Veritas™ Volume Manager Administrator's Guide

Linux

5.1 Service Pack 1

Symantec

Veritas™ Volume Manager Administrator's Guide

Product version: 5.1 SP1

Document version: 5.1SP1.1

Legal Notice

Symantec Corporation
350 Ellis Street
Mountain View, CA 94043

http://www.symantec.com

Technical Support

Symantec Technical Support maintains support centers globally. Technical Support's primary role is to respond to specific queries about product features and functionality. The Technical Support group also creates content for our online Knowledge Base. The Technical Support group works collaboratively with the other functional areas within Symantec to answer your questions in a timely fashion. For example, the Technical Support group works with Product Engineering and Symantec Security Response to provide alerting services and virus definition updates.

Symantec's support offerings include the following:

- A range of support options that give you the flexibility to select the right amount of service for any size organization
- Telephone and/or Web-based support that provides rapid response and up-to-the-minute information
- Upgrade assurance that delivers software upgrades
- Global support purchased on a regional business hours or 24 hours a day, 7 days a week basis
- Premium service offerings that include Account Management Services

For information about Symantec's support offerings, you can visit our Web site at the following URL:

www.symantec.com/business/support/index.jsp

All support services will be delivered in accordance with your support agreement and the then-current enterprise technical support policy.

Contacting Technical Support

Customers with a current support agreement may access Technical Support information at the following URL:

www.symantec.com/business/support/contact_techsupp_static.jsp

Before contacting Technical Support, make sure you have satisfied the system requirements that are listed in your product documentation. Also, you should be at the computer on which the problem occurred, in case it is necessary to replicate the problem.

When you contact Technical Support, please have the following information available:

- Product release level

- Hardware information
- Available memory, disk space, and NIC information
- Operating system
- Version and patch level
- Network topology
- Router, gateway, and IP address information
- Problem description:
  - Error messages and log files
  - Troubleshooting that was performed before contacting Symantec
  - Recent software configuration changes and network changes

Licensing and registration

If your Symantec product requires registration or a license key, access our technical support Web page at the following URL:

www.symantec.com/business/support/

Customer service

Customer service information is available at the following URL:

www.symantec.com/business/support/

Customer Service is available to assist with non-technical questions, such as the following types of issues:

- Questions regarding product licensing or serialization
- Product registration updates, such as address or name changes
- General product information (features, language availability, local dealers)
- Latest information about product updates and upgrades
- Information about upgrade assurance and support contracts
- Information about the Symantec Buying Programs
- Advice about Symantec's technical support options
- Nontechnical presales questions
- Issues that are related to CD-ROMs or manuals

Documentation

Your feedback on product documentation is important to us. Send suggestions for improvements and reports on errors or omissions. Include the title and document version (located on the second page), and chapter and section titles of the text on which you are reporting. Send feedback to:

docs@symantec.com

About Symantec Connect

Symantec Connect is the peer-to-peer technical community site for Symantec's enterprise customers. Participants can connect and share information with other product users, including creating forum posts, articles, videos, downloads, blogs and suggesting ideas, as well as interact with Symantec product teams and Technical Support. Content is rated by the community, and members receive reward points for their contributions.

http://www.symantec.com/connect/storage-management

Support agreement resources

If you want to contact Symantec regarding an existing support agreement, please contact the support agreement administration team for your region as follows:

Asia-Pacific and Japan	customercare_apac@symantec.com
Europe, Middle-East, and Africa	semea@symantec.com
North America and Latin America	supportsolutions@symantec.com

Contents

Chapter 1

Understanding Veritas Volume Manager

This chapter includes the following topics:

About Veritas Volume Manager

Veritas™ Volume Manager (VxVM) by Symantec 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.

VxVM and the operating system

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.
`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.

`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.

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 VxVM handles storage management" on page 21.

How VxVM handles storage management

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 21.
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 26.

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 1-1 shows how a physical disk and device name (*devname*) are illustrated in this document.

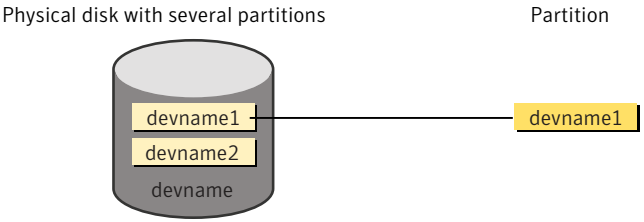
Figure 1-1 Physical disk example

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.

Partitions

Figure 1-2 shows how a physical disk can be divided into one or more partitions.

Figure 1-2 Partition example



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

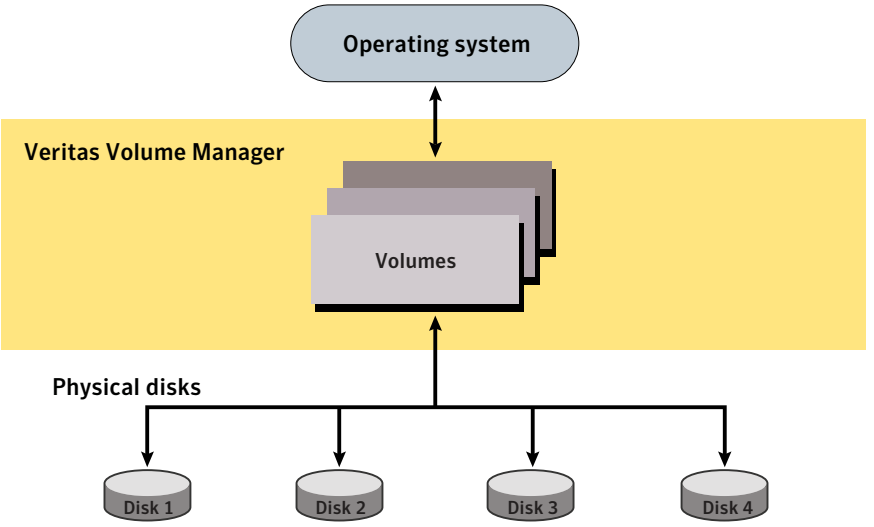
Disk arrays

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 1-3 shows how VxVM represents the disks in a disk array as several volumes to the operating system.

Figure 1-3 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.

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 159.

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 dynamically for new disk arrays. This operation, which uses a facility called the Device Discovery Layer (DDL), is achieved without the need for a reboot.

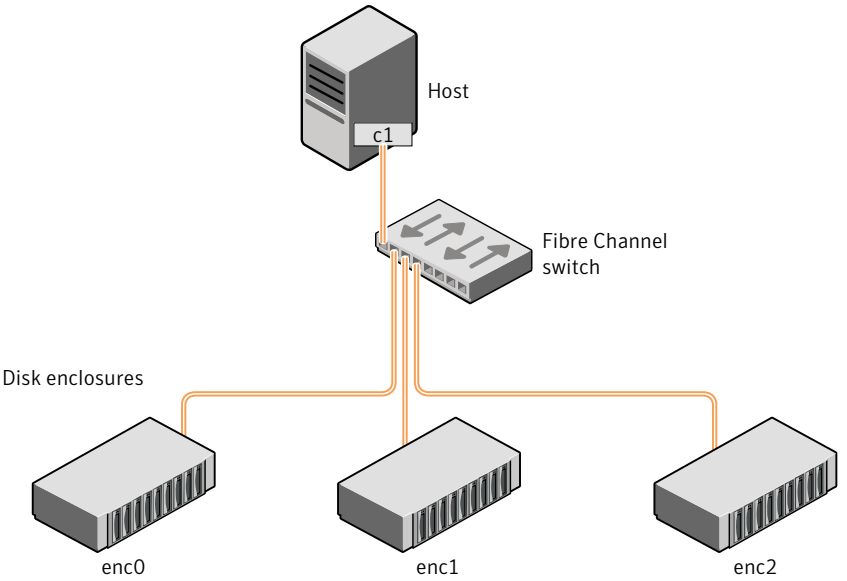
See "How to administer the Device Discovery Layer" on page 86.

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 VxVM 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 1-4 shows a typical SAN environment where host controllers are connected to multiple enclosures through a Fibre Channel switch.

Figure 1-4 Example configuration for disk enclosures connected via 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 allows you to 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.

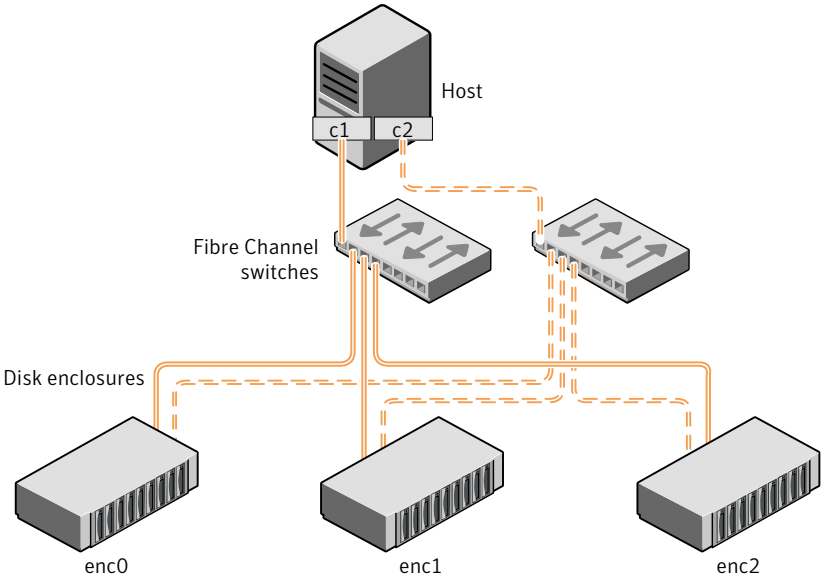
Another important benefit of enclosure-based naming is that it enables VxVM to avoid placing redundant copies of data in the same enclosure. This is a good thing to avoid as each enclosure can be considered to be a separate fault domain. For example, if a mirrored volume were configured only on the disks 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 VxVM assigns to an enclosure with one that is more meaningful to your configuration.

See "Renaming an enclosure" on page 202.

Figure 1-5 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 1-5 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 "Disk device naming in VxVM" on page 77.

See "Changing the disk-naming scheme" on page 100.

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

See "Mirroring across targets, controllers or enclosures" on page 331.

Virtual objects

VxVM uses multiple virtualization layers to provide distinct functionality and reduce physical limitations.

Virtual objects in VxVM include the following:

- Disk groups
  See "Disk groups" on page 29.
- VM disks
  See "VM disks" on page 29.
- Subdisks
  See "Subdisks" on page 30.
- Plexes
  See "Plexes" on page 31.
- Volumes
  See "Volumes" on page 32.

The connection between physical objects and VxVM objects is made when you place a physical disk under VxVM control.

After installing VxVM on a host system, you must bring the contents of physical disks under VxVM control by collecting the VM 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 VM disks co-exist or how to convert LVM disks to VM disks, see the *Veritas Storage Foundation Advanced Features Administrator's 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 "Displaying volume information" on page 348.

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

Combining virtual objects in VxVM

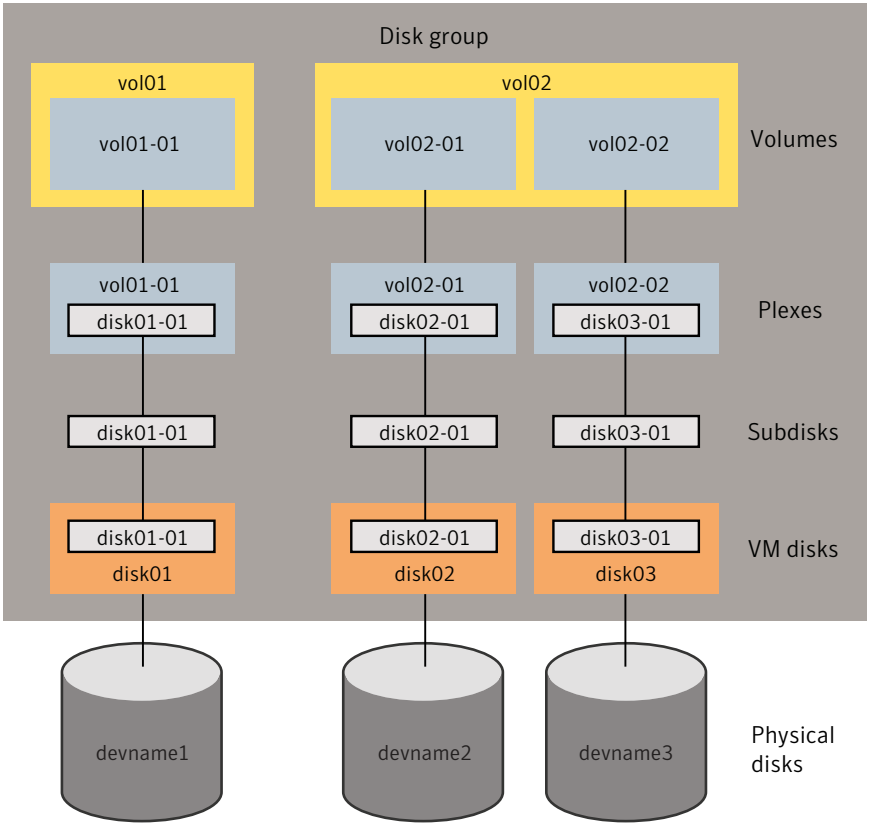
VxVM virtual objects are combined to build volumes. The virtual objects contained in volumes are VM disks, disk groups, subdisks, and plexes. VxVM virtual objects are organized in the following ways:

- VM 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 1-6 shows the connections between Veritas Volume Manager virtual objects and how they relate to physical disks.

