfs_ap_assign_file_pat	Assigns a pattern-based allocation policy for a directory.

Table C-14 Section 3 manual pages *(continued)*

Section 3	Description
`vxfs_ap_assign_fs`	Assigns an allocation policy for all file data and metadata within a specified file system.
`vxfs_ap_assign_fs_pat`	Assigns an pattern-based allocation policy for a file system.
`vxfs_ap_define`	Defines a new allocation policy.
`vxfs_ap_define2`	Defines a new allocation policy.
`vxfs_ap_enforce_ckpt`	Reorganizes blocks in a Storage Checkpoint to match a specified allocation policy.
`vxfs_ap_enforce_ckptchain`	Enforces the allocation policy for all of the Storage Checkpoints of a VxFS file system.
`vxfs_ap_enforce_file`	Ensures that all blocks in a specified file match the file allocation policy.
`vxfs_ap_enforce_file2`	Reallocates blocks in a file to match allocation policies.
`vxfs_ap_enforce_range`	Reallocates blocks in a file within a specified range to match allocation policies.
`vxfs_ap_enumerate`	Returns information about all allocation policies.
`vxfs_ap_enumerate2`	Returns information about all allocation policies.
`vxf_ap_free2`	Frees one or more fsap_info2 structures.
`vxfs_ap_query`	Returns information about a specific allocation policy.
`vxfs_ap_query2`	Returns information about a specific allocation policy.
`vxfs_ap_query_ckpt`	Returns information about allocation policies for each Storage Checkpoint.
`vxfs_ap_query_ckptdef`	Retrieves the default allocation policies for new Storage Checkpoints of a VxFS file system
`vxfs_ap_query_file`	Returns information about allocation policies assigned to a specified file.
`vxfs_ap_query_file_pat`	Returns information about the pattern-based allocation policy assigned to a directory.
`vxfs_ap_query_fs`	Retrieves allocation policies assigned to a specified file system.
`vxfs_ap_query_fs_pat`	Returns information about the pattern-based allocation policy assigned to a file system.
`vxfs_ap_remove`	Deletes a specified allocation policy.

Table C-14 Section 3 manual pages *(continued)*

Section 3	Description
vxfs_fcl_sync	Sets a synchronization point in the VxFS File Change Log.
vxfs_fiostats_dump	Returns file and file range I/O statistics.
vxfs_fiostats_getconfig	Gets file range I/O statistics configuration values.
vxfs_fiostats_set	Turns on and off file range I/O statistics and resets statistics counters.
vxfs_get_ioffsets	Obtains VxFS inode field offsets.
vxfs_inotopath	Returns path names for a given inode number.
vxfs_inostat	Gets the file statistics based on the inode number.
vxfs_inotofd	Gets the file descriptor based on the inode number.
vxfs_nattr_check vxfs_nattr_fcheck	Checks for the existence of named data streams.
vxfs_nattr_link	Links to a named data stream.
vxfs_nattr_open	Opens a named data stream.
vxfs_nattr_rename	Renames a named data stream.
vxfs_nattr_unlink	Removes a named data stream.
vxfs_nattr_utimes	Sets access and modification times for named data streams.
vxfs_vol_add	Adds a volume to a multi-volume file system.
vxfs_vol_clearflags	Clears specified flags on volumes in a multi-volume file system.
vxfs_vol_deencapsulate	De-encapsulates a volume from a multi-volume file system.
vxfs_vol_encapsulate	Encapsulates a volume within a multi-volume file system.
vxfs_vol_encapsulate_bias	Encapsulates a volume within a multi-volume file system.
vxfs_vol_enumerate	Returns information about the volumes within a multi-volume file system.
vxfs_vol_queryflags	Queries flags on volumes in a multi-volume file system.
vxfs_vol_remove	Removes a volume from a multi-volume file system.
vxfs_vol_resize	Resizes a specific volume within a multi-volume file system.
vxfs_vol_setflags	Sets specified flags on volumes in a multi-volume file system.

Table C-14 Section 3 manual pages *(continued)*

Section 3	Description
`vxfs_vol_stat`	Returns free space information about a component volume within a multi-volume file system.

Table C-15 describes the VxFS-specific section 4 manual pages.

Table C-15 Section 4 manual pages

Section 4	Description
`fs_vxfs`	Provides the format of a VxFS file system volume.
`inode_vxfs`	Provides the format of a VxFS file system inode.
`tunefstab`	Describes the VxFS file system tuning parameters table.

Table C-16 describes the VxFS-specific section 7 manual pages.

Table C-16 Section 7 manual pages

Section 7	Description
`vxfsio`	Describes the VxFS file system control functions.

SmartIO command reference

Table C-17 lists commands for using the SmartIO feature.

For more information about SmartIO, refer to the *Veritas InfoScale SmartIO for Solid State Drives Solutions Guide*.

See the `sfcache`(1M) manual page.

Table C-17 SmartIO command reference

Command	Description
`sfcache app`	Applies the specified template name.
`sfcache create`	Creates a cache area.
`sfcache delete`	Deletes the specified cache area.
`sfcache disable`	Disables caching for the specified data object.

Table C-17 SmartIO command reference *(continued)*

Command	Description
`sfcache enable`	Enables caching for the specified data object.
`sfcache flush`	Flushes any write-back data for this file system or cache.
`sfcache list`	Displays the cached file systems or volumes and their cache usage.
`sfcache load`	Loads the specified file into the cache area.
`sfcache maxsize`	Displays the amount of free space in the devices that are already provisioned for caching.
`sfcache offline`	Stops VxFS or VxVM from using a cache area.
`sfcache online`	Explicitly makes a cache area available.
`sfcache pin`	Marks a file or directory to be held in the cache until the file or directory is deleted, truncated, or unpinned.
`sfcache purge`	Removes the cached contents for the specified file system.
`sfcache resize`	Resizes the specified cache area.
`sfcache restore-access`	Enables read or write access to files that are missing writeback data. This command does not restore the missing data.
`sfcache rmdev`	Removes the device or devices from use for caching.
`sfcache set`	Sets the values for the specified attributes.
`sfcache stat`	Displays the cache statistics, including cache hit rate, misses, average read and write latencies.
`sfcache unpin`	Removes the file or directory from the pinned state.