Figure 1-6 Connection between objects in VxVM



The disk group contains three VM 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.

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 31 characters long.

See "VM disks" on page 29.

In releases before VxVM 4.0, the default disk group was `rootdg` (the root disk group). For VxVM to function, the `rootdg` disk group had to exist and it had to contain at least one disk. This requirement no longer exists, and VxVM can work without any disk groups configured (although you must set up at least one disk group before you can create any volumes of other VxVM objects).

See "System-wide reserved disk groups" on page 222.

You can create additional disk groups when you need them. Disk groups allow you to group disks into logical collections. A disk group and its components can be moved as a unit from one host machine to another.

See "Reorganizing the contents of disk groups" on page 262.

Volumes are created within a disk group. A given volume and its plexes and subdisks must be configured from disks in the same disk group.

VM disks

When you place a physical disk under VxVM control, a VM disk is assigned to the physical disk. A VM disk is under VxVM control and is usually in a disk group. Each VM disk corresponds to at least one physical disk or disk partition. VxVM allocates storage from a contiguous area of VxVM disk space.

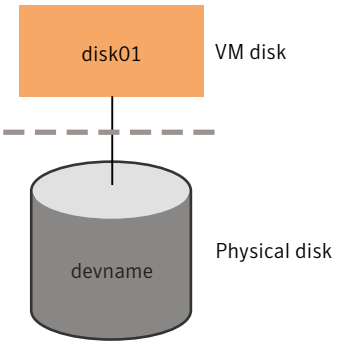
A VM disk typically includes a public region (allocated storage) and a small private region where VxVM internal configuration information is stored.

Each VM disk has a unique disk media name (a virtual disk name). You can either define a disk name of up to 31 characters, or allow VxVM to assign a default name that takes the form `diskgroup##`, where *diskgroup* is the name of the disk group to which the disk belongs.

See "Disk groups" on page 29.

Figure 1-7 shows a VM disk with a media name of `disk01` that is assigned to the physical disk, *devname*.

Figure 1-7 VM disk example



Subdisks

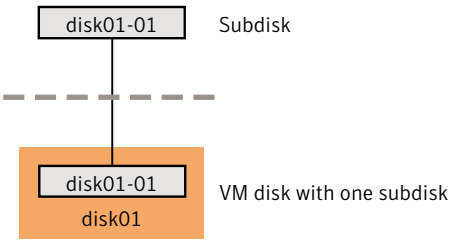
A subdisk is a set of contiguous disk blocks. A block is a unit of space on the disk. VxVM allocates disk space using subdisks. A VM disk can be divided into one or more subdisks. Each subdisk represents a specific portion of a VM disk, which is mapped to a specific region of a physical disk.

The default name for a VM disk is `diskgroup##` and the default name for a subdisk is `diskgroup##-##`, where *diskgroup* is the name of the disk group to which the disk belongs.

See "Disk groups" on page 29.

Figure 1-8 shows `disk01-01` is the name of the first subdisk on the VM disk named `disk01`.

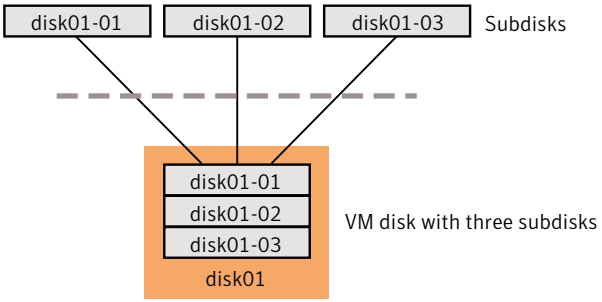
Figure 1-8 Subdisk example



A VM disk can contain multiple subdisks, but subdisks cannot overlap or share the same portions of a VM disk. To ensure integrity, VxVM rejects any commands that try to create overlapping subdisks.

Figure 1-9 shows a VM disk with three subdisks, which are assigned from one physical disk.

Figure 1-9 Example of three subdisks assigned to one VM Disk



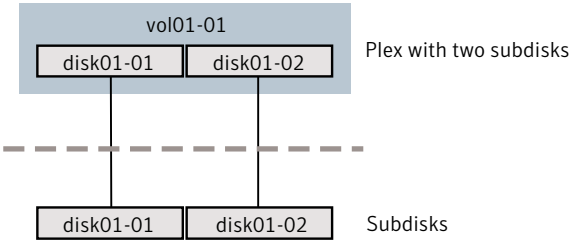
Any VM disk space that is not part of a subdisk is free space. You can use free space to create new subdisks.

Plexes

VxVM uses subdisks to build virtual objects called plexes. A plex consists of one or more subdisks located on one or more physical disks.

Figure 1-10 shows an example of a plex with two subdisks.

Figure 1-10 Example of a plex with two subdisks



You can organize data on subdisks to form a plex by using the following methods:

- concatenation
- striping (RAID-0)
- mirroring (RAID-1)
- striping with parity (RAID-5)

Concatenation, striping (RAID-0), mirroring (RAID-1) and RAID-5 are types of volume layout.

See "Volume layouts in VxVM" on page 33.

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.

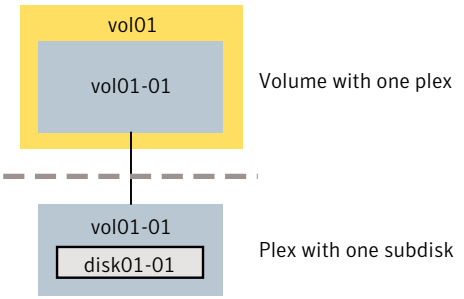
VxVM uses the default naming conventions of `vol##` for volumes and `vol##-##` for plexes in a volume. For ease of administration, you can choose to select more meaningful names for the volumes that you create.

A volume may be created under the following constraints:

- Its name can contain up to 31 characters.
- It can consist of up to 32 plexes, each of which contains one or more subdisks.
- It must have at least one associated plex that has a complete copy of the data in the volume with at least one associated subdisk.
- All subdisks within a volume must belong to the same disk group.

Figure 1-11 shows a volume `vol01` with a single plex.

Figure 1-11 Example of a volume with one plex

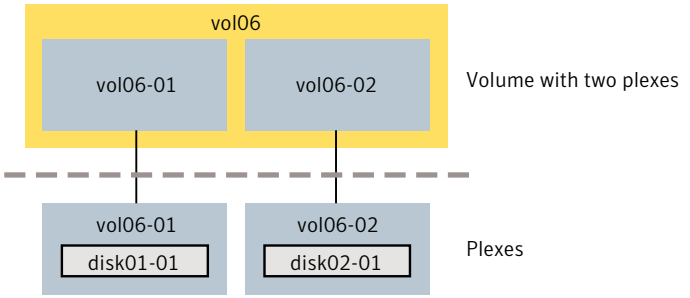


The volume `vol01` has the following characteristics:

- It contains one plex named `vol01-01`.
- The plex contains one subdisk named `disk01-01`.
- The subdisk `disk01-01` is allocated from VM disk `disk01`.

Figure 1-12 shows a mirrored volume, `vol06`, with two data plexes.

Figure 1-12 Example of a volume with two plexes



Each plex of the mirror contains a complete copy of the volume data.

The volume `vol06` has the following characteristics:

- It contains two plexes named `vol06-01` and `vol06-02`.
- Each plex contains one subdisk.
- Each subdisk is allocated from a different VM disk (`disk01` and `disk02`).

See "Mirroring (RAID-1)" on page 40.

VxVM supports the concept of layered volumes in which subdisks can contain volumes.

See "Layered volumes" on page 48.

Volume layouts in VxVM

A 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 VM disk. This allows the subdisk to define a contiguous extent of storage space backed by the public region of a VM disk. When active, the VM 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 VM 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 "Layered volumes" on page 48.

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 VM 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 35.
- Striping (RAID-0)
  See "Striping (RAID-0)" on page 37.
- Mirroring (RAID-1)
  See "Mirroring (RAID-1)" on page 40.
- Striping plus mirroring (mirrored-stripe or RAID-0+1)
  See "Striping plus mirroring (mirrored-stripe or RAID-0+1)" on page 41.
- 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 42.

- RAID-5 (striping with parity)
  See "RAID-5 (striping with parity)" on page 43.

Concatenation, spanning, and carving

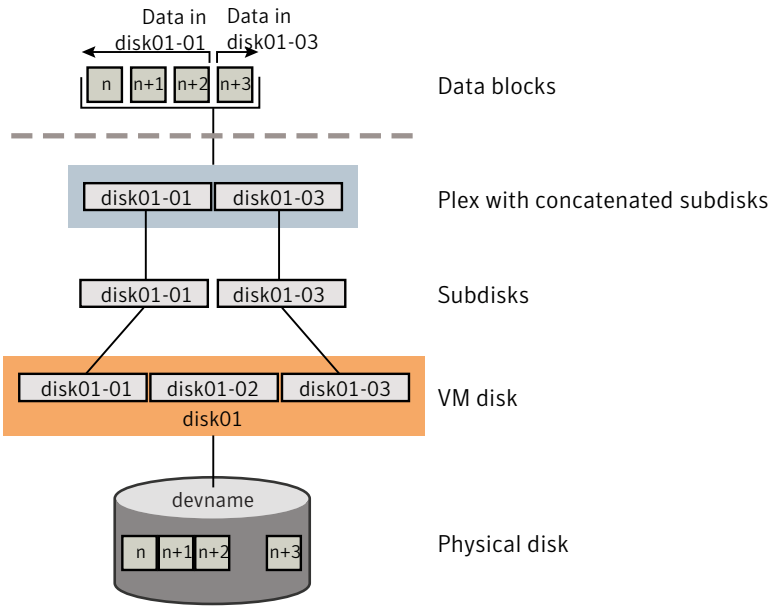
Concatenation maps data in a linear manner onto one or more subdisks in a plex. To access all of the data in a concatenated plex sequentially, data is first accessed in the first subdisk from the beginning to the end. Data is then accessed in the remaining subdisks sequentially from the beginning to 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 VM disk. Concatenation using subdisks that reside on more than one VM disk is called spanning.

Figure 1-13 shows the concatenation of two subdisks from the same VM 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 1-13 Example of concatenation



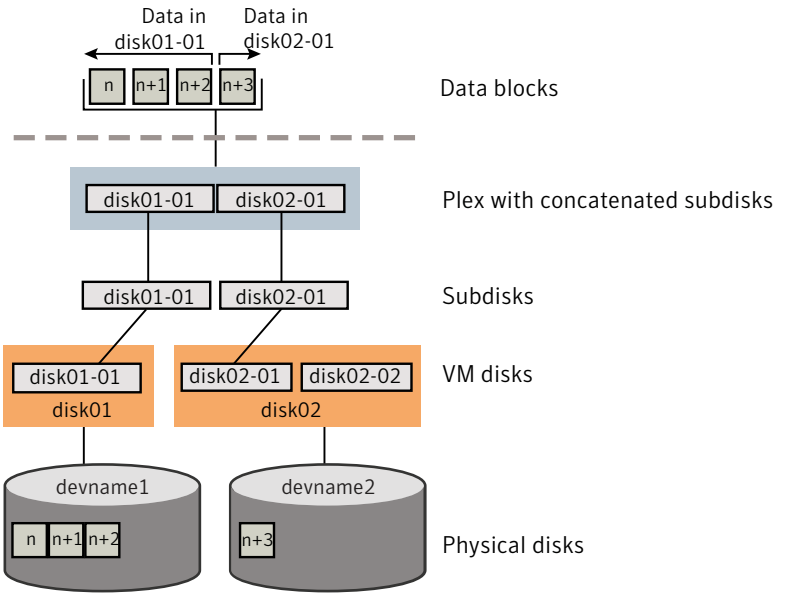
The blocks `n`, `n+1`, `n+2` and `n+3` (numbered relative to the start of the plex) are contiguous on the plex, but actually come from two distinct subdisks on the same physical disk.

The remaining free space in the subdisk, `disk01-02`, on VM 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 1-14 shows data spread over two subdisks in a spanned plex.

Figure 1-14 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 VM 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.

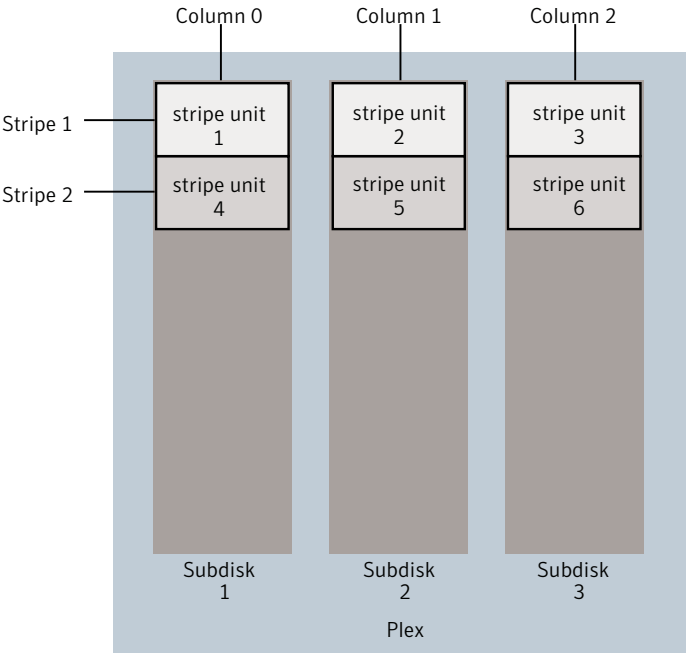
Warning: Striping a volume, or splitting a volume across multiple disks, increases the chance that a disk failure will result in failure of that volume.

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 1-15 shows an example with three columns in a striped plex, six stripe units, and data striped over the three columns.

Figure 1-15 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

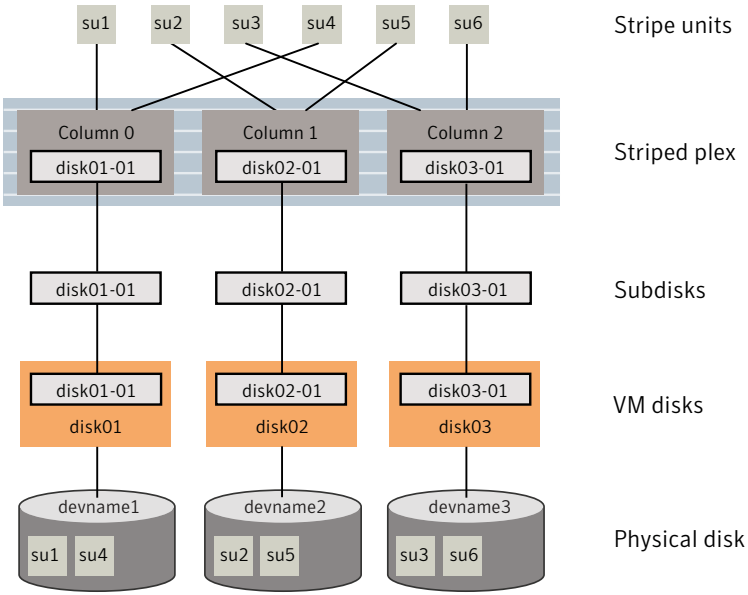
The second stripe consists of:

- 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 1-16 shows a striped plex with three equal sized, single-subdisk columns.

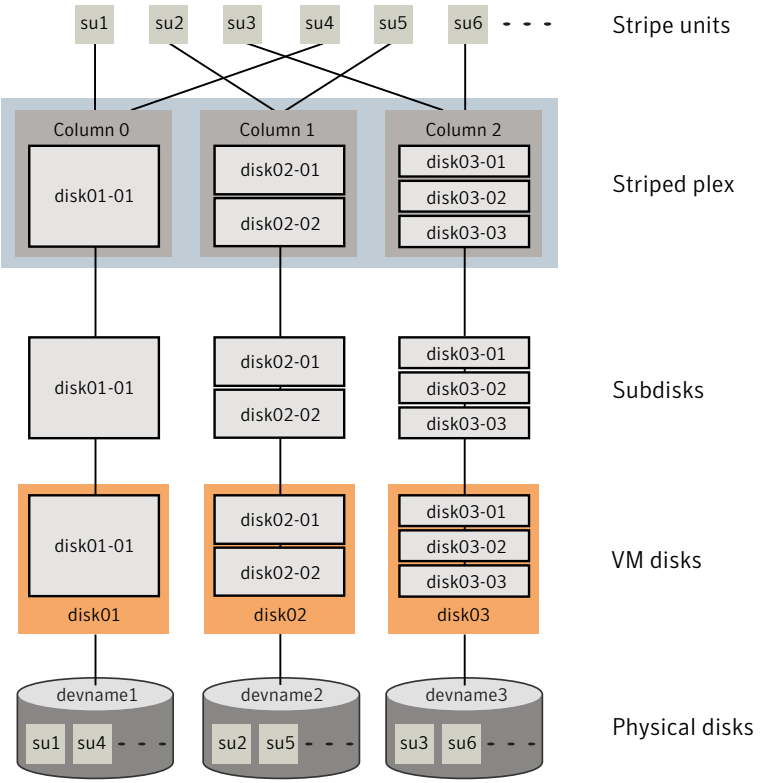
Figure 1-16 Example of a striped plex with one subdisk per column



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 1-17 shows a striped plex with three columns containing subdisks of different sizes.

Figure 1-17 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 329.

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 323.

See "Mirroring across targets, controllers or enclosures" on page 331.

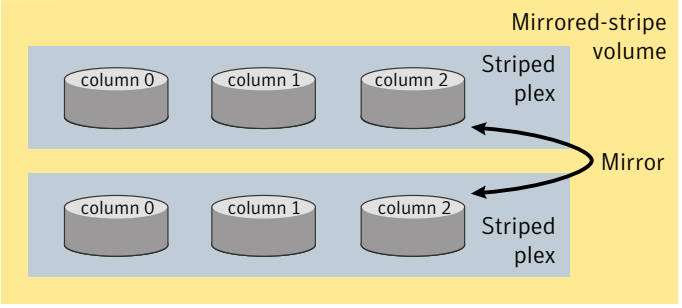
Striping plus mirroring (mirrored-stripe or RAID-0+1)

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 1-18 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 1-18 Mirrored-stripe volume laid out on six disks



See "Creating a mirrored-stripe volume" on page 330.

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)

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.

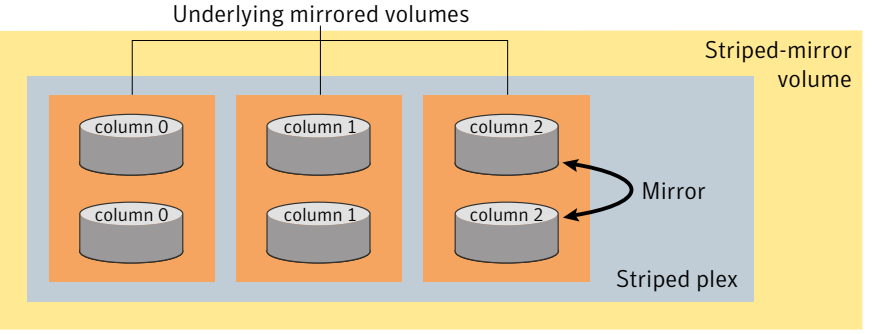
A striped-mirror volume is an example of a layered volume.

See "Layered volumes" on page 48.

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 1-19 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.

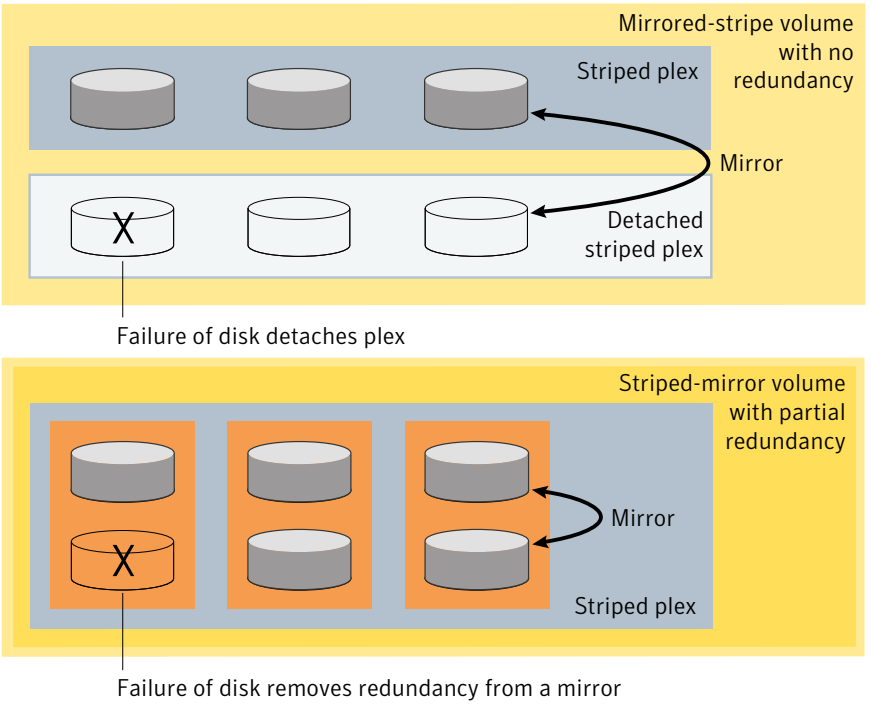
Figure 1-19 Striped-mirror volume laid out on six disks



See "Creating a striped-mirror volume" on page 331.

Figure 1-20 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 1-20 How the failure of a single disk affects mirrored-stripe and striped-mirror volumes



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)

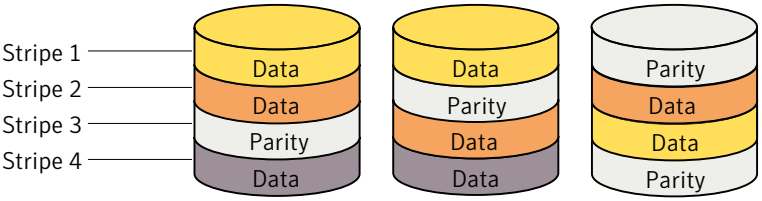
Although both mirroring (RAID-1) and RAID-5 provide redundancy of data, they use different methods. Mirroring provides data redundancy by maintaining multiple complete copies of the data in a volume. Data being written to a mirrored

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 1-21 shows parity locations in a RAID-5 array configuration.

Figure 1-21 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 45.

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

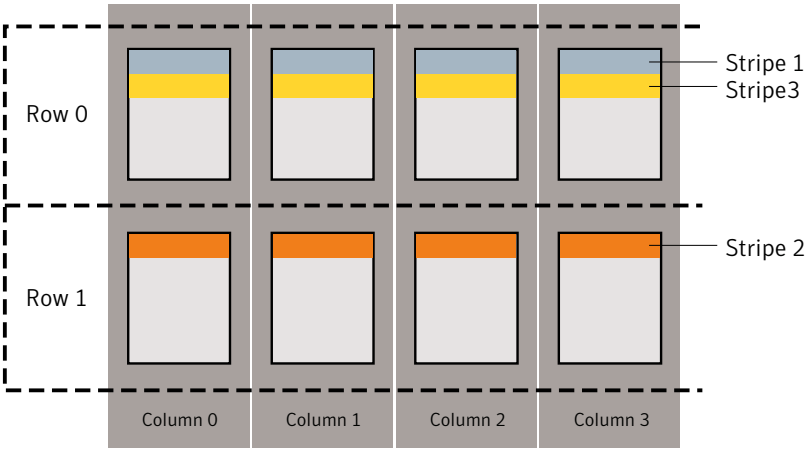
Traditional RAID-5 arrays

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 1-22 shows the row and column arrangement of a traditional RAID-5 array.

Figure 1-22 Traditional RAID-5 array



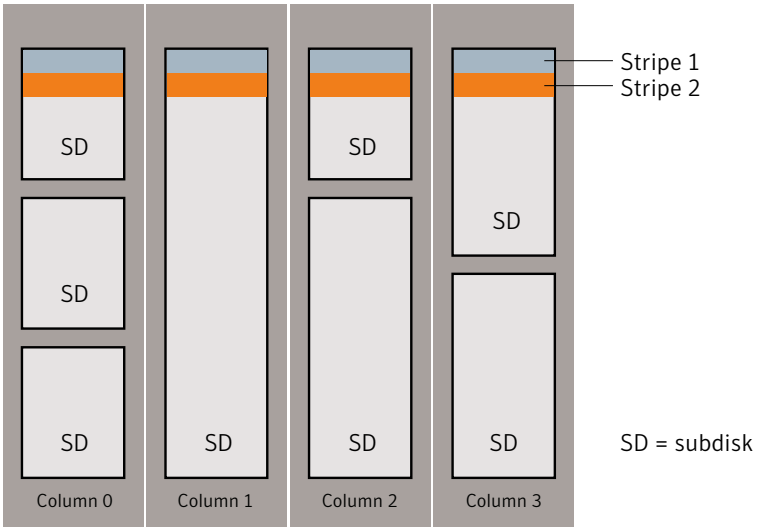
This traditional array structure supports growth by adding more rows per column. Striping is accomplished by applying the first stripe across the disks in Row 0, then the second stripe across the disks in Row 1, then the third stripe across the Row 0 disks, and so on. This type of array requires all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

Veritas Volume Manager RAID-5 arrays

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

Figure 1-23 shows how VxVM uses columns consisting of variable length subdisks, where each subdisk represents a specific area of a disk.

Figure 1-23 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 37.

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

See "Creating a RAID-5 volume" on page 333.

Left-symmetric layout

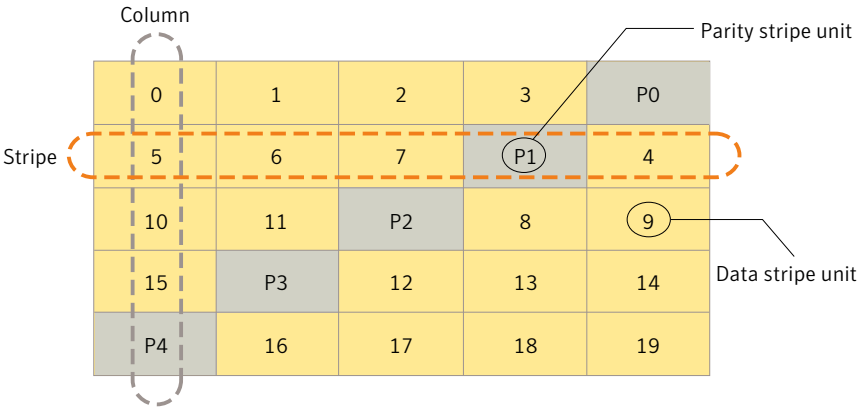
There are several layouts for data and parity that can be used in the setup of a RAID-5 array. The implementation of RAID-5 in VxVM uses a left-symmetric layout. This provides optimal performance for both random I/O operations and large sequential I/O operations. However, the layout selection is not 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 1-24 shows a left-symmetric parity layout with five disks (one per column).

Figure 1-24 Left-symmetric layout



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.

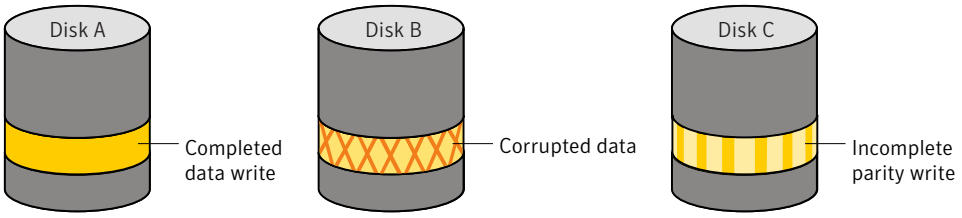
RAID-5 logging

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 1-25 shows a RAID-5 volume configured across three disks (A, B and C).

Figure 1-25 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.

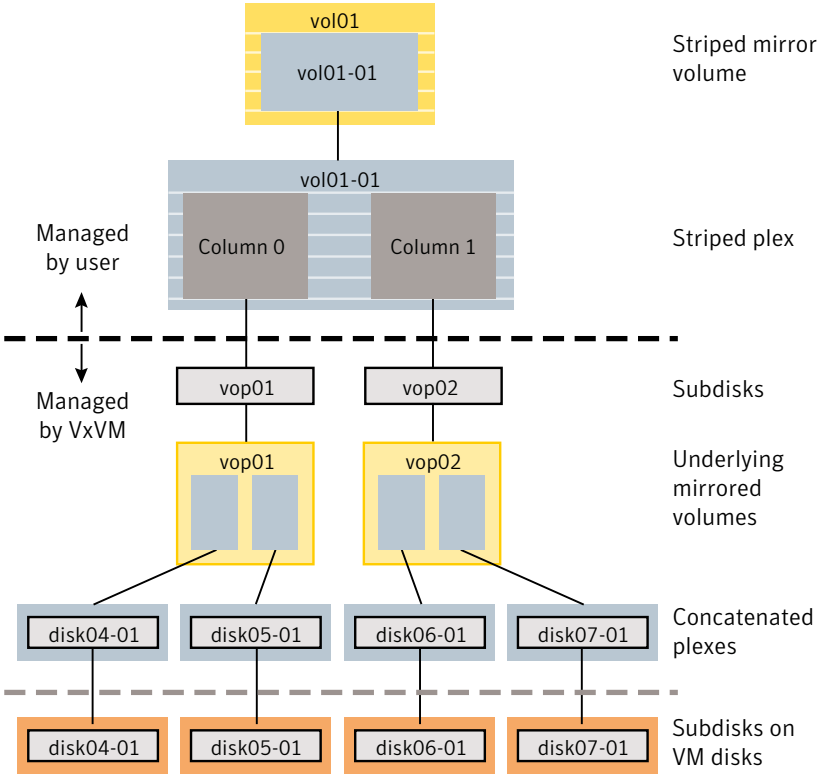
See "Adding a RAID-5 log" on page 394.

Layered volumes

A layered volume is a virtual Veritas Volume Manager 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 1-26 shows a typical striped-mirror layered volume where each column is represented by a subdisk that is built from an underlying mirrored volume.

Figure 1-26 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 331. See the `vxassist`(1M) manual page.
Creating concatenated-mirrors	See "Creating a concatenated-mirror volume" on page 324. See the `vxassist`(1M) manual page.
Online Relayout	See "Online relayout" on page 50. 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 the *Veritas Storage Foundation Advanced Features Administrator's Guide* 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 387.

See "Converting between layered and non-layered volumes" on page 393.

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`.

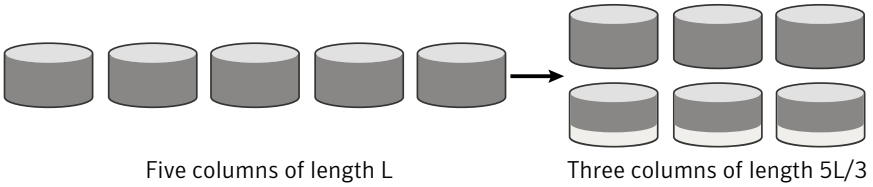
See the `vxassist`(1M) manual page.

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 1-27 shows how decreasing the number of columns can require disks to be added to a volume.

Figure 1-27 Example of decreasing the number of columns in a volume

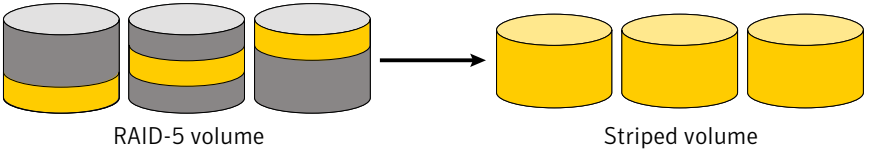


Note that the size of the volume remains the same but an extra disk is needed to extend one of the columns.

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 1-28 shows an example of applying relayout a RAID-5 volume.

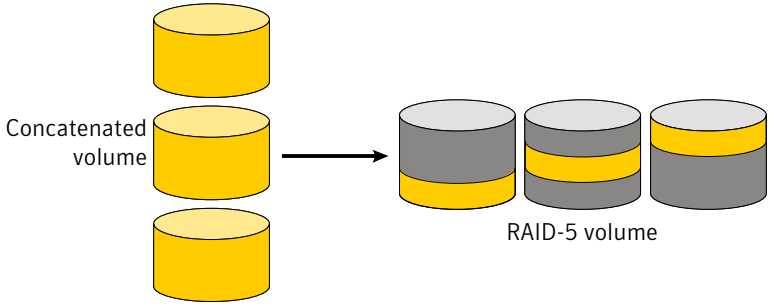
Figure 1-28 Example of relayout of a RAID-5 volume to a striped volume



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 1-29 shows an example.

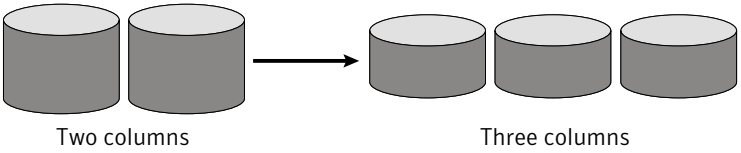
Figure 1-29 Example of relayout of a concatenated volume to a RAID-5 volume



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

- Change the number of columns in a volume.
  Figure 1-30 shows an example of changing the number of columns.

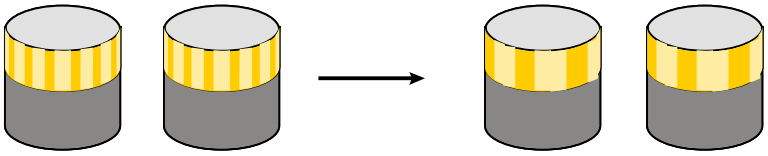
Figure 1-30 Example of increasing the number of columns in a volume



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 1-31 shows an example of changing the column stripe width.

Figure 1-31 Example of increasing the stripe width for the columns in a volume



See "Performing online relayout" on page 387.

See "Permitted relayout transformations" on page 387.

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.
See "Converting between layered and non-layered volumes" on page 393.

- 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 "Resizing a volume" on page 363.

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 56.

See "FastResync" on page 61.

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

See "SmartSync recovery accelerator" on page 57.

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 a version 20 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 a version 20 DCO volume.

See "DCO volume versioning" on page 63.

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`.

See "Tunable parameters for VxVM" on page 508.

See "Adding traditional DRL logging to a mirrored volume" on page 377.

See "Preparing a volume for DRL and instant snapshots" on page 371.

SmartSync recovery accelerator

The SmartSync feature of Veritas Volume Manager increases the availability of mirrored volumes by only resynchronizing changed data. (The process of resynchronizing mirrored databases is also sometimes referred to as resilvering.) SmartSync reduces the time required to restore consistency, freeing more I/O bandwidth for business-critical applications. SmartSync uses an extended interface between VxVM volumes, VxFS file systems, and the Oracle database to avoid unnecessary work during mirror resynchronization and to reduce the I/O overhead of the DRL. For example, Oracle® automatically takes advantage of SmartSync to perform database resynchronization when it is available.

Note: To use SmartSync with volumes that contain file systems, see the discussion of the Oracle Resilvering feature of Veritas File System (VxFS).

The following section describes how to configure VxVM raw volumes and SmartSync. The database uses the following types of volumes:

- Data volumes are the volumes used by the database (control files and tablespace files).
- Redo log volumes contain redo logs of the database.

SmartSync works with these two types of volumes differently, so they must be configured as described in the following sections.

Data volume configuration

The recovery takes place when the database software is started, not at system startup. This reduces the overall impact of recovery when the system reboots. Because the recovery is controlled by the database, the recovery time for the volume is the resilvering time for the database (that is, the time required to replay the redo logs).

Because the database keeps its own logs, it is not necessary for VxVM to do logging. Data volumes should be configured as mirrored volumes without dirty region logs. In addition to improving recovery time, this avoids any run-time I/O overhead due to DRL, and improves normal database write access.

Redo log volume configuration

A redo log is a log of changes to the database data. Because the database does not maintain changes to the redo logs, it cannot provide information about which sections require resilvering. Redo logs are also written sequentially, and since traditional dirty region logs are most useful with randomly-written data, they are of minimal use for reducing recovery time for redo logs. However, VxVM can reduce the number of dirty regions by modifying the behavior of its dirty region logging feature to take advantage of sequential access patterns. Sequential DRL decreases the amount of data needing recovery and reduces recovery time impact on the system.

The enhanced interfaces for redo logs allow the database software to inform VxVM when a volume is to be used as a redo log. This allows VxVM to modify the DRL behavior of the volume to take advantage of the access patterns. Since the improved recovery time depends on dirty region logs, redo log volumes should be configured as mirrored volumes with sequential DRL.

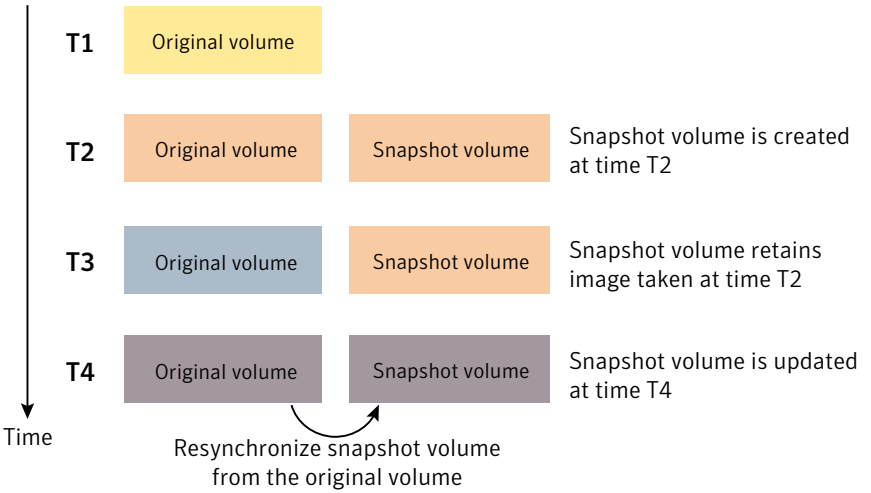
See “Sequential DRL” on page 57.

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 1-32 shows how a snapshot volume represents a copy of an original volume at a given point in time.

Figure 1-32 Volume snapshot as a point-in-time image of a 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.

See "FastResync" on page 61.

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.

See "Comparison of snapshot features" on page 60.

For details about the snapshots and how to use them, see the *Veritas Storage Foundation Advanced Features Administrator's Guide*.

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

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

Comparison of snapshot features

Table 1-1 compares the features of the various types of snapshots that are supported in VxVM.

Table 1-1 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

Table 1-1 Comparison of snapshot features for supported snapshot types *(continued)*

Snapshot feature	Full-sized instant (vxsnap)	Space-optimized instant (vxsnap)	Break-off (vxassist or vxsnap)
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
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.

FastResync

Note: Only certain Storage Foundation products have a 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 increases the efficiency of the 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.

See "Enabling FastResync on a volume" on page 385.

FastResync enhancements

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.
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.

Non-persistent FastResync

Non-persistent FastResync allocates its change maps in memory. They do not reside on disk nor in persistent store. This has the advantage that updates to the

FastResync map have little impact on I/O performance, as no disk updates needed to be performed. However, if a system is rebooted, the information in the map is lost, so 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.

How non-persistent FastResync works with snapshots

The snapshot feature of VxVM takes advantage of FastResync change tracking to record updates to the original volume after a snapshot plex is created. After a snapshot is taken, the `snapback` option is used to reattach the snapshot plex. Provided that FastResync is enabled on a volume before the snapshot is taken, and that it is not disabled at any time before the snapshot is reattached, the changes that FastResync records are used to resynchronize the volume during the snapback. This considerably reduces the time needed to resynchronize the volume.

Non-Persistent FastResync uses a map in memory to implement change tracking. Each bit in the 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.

See "Tunable parameters for VxVM" on page 508.

Persistent FastResync

Unlike non-persistent FastResync, persistent FastResync keeps the FastResync maps on disk so that they can survive system reboots, system crashes and cluster crashes. 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 262.

If 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.

DCO volume versioning

The internal layout of the DCO volume changed in VxVM 4.0 to support new features such as full-sized and space-optimized instant snapshots, and a unified

DRL/DCO. Because the DCO volume layout is versioned, VxVM software continues to support the version 0 layout for legacy volumes. However, you must configure a volume to have a version 20 DCO volume if you want to take instant snapshots of the volume. Future releases of Veritas Volume Manager may introduce new versions of the DCO volume layout.

See "Determining the DCO version number" on page 374.

Version 0 DCO volume layout

In earlier releases of VxVM, the DCO object only managed 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.

Persistent FastResync uses the maps in a version 0 DCO volume on disk to implement change tracking. As for non-persistent FastResync, each bit in the map represents a region (a contiguous number of blocks) in a volume's address space. 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 132 512-byte 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` that you need to specify. For example, to use an 8-block map, you would 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.

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

Version 20 DCO volume layout

In VxVM 4.0 and later releases, the DCO object 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.

See "Dirty region logging" on page 56.

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 a version 20 DCO volume includes an accumulator that stores the DRL map and a per-region state map for the volume, plus 32 per-volume maps (by default) including a DRL recovery map, and a map for tracking detaches that are initiated by the kernel due to I/O error. The remaining 30 per-volume maps (by default) are used either for tracking writes to snapshots, or as copymaps. The size of the DCO volume is determined by the size of the regions that are tracked, and by the number of per-volume maps. Both the region size and the number of per-volume maps in a DCO volume may be configured when a volume is prepared for use with snapshots. The region size must be a power of 2 and be greater than or equal to 16KB.

As the accumulator is approximately 3 times the size of a per-volume map, the size of each plex in the DCO volume can be estimated from this formula:

```
DCO_plex_size = ( 3 + number_of_per-volume_maps ) * map_size
```

where the size of each map in bytes is:

```
map_size = 512 + ( volume_size / ( region_size * 8 ))
```

rounded up to the nearest multiple of 8KB. Note that each map includes a 512-byte header.

For the default number of 32 per-volume maps and region size of 64KB, a 10GB volume requires a map size of 24KB, and so each plex in the DCO volume requires 840KB of storage.

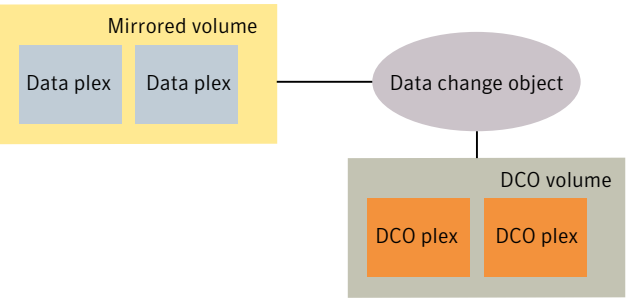
Note: Full-sized and space-optimized instant snapshots, which are administered using the `vxsnap` command, are supported for a version 20 DCO volume layout. The use of the `vxassist` command to administer traditional (third-mirror break-off) snapshots is not supported for a version 20 DCO volume layout.

How persistent FastResync works with snapshots

Persistent FastResync uses a map in a DCO volume on disk to implement change tracking. As for non-persistent FastResync, each bit in the map represents a contiguous number of blocks in a volume's address space.

Figure 1-33 shows an example of a mirrored volume with two plexes on which Persistent FastResync is enabled.

Figure 1-33 Mirrored volume with persistent FastResync enabled

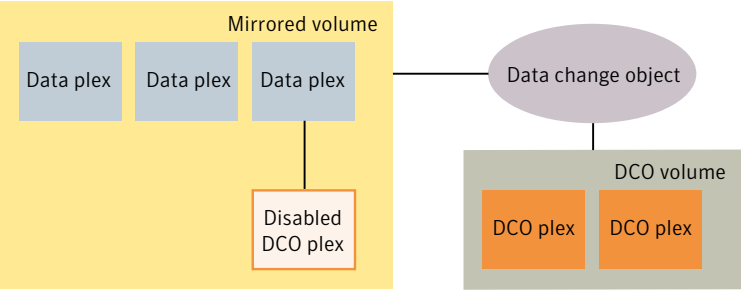


Associated with the volume are a DCO object and a DCO volume with two plexes.

To create a traditional third-mirror snapshot or an instant (copy-on-write) snapshot, the `vxassist snapstart` or `vxsnap make` operation respectively is performed on the volume.

Figure 1-34 shows how a snapshot plex is set up in the volume, and how a disabled DCO plex is associated with it.

Figure 1-34 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.

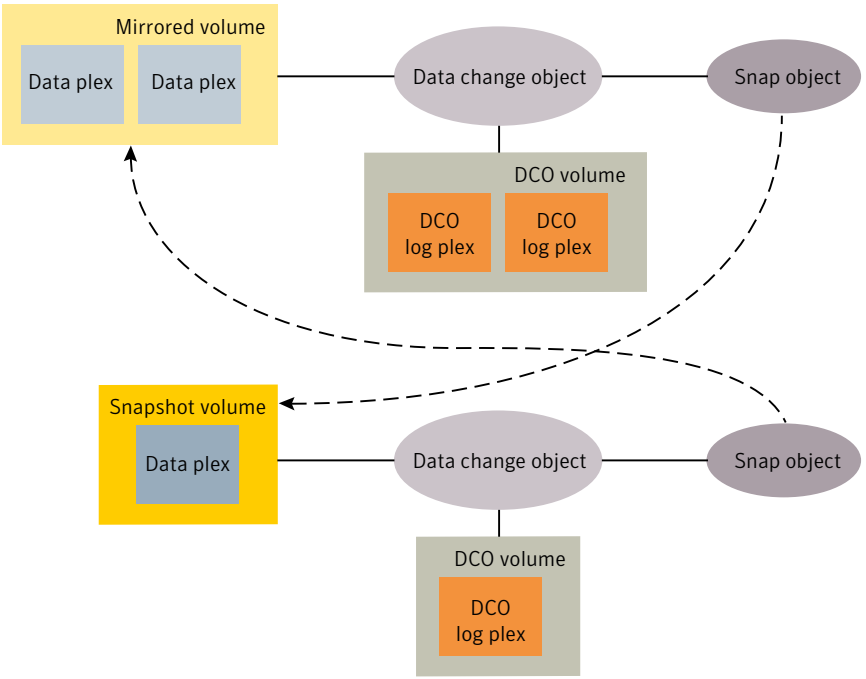
Space-optimized instant snapshots do not require additional full-sized plexes to be created. Instead, they use a storage cache that typically requires only 10% of the storage that is required by full-sized snapshots. There is a trade-off in functionality in using space-optimized snapshots. The storage cache is formed within a cache volume, and this volume is associated with a cache object. For convenience of operation, this cache can be shared by all the space-optimized instant snapshots within a disk group.

See "Comparison of snapshot features" on page 60.

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 1-35 shows how the creation of the snapshot volume also sets up a DCO object and a DCO volume for the snapshot volume.

Figure 1-35 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 VM 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.

For details about the snapshots and how to use them, see the *Veritas Storage Foundation Advanced Features Administrator's Guide*.

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

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

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 a version 20 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.

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 a version 20 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.

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 418.

Volume sets

Volume sets are an enhancement to VxVM that allow several volumes to be represented by a single logical object. All I/O from and to the underlying volumes is directed via the I/O interfaces of the volume set. The 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 398.

Chapter 2

Provisioning new usable storage

This chapter includes the following topics:

Provisioning new usable storage

The following procedure describes how to provision new usable storage.

To provision new usable 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 for Veritas Volume Manager (VxVM), using one of the following commands:

```
# 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 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 -t vxfs -g dg1 vol1 100g
```

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 VxVM aware of the new LUN size.

   ```
   # vxdisk -g dg1 resize c0t1d0s4
   ```

3. Calculate the new maximum volume size:

   ```
   # vxassist -b maxgrow vol1
   ```

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

   ```
   # vxresize -b -t vxfs -g dg1 vol1 150g
   ```

Chapter 3

Administering disks

This chapter includes the following topics:

About disk management

Veritas Volume Manager (VxVM) allows you to place LUNs and disks under VxVM control, to initialize or encapsulate disks, and to remove and replace disks.

Note: Most VxVM commands require superuser or equivalent privileges.

Disks that are controlled by LVM cannot be used directly as VxVM disks, but the disks can be converted so that their volume groups and logical volumes become VxVM disk groups and volumes.

For detailed information about migrating volumes, see the *Veritas Storage Foundation Advanced Features Administrator's Guide*.

Veritas Dynamic Multi-Pathing (DMP) is used to administer multiported disk arrays.

See "How DMP works" on page 159.

Disk devices

When performing disk administration, it is important to understand the difference between a disk name and a device name.

The disk name (also known as a disk media name) is the symbolic name assigned to a VM disk. When you place a disk under VxVM control, a VM disk is assigned to it. The disk name is used to refer to the VM disk for the purposes of administration. A disk name can be up to 31 characters long. When you add a disk to a disk group, you can assign a disk name or allow VxVM to assign a disk name. The default disk name is `diskgroup##` where `diskgroup` is the name of the disk group to which the disk is being added, and `##` is a sequence number. Your system may use device names that differ from those given in the examples.

The device name (sometimes referred to as devname or disk access name) defines the name of a disk device as it is known to the operating system.

Such devices are usually, but not always, located in the `/dev` directory. Devices that are specific to hardware from certain vendors may use their own path name conventions.

VxVM supports the disk partitioning scheme provided by the operating system. The syntax of a device name is `hdx[N]` or `sdx[N]`, where *x* is a letter that indicates the order of EIDE (`hd`) or SCSI (`sd`) disks seen by the operating system, and *N* is an optional partition number in the range 1 through 15. An example of a device name is `sda7`, which references partition 7 on the first SCSI disk. If the partition number is omitted, the device name indicates the entire disk.

Devices that are specific to hardware from certain vendors may have different path names. For example, the COMPAQ SMART and SMARTII controllers use device names of the form `/dev/ida/cXdXpX` and `/dev/cciss/cXdXpX`.

VxVM uses the device names to create metadevices in the `/dev/vx/[r]dmp` directories. Dynamic Multi-Pathing (DMP) uses these metadevices (or DMP nodes) to represent disks that can be accessed by one or more physical paths, perhaps via different controllers. The number of access paths that are available depends on whether the disk is a single disk, or is part of a multiported disk array that is connected to a system.

You can use the `vxdisk` utility to display the paths that are subsumed by a DMP metadevice, and to display the status of each path (for example, whether it is enabled or disabled).

See "How DMP works" on page 159.

Device names may also be remapped as enclosure-based names.

See "Disk device naming in VxVM" on page 77.

Disk device naming in VxVM

Device names for disks are assigned according to the naming scheme which you specify to VxVM. The format of the device name may vary for different categories of disks.

See "Disk categories" on page 84.

Device names can use one of the following naming schemes:

- Operating system-based naming
- Enclosure-based naming

Devices with device names longer than 31 characters always use enclosure-based names.

By default, VxVM uses enclosure-based naming.

You can change the disk-naming scheme if required.

See "Changing the disk-naming scheme" on page 100.

Operating system-based naming

In the OS-based naming scheme, all disk devices are named using the `hdx[N]` format or the `sdx[N]` format, where *x* is a letter that indicates the order of EIDE (`hd`) or SCSI (`sd`) disks seen by the operating system, and *N* is an optional partition number in the range 1 through 15.

DMP assigns the name of the DMP meta-device (disk access name) from the multiple paths to the disk. DMP sorts the names by sd or hd number, and selects the smallest number. For example, sd1 rather than sd2.This behavior make it easier to correlate devices with the underlying storage.

If a CVM cluster is symmetric, each node in the cluster accesses the same set of disks. This naming scheme makes the naming consistent across nodes in a symmetric cluster.

By default, OS-based names are not persistent, and are regenerated if the system configuration changes the device name as recognized by the operating system. If you do not want the OS-based names to change after reboot, set the persistence attribute for the naming scheme.

See "Changing the disk-naming scheme" on page 100.

Enclosure-based naming

By default, VxVM uses enclosure-based naming.

Enclosure-based naming operates as follows:

- All fabric or non-fabric disks in supported disk arrays are named using the *`enclosure_name_#`* format. For example, disks in the supported disk array, `enggdept` are named `enggdept_0`, `enggdept_1`, `enggdept_2` and so on.
  You can use the `vxdmpadm` command to administer enclosure names.
  See "Renaming an enclosure" on page 202.
  See the `vxdmpadm`(1M) manual page.
- Disks in the `DISKS` category (JBOD disks) are named using the `Disk_#` format.
- A disk partition is indicated by appending *`s#`* to the name, where *#* is the partition number. For example, `Disk_0s5` and `Disk_0s6` indicate the extended partitions that are used for the private and public regions of the sliced disk `Disk_0`. `ACME_0s5` indicates the extended partition for the simple disk, `ACME_0`. For CDS disks, partition 3 is used for both the private and public regions.
  See "Private and public disk regions" on page 79.

- Disks in the `OTHER_DISKS` category (disks that are not multi-pathed by DMP) are named using the `hdx[N]` format or the `sdx[N]` format.
- Encapsulated root disks always use the `hdx[N]` format or the `sdx[N]` format.

By default, enclosure-based names are persistent, so they do not change after reboot.

If a 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.

To display the native OS device names of a VxVM disk (such as `mydg01`), use the following command:

```
# vxdisk path | grep diskname
```

See "Renaming an enclosure" on page 202.

See "Disk categories" on page 84.

Private and public disk regions

Most VM disks consist of the following regions:

private region	A small area where configuration information is stored, including a disk header label, configuration records for VxVM objects, and an intent log for the configuration database. The default private region size is 32 megabytes, which is large enough to record the details of several thousand VxVM objects in a disk group. Under most circumstances, the default private region size should be sufficient. For administrative purposes, it is usually much simpler to create more disk groups that contain fewer volumes, or to split large disk groups into several smaller ones. See "Splitting disk groups" on page 272. If required, the value for the private region size may be overridden when you add or replace a disk using the `vxdiskadm` command. Each disk that has a private region holds an entire copy of the configuration database for the disk group. The size of the configuration database for a disk group is limited by the size of the smallest copy of the configuration database on any of its member disks.
public region	An area that covers the remainder of the disk, and which is used for the allocation of storage space to subdisks.

A disk's type identifies how VxVM accesses a disk, and how it manages the disk's private and public regions.

The following disk access types are used by VxVM:

`auto`	When the `vxconfigd` daemon is started, VxVM obtains a list of known disk device addresses from the operating system and configures disk access records for them automatically.
`nopriv`	There is no private region (only a public region for allocating subdisks). This is the simplest disk type consisting only of space for allocating subdisks. Such disks are most useful for defining special devices (such as RAM disks, if supported) on which private region data would not persist between reboots. They can also be used to encapsulate disks where there is insufficient room for a private region. The disks cannot store configuration and log copies, and they do not support the use of the `vxdisk addregion` command to define reserved regions. VxVM cannot track the movement of `nopriv` disks on a SCSI chain or between controllers.
`simple`	The public and private regions are both configured on a single extended partition, such as partition 5, with the public area following the private area. The partition number (displayed as `#` or `s#`, depending on the naming scheme) is always displayed in command output for simple disks.
`sliced`	The private and public regions are configured on different extended partitions: partition 5 for the private region, and partition 6 for the public region.

Auto-configured disks (with disk access type `auto`) support the following disk formats:

`cdsdisk`	The disk is formatted as a Cross-platform Data Sharing (CDS) disk that is suitable for moving between different operating systems. This is the default format for disks that are not used to boot the system. Typically, most disks on a system are configured as this disk type. However, it is not a suitable format for boot, root or swap disks, or for mirrors or hot-relocation spares of such disks. Also note that partition 3 is used for both the private and public regions of a CDS disk.
`simple`	The disk is formatted as a simple disk that can be converted to a CDS disk.

`sliced`	The disk is formatted as a sliced disk. This format can be applied to disks that are used to boot the system. The disk can be converted to a CDS disk if it was not initialized for use as a boot disk.

The `vxcdsconvert` utility can be used to convert disks to the `cdsdisk` format.

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

Warning: If a disk is initialized by VxVM as a CDS disk, the CDS header occupies the portion of the disk where the partition table would usually be located. If you subsequently use a command such as `fdisk` to create a partition table on a CDS disk, it erases the CDS information and could cause data corruption.

By default, all auto-configured disks are formatted as `cdsdisk` disks when they are initialized for use with VxVM. You can change the default format by using the `vxdiskadm(1M)` command to update the `/etc/default/vxdisk` defaults file.

See "Displaying or changing default disk layout attributes" on page 106.

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

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 DMP internal database to reflect the new state of the system.

To reconfigure the DMP database, first reboot the system to 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 VxVM 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 sets of disks or disks attached to controllers 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 VxVM 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.

Discovering disks and dynamically adding disk arrays

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, 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 plugins that allow close tie-in of DMP with any specific array model.

See the Hardware Compatibility List for the complete list of supported arrays.

http://entsupport.symantec.com/docs/330441

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 94.

DMP can provide basic multi-pathing to ALUA-compliant arrays 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 Veritas Volume Manager 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 which are capable of being multipathed 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 94.

Disks in JBODs that do not fall into any supported category, and which are not capable of being multipathed by DMP are placed in the OTHER_DISKS category.

Adding 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 Symantec. Symantec 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 tech note. 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.

To access the hardware compatibility list, go to the following URL:

http://entsupport.symantec.com/docs/330441

The new disk array does not need to be already connected to the system when the VRTSaslapm rpm is installed. On a SLES11 system, if any of the disks in the new disk array are subsequently connected, and if vxconfigd is running, vxconfigd immediately invokes the Device Discovery function and includes the new disks in the VxVM device list. For other Linux flavors, reboot the system to make Linux recognize the new disks, and then use the vxdctl enable command to include the new disks in the VxVM device list.

See "Adding new LUNs dynamically to a new target ID" on page 214.

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 Volume Manager Troubleshooting Guide*.

Enabling discovery of new disk arrays

The `vxdctl enable` command scans all of the disk devices and their attributes, updates the VxVM 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
```

Third-party driver coexistence

The third-party driver (TPD) coexistence feature of VxVM allows I/O that is controlled by some third-party multi-pathing drivers to bypass DMP while retaining the monitoring capabilities of DMP. If a suitable 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. In previous releases, VxVM only supported TPD coexistence if the code of the third-party driver was intrusively modified. Now, the TPD coexistence feature maintains backward compatibility with such methods, but it also permits coexistence without requiring any change in a third-party multi-pathing driver.

See "Changing device naming for TPD-controlled enclosures" on page 102.

See "Displaying information about TPD-controlled devices" on page 181.

Autodiscovery of EMC Symmetrix arrays

In VxVM 4.0, there were two possible ways to configure EMC Symmetrix arrays:

- With EMC PowerPath installed, EMC Symmetrix arrays could be configured as foreign devices.
  See "Foreign devices" on page 98.
- Without EMC PowerPath installed, DMP could be used to perform multi-pathing.

On upgrading a system to VxVM 4.1 or later release, existing EMC PowerPath devices can be discovered by DDL, and configured into DMP as autoconfigured

disks with DMP nodes, even if PowerPath is being used to perform multi-pathing. There is no need to configure such arrays as foreign devices.

Table 3-1 shows the scenarios for using DMP with PowerPath.

The ASLs are all included in the ASL-APM rpm, which is installed when you install Storage Foundation products.

Table 3-1 Scenarios for using DMP with PowerPath

PowerPath	DMP	Array configuration mode
Installed.	The `libvxpp` ASL handles EMC Symmetrix arrays and DGC CLARiiON claiming internally. PowerPath handles failover.	EMC Symmetrix - Any DGC CLARiiON - Active/Passive (A/P), Active/Passive in Explicit Failover mode (A/P-F) and ALUA Explicit failover
Not installed; the array is EMC Symmetrix.	DMP handles multi-pathing. The ASL name is `libvxemc`.	Active/Active
Not installed; the array is DGC CLARiioN (CXn00).	DMP handles multi-pathing. The ASL name is `libvxCLARiiON`.	Active/Passive (A/P), Active/Passive in Explicit Failover mode (A/P-F) and ALUA

If any EMCpower disks are configured as foreign disks, use the `vxddladm rmforeign` command to remove the foreign definitions, as shown in this example:

```
# vxddladm rmforeign blockpath=/dev/emcpowera10 \
  charpath=/dev/emcpowera10
```

To allow DMP to receive correct inquiry data, the Common Serial Number (C-bit) Symmetrix Director parameter must be set to enabled.

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 VxVM 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 supported 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)
                 LUN 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)
                 LUN sdb
                 LUN 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 configured on the system, including iSCSI adapters. This includes the following information:

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

PortID   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. This includes the following information:

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:

```
TgtID      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

  TgtID      Alias  HBA-ID   State      Address
  -----------------------------------------------------------------
  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. This includes the following information:

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
  sdc       fscsi0_p0_t0  Offline  ERROR
  sdd       fscsi0_p0_t0  Online   EXCLUDED
  sde       fscsi0_p0_t0  Offline  MASKED
  ```

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 3-2 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
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. The following is a sample output:

  ```
  # vxddladm getiscsi target=c2_p2_t0
  ```

```
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

- Type the following command:

  ```
  # vxddladm listsupport all
  ```

Excluding support for a disk array library

To exclude support for a disk array library

- Type the following command:

  ```
  # vxddladm excludearray libname=libvxenc.so
  ```

  This example excludes support for disk arrays that depends on the library `libvxenc.so`. You can also exclude support for disk arrays from a particular vendor, as shown in this example:

  ```
  # vxddladm excludearray vid=ACME pid=X1
  ```

Re-including support for an excluded disk array library

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, as shown in the following example:

  ```
  # vxddladm includearray libname=libvxenc.so
  ```

Listing excluded disk arrays

To list all disk arrays that are currently excluded from use by VxVM

- Type the following command:

  ```
  # vxddladm listexclude
  ```

Listing supported disks in the DISKS category

To list disks that are supported in the `DISKS` (JBOD) category

- Type the following command:

  ```
  # vxddladm listjbod
  ```

Displaying details about a supported array library

To display details about a supported array library

- Type the following command:

  ```
  # vxddladm listsupport libname=library_name.so
  ```

  The Array Support Libraries are in the directory `/etc/vx/lib/discovery.d`.

Adding unsupported disk arrays to the DISKS category

Disk arrays should be added as JBOD devices if no 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 159.

Warning: This procedure ensures that Dynamic Multi-Pathing (DMP) is set up correctly on an array that is not supported by Veritas Volume Manager. Otherwise, Veritas Volume Manager 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 shows the output for the example disk with the device name

```
/dev/sdj# /etc/vx/diag.d/vxscsiinq /dev/sdj

Vendor id (VID)      : SEAGATE
Product id (PID)     : ST318404LSUN18G
Revision             : 8507
Serial Number        : 0025T0LA3H
```

In this example, the vendor ID is `SEAGATE` and the product ID is `ST318404LSUN18G`.

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 argument to 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 85.

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
============================================================
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
Disk_0   auto:none  -        -          online invalid
Disk_1   auto:none  -        -          online invalid
...
```

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
=====================================================
Disk_0 ENABLED Disk       2     2    0    Disk
Disk_1 ENABLED Disk       2     2    0    Disk
...
```

This 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

To remove disks from the `DISKS` category

- Use the `vxddladm` command with the `rmjbod` keyword. The following example illustrates the command for removing disks which have the vendor id of SEAGATE:

```
# vxddladm rmjbod vid=SEAGATE
```

Foreign devices

DDL may not be able to discover some devices that are controlled by third-party drivers, such as those that provide multi-pathing or RAM disk capabilities. For these devices it may be preferable to use the multi-pathing capability that is provided by the third-party drivers for some arrays rather than using Dynamic Multi-Pathing (DMP). Such foreign devices can be made available as simple disks to 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.

The foreign device feature was introduced in VxVM 4.0 to support non-standard devices such as RAM disks, some solid state disks, and pseudo-devices such as EMC PowerPath.

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.

If a suitable ASL is available and installed for an array, these limitations are removed.

See "Third-party driver coexistence" on page 85.

Disks under VxVM control

When you add a disk to a system that is running VxVM, you need to put the disk under VxVM control so that VxVM can control the space allocation on the disk.

See "Adding a disk to VxVM" on page 106.

Unless you specify a disk group, VxVM places new disks in a default disk group according to the rules for determining the default disk group.

See "Rules for determining the default disk group" on page 222.

The method by which you place a disk under VxVM control depends on the following circumstances:

- If the disk is new, it must be initialized and placed under VxVM control. You can use the menu-based `vxdiskadm` utility to do this.

  Warning: Initialization destroys existing data on disks.

- If the disk is not needed immediately, it can be initialized (but not added to a disk group) and reserved for future use. To do this, enter `none` when asked to name a disk group. Do not confuse this type of "spare disk" with a hot-relocation spare disk.
- If the disk was previously initialized for future use by VxVM, it can be reinitialized and placed under VxVM control.
- If the disk was previously in use by LVM, you can preserve existing data while still letting VxVM take control of the disk. This is accomplished using conversion. With conversion, the virtual layout of the data is fully converted to VxVM control.
  See the *Veritas Storage Foundation Advanced Features Administrator's Guide*.
- If the disk was previously in use, but not under VxVM control, you may wish to preserve existing data on the disk while still letting VxVM take control of the disk. This can be accomplished using encapsulation.
  Encapsulation preserves existing data on disks.
- Multiple disks on one or more controllers can be placed under VxVM control simultaneously. Depending on the circumstances, all of the disks may not be processed the same way.

It is possible to configure the `vxdiskadm` utility not to list certain disks or controllers as being available. For example, this may be useful in a SAN environment where disk enclosures are visible to a number of separate systems.

To exclude a device from the view of VxVM, select `Prevent multipathing/Suppress devices from VxVM's view` from the `vxdiskadm` main menu.

See "Disabling multi-pathing and making devices invisible to VxVM" on page 166.

Changing the disk-naming scheme

You can either use enclosure-based naming for disks or the operating system's naming scheme. VxVM commands display device names according to the current naming scheme.

The default naming scheme is enclosure-based naming (EBN). When you use DMP with native volumes, the disk naming scheme must be EBN, the use_avid attribute must be on, 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 VxVM to use. When prompted, enter `y` to change the naming scheme.

  Alternatively, you can change the naming scheme from the command line. Use the following command to select enclosure-based naming:

  ```
  # vxddladm set namingscheme=ebn [persistence={yes|no}] \
  [lowercase=yes|no] [use_avid=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 VxVM 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 lower case. 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 102.

Displaying the disk-naming scheme

VxVM disk naming can be operating-system based naming or enclosure-based naming. This command displays whether the VxVM 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
```

See "Disk device naming in VxVM" on page 77.

Regenerating persistent device names

The persistent device naming feature makes the names of disk devices persistent across system reboots. 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 OS-based and enclosure-based names are regenerated.

  The disk names now correspond to the new path names.

Changing device naming for TPD-controlled enclosures

By default, TPD-controlled enclosures 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 VxVM 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 101.

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 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 `EMC0`. In this example, the device-naming scheme is set to OSN.

  ```
  # vxdisk list

  DEVICE          TYPE             DISK       GROUP      STATUS
  emcpower10      auto:sliced      disk1      mydg       online
  emcpower11      auto:sliced      disk2      mydg       online
  emcpower12      auto:sliced      disk3      mydg       online
  emcpower13      auto:sliced      disk4      mydg       online
  emcpower14      auto:sliced      disk5      mydg       online
  emcpower15      auto:sliced      disk6      mydg       online
  emcpower16      auto:sliced      disk7      mydg       online
  emcpower17      auto:sliced      disk8      mydg       online
  emcpower18      auto:sliced      disk9      mydg       online
  emcpower19      auto:sliced      disk10     mydg       online

  # vxdmpadm setattr enclosure EMC0 tpdmode=native

  # vxdisk list

  DEVICE          TYPE             DISK       GROUP      STATUS
  sdj             auto:sliced      disk1      mydg       online
  sdk             auto:sliced      disk2      mydg       online
  sdl             auto:sliced      disk3      mydg       online
  sdm             auto:sliced      disk4      mydg       online
  sdn             auto:sliced      disk5      mydg       online
  sdo             auto:sliced      disk6      mydg       online
  sdp             auto:sliced      disk7      mydg       online
  sdq             auto:sliced      disk8      mydg       online
  sdr             auto:sliced      disk9      mydg       online
  sds             auto:sliced      disk10     mydg       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 meta-devices 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 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 VxVM 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) .

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*.

In a cluster environment, the DMP device names are the same across all nodes in the cluster.

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
```

Discovering the association between enclosure-based disk names and OS-based disk names

If you enable enclosure-based naming, the `vxprint` command displays the structure of a volume using enclosure-based disk device names (disk access names) rather than OS-based names.

To discover the association between enclosure-based disk names and OS-based disk names

- To discover the operating system-based names that are associated with a given enclosure-based disk name, use either of the following commands:

```
# vxdisk list enclosure-based_name
# vxdmpadm getsubpaths dmpnodename=enclosure-based_name
```

For example, to find the physical device that is associated with disk `ENC0_21`, the appropriate commands would be:

```
# vxdisk list ENC0_21
# vxdmpadm getsubpaths dmpnodename=ENC0_21
```

To obtain the full pathname for the block disk device and the character disk device from these commands, append the displayed device name to `/dev/vx/dmp` or `/dev/vx/rdmp`.

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 "Displaying or changing default disk layout attributes" on page 106.

See "Adding a disk to VxVM" on page 106.

Displaying or changing default disk layout attributes

To display or change the default values for initializing the layout of disks

- Select `Change/display the default disk layout` from the `vxdiskadm` main menu. For disk initialization, you can change the default format and the default length of the private region. The attribute settings for initializing disks are stored in the file, `/etc/default/vxdisk`.

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

  For disk encapsulation, you can additionally change the offset values for both the private and public regions. The attribute settings for encapsulating disks are stored in the file, `/etc/default/vxencap`.

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

Adding a disk to VxVM

Formatted disks being placed under 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 "Disabling multi-pathing and making devices invisible to VxVM" on page 166.

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: n)
```

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 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)
```

The default layout for disks can be changed.

See "Displaying or changing default disk layout attributes" on page 106.

Disk reinitialization

You can reinitialize a disk that has previously been initialized for use by VxVM by putting it under VxVM control as you would a new disk.

See "Adding a disk to VxVM" on page 106.

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 Storage Foundation Advanced Features Administrator's Guide*.

Using vxdiskadd to put a disk under VxVM control

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 116.

  If you are adding an uninitialized disk, warning and error messages are displayed on the console by the `vxdiskadd` command. 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 106.

RAM disk support in VxVM

Some systems support the creation of RAM disks. A RAM disk is a device made from system memory that looks like a small disk device. Often, the contents of a RAM disk are erased when the system is rebooted. RAM disks that are erased on reboot prevent VxVM from identifying physical disks. This is because information stored on the physical disks (now erased on reboot) is used to identify the disk.

`nopriv` devices have a special feature to support RAM disks: a volatile option which indicates to VxVM that the device contents do not survive reboots. Volatile devices receive special treatment on system startup. If a volume is mirrored, plexes made from volatile devices are always recovered by copying data from nonvolatile plexes.

To use a RAM disk with VxVM, both block and character device nodes must exist for the RAM disk.

To create a RAM disk with VxVM

1. Block RAM devices are already present. Select a block RAM device such as `/dev/raw/raw6` and create a raw device. For example:

   ```
   # mknod /dev/raw/raw6 c 162 26
   ```

2. Bind the character device to the block device. For example:

   ```
   # raw /dev/raw/raw6 /dev/ram6
   ```

3. Add the RAM disk as a foreign device to VxVM using the `vxddladm addforeign` command. For example:

   ```
   # vxddladm addforeign blockpath=/dev/ram6 charpath=/dev/raw/raw6
   ```

4. Discover the RAM disk using the `vxdisk scandisks` command:

   ```
   # vxdisk scandisks
   ```

Normally, VxVM does not start volumes that are formed entirely from plexes with volatile subdisks. That is because there is no plex that is guaranteed to contain the most recent volume contents.

Some RAM disks are used in situations where all volume contents are recreated after reboot. In these situations, you can force volumes formed from RAM disks to be started at reboot by using the following command:

```
# vxvol set startopts=norecov volume
```

This option can be used only with volumes of type `gen`.

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

Encapsulating a disk

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 122.

See "Rootability" on page 122.

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 and tells you that a reboot is needed:

```
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.

See "Displaying or changing default disk layout attributes" on page 106.

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 120.

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`.

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 Volume Manager 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:

- Sample supported root disk layouts for encapsulation
- Sample unsupported root disk layouts for encapsulation

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 `gpt` labels. It is not supported for disks with `sun` 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. Multipathing configuration changes (for multiple HBA systems) can have the same effect. VxVM supports only 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 5" on page 127.
- 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.

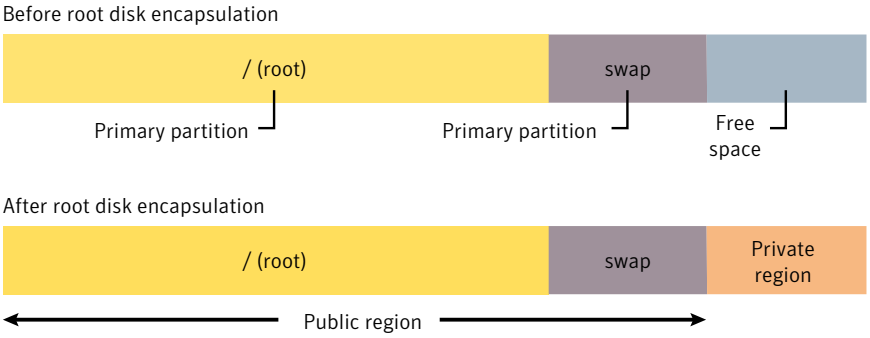
Sample supported root disk layouts for encapsulation

The following examples show root disk layouts that support encapsulation.

Example 1

Figure 3-1 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 3-1 Root and swap configured on two primary partitions, and free space on the disk

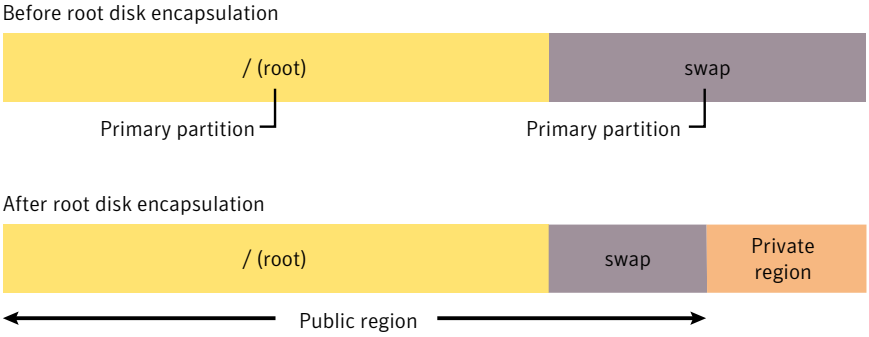


Two primary partitions are in use by `/` and `swap`. There are two unused primary partitions, and free space exists on the disk that can be assigned to a primary partition for the private region.

Example 2

Figure 3-2 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 3-2 Root and swap configured on two primary partitions, and no free space

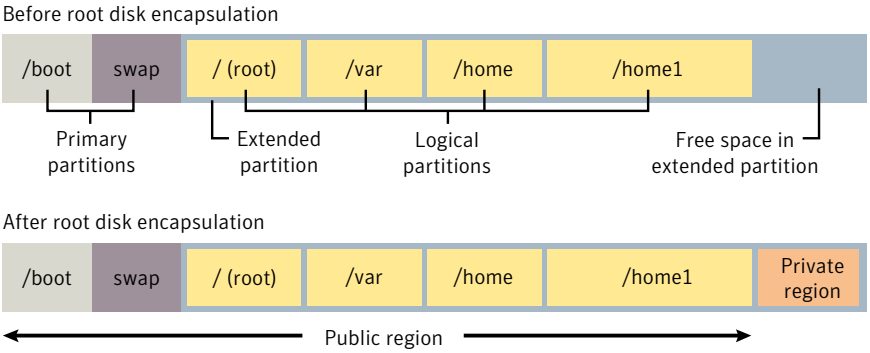


Two primary partitions are in use by `/` and `swap`. There are two unused primary partitions, and the private region can be allocated to a new primary partition by taking space from the end of the swap partition.

Example 3

Figure 3-3 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 3-3 Boot and swap configured on two primary partitions, and free space in the extended partition

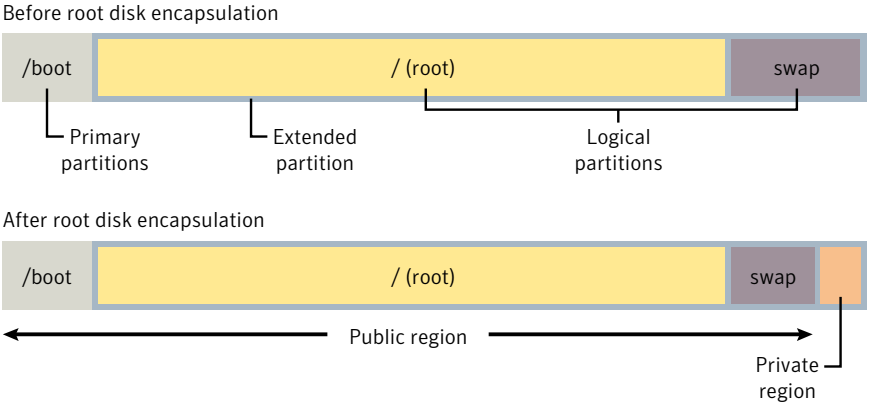


Three primary partitions are in use by `/boot`, `swap` and an extended partition that contains four file systems including `root`. There is free space at the end of the extended primary partition that can be used to create a new logical partition for the private region.

Example 4

Figure 3-4 shows an example of a supported layout with `boot` configured on a primary partition, and `root` and `swap` configured in the extended partition.

Figure 3-4 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.

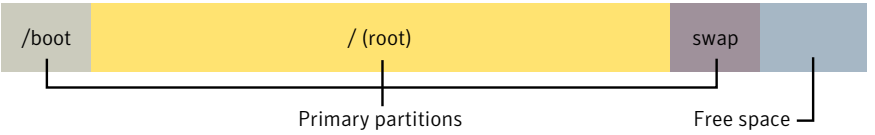
Sample unsupported root disk layouts for encapsulation

The following examples show root disk layouts that do not support encapsulation.

Example 5

Figure 3-5 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.

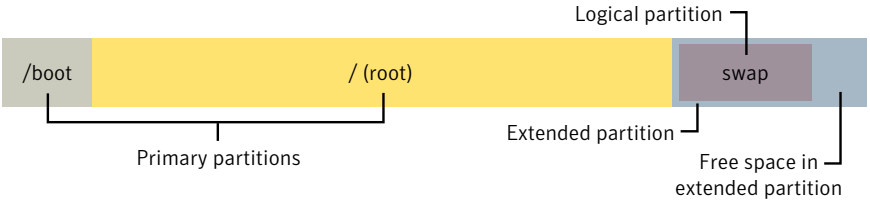
Figure 3-5 Boot, swap and root configured on three primary partitions, and free space on the disk



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 3-6 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 3-6 Workaround by reconfiguring swap as a logical partition

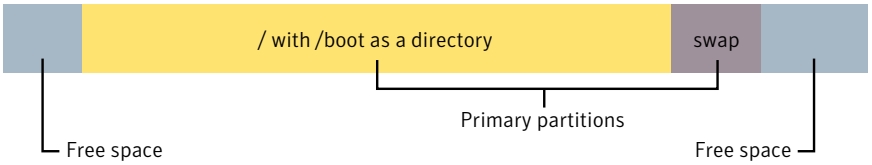


The original swap partition should be deleted. After reconfiguration, this root disk can be encapsulated.

See "Example 3" on page 126.

Figure 3-7 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 3-7 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.

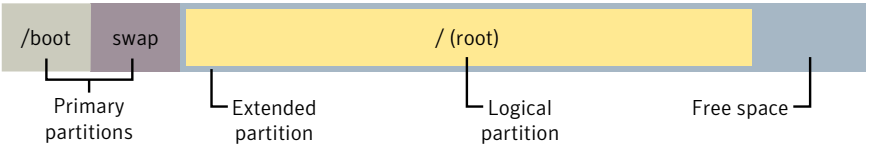
After reconfiguration, this root disk can be encapsulated.

See "Example 1" on page 125.

Example 6

Figure 3-8 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.

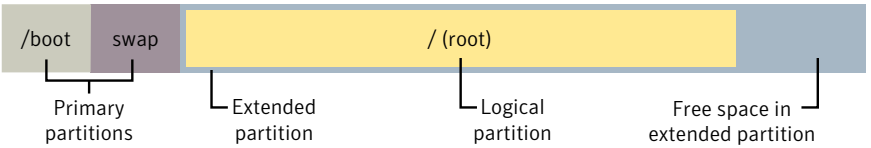
Figure 3-8 Boot and swap configured on two primary partitions, and no free space in the extended partition



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 3-9 shows a simple workaround that uses a partition configuration tool to grow the extended partition into the free space on the disk.

Figure 3-9 Workaround by growing the extended partition



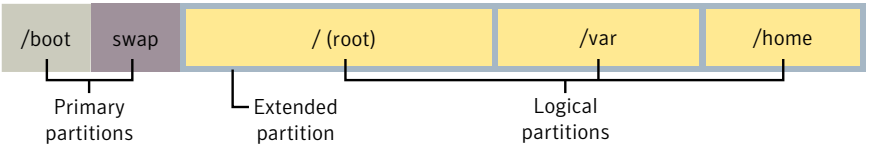
Care should be taken to preserve the boundaries of the logical partition that contains the `root` file system. After reconfiguration, this root disk can be encapsulated.

See "Example 3" on page 126.

Example 7

Figure 3-10 shows an example of an unsupported layout with `boot` and `swap` configured on two primary partitions, and no existing free space on the disk.

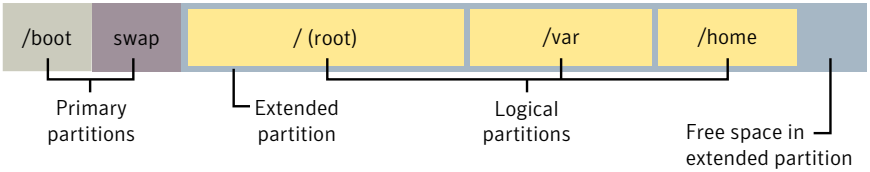
Figure 3-10 Boot and swap configured on two primary partitions, and no free space



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 3-11 shows a possible workaround by shrinking one or more of the existing file systems and the corresponding logical partitions.

Figure 3-11 Workaround by shrinking existing logical partitions



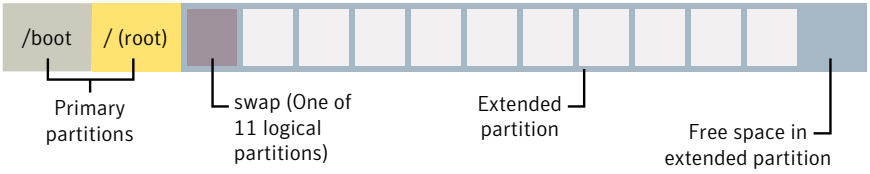
Shrinking existing logical partitions frees up space in the extended partition for the private region. After reconfiguration, this root disk can be encapsulated.

See "Example 3" on page 126.

Example 8

Figure 3-12 shows an example of an unsupported layout with `boot` and `root` configured on two primary partitions, and no more available logical partitions.

Figure 3-12 Boot and swap configured on two primary partitions, and no more available logical partitions



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" on page 126.

See "Example 4" on page 127.

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 Volume Manager Troubleshooting Guide*.

See "Restrictions on using rootability with Linux" on page 122.

You can use the `vxdiskadm` command to encapsulate the root disk.

See "Encapsulating a disk" on page 116.

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

- Choose a disk 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).

  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.

  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 141.

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 132.

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 snapshot of an encapsulated boot disk.

`vxrootadm` 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.

Creating a snapshot of an encapsulated boot disk

When you create a snapshot of an encapsulated boot disk, the `vxrootadm` command has the following format:

`vxrootadm -s srcdisk [-g dg] mksnap` *`destdisk newdg`*

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 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 102.

Displaying disk information

Before you use a disk, you need to know if it has been initialized and placed under 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

- Type the following command:

```
# vxdisk list
```

VxVM returns a display similar to the following:

```
DEVICE       TYPE            DISK         GROUP        STATUS
sdb          auto:sliced     mydg04       mydg         online
sdc          auto:sliced     mydg03       mydg         online
sdd          auto:sliced     -            -            online invalid
sde          auto:sliced     -            -            online thinrclm
```

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

- Type 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. Without this option, no tags are displayed.

Displaying disk information with vxdiskadm

Displaying disk information shows you which disks are initialized, to which disk groups they belong, and the disk status. The `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 disk information

1. Start the `vxdiskadm` program, and select `list (List disk information)` from the main menu.
2. At the following display, enter the address of the disk you want to see, or enter `all` for a list of all disks:

```
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.
- If you enter the address of the device for which you want information, complete disk information (including the device name, the type of disk, and information about the public and private areas of the disk) is displayed.

Once you have examined this information, press Return to return to the main menu.

Removing disks

You must disable a disk group before you can remove the last disk in that group.

See "Disabling a disk group" on page 275.

As an alternative to disabling the disk group, you can destroy the disk group.

See "Destroying a disk group" on page 276.

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 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 383.

See "Taking plexes offline" on page 299.

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

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`.

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.

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 Volume Manager 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 150.

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 428.

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)
```

Enabling a disk

If you move a disk from one system to another during normal system operation, VxVM does not recognize the disk automatically. The enable disk task enables VxVM to identify the disk and to determine if this disk is part of a disk group. Also, this task re-enables access to a disk that was disabled by either the disk group deport task or the disk device disable (offline) task.

To enable a disk

1 Select `Enable (online) a disk device` from the `vxdiskadm` main menu.

2 At the following prompt, enter the device name of the disk to be enabled (or enter list for a `list` of devices):

```
Select a disk device to enable [<address>,list,q,?]
sdc
```

`vxdiskadm` enables the specified device.

3 At the following prompt, indicate whether you want to enable another device (`y`) or return to the `vxdiskadm` main menu (`n`):

```
Enable another device? [y,n,q,?] (default: n)
```

Taking a disk offline

There are instances when you must take a disk offline. If a disk is corrupted, you must disable the disk before removing it. You must also disable a disk before moving the physical disk device to another location to be connected to another system.

Warning: Taking a disk offline is only useful on systems that support hot-swap removal and insertion of disks. If a system does not support hot-swap removal and insertion of disks, you must shut down the system.

To take a disk offline

1. Select `Disable (offline) a disk device` from the `vxdiskadm` main menu.
2. At the following prompt, enter the address of the disk you want to disable:

   ```
   Select a disk device to disable [<address>,list,q,?]
   sdc
   ```

   The `vxdiskadm` program disables the specified disk.
3. At the following prompt, indicate whether you want to disable another device (`y`) or return to the `vxdiskadm` main menu (`n`):

   ```
   Disable another device? [y,n,q,?] (default: n)
   ```

Renaming a disk

If you do not specify a VM disk name, VxVM gives the disk a default name when you add the disk to VxVM control. The VM disk name is used by VxVM 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 VM disk on which they are located. Renaming a VM 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
  ```

Reserving disks

By default, the `vxassist` command allocates space from any disk that has free space. You can reserve a set of disks for special purposes, such as to avoid general use of a particularly slow or a particularly fast disk.

To reserve a disk

- Type the following command:

  ```
  # vxedit [-g diskgroup] set reserve=on diskname
  ```

  After you enter this command, the `vxassist` program does not allocate space from the selected disk unless that disk is specifically mentioned on the `vxassist` command line. For example, if mydg03 is reserved, use the following command:

  ```
  # vxassist [-g diskgroup] make vol03 20m mydg03
  ```

  The `vxassist` command overrides the reservation and creates a 20 megabyte volume on mydg03. However, this command does not use mydg03, even if there is no free space on any other disk:

  ```
  # vxassist -g mydg make vol04 20m
  ```

To turn off reservation of a disk

- Type the following command:

  ```
  # vxedit [-g diskgroup] set reserve=off diskname
  ```

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

Extended Copy Service

The Extended Copy Service feature of VxVM works in tandem with the extended copy engines from array vendors. When VxVM detects that the source and destination devices are enabled for extended copy, VxVM automatically off loads copy requests to the array's copy manager.

The benefits of the Extended Copy Service are:

- Non-disruptive copy operations from disk to disk. The host server remains online during the copy and the data being copied remains accessible to the server.
- Server-free copy operation. The copy operation is done between the array subsystem and the target disk. The data copy operation does not use any CPU or I/O resources on the host server.

To see whether the Extended Copy Service feature is enabled on a disk, use the `vxprint` command as shown in the following example. The feature is enabled if an `ecopy_enabled` entry appears in the `flags` line.

```
# vxprint -l tagmastore-usp0_1b6f
Disk group: privatedg5

Disk:     tagmastore-usp0_1b6f
info:     diskid=1246621818.714.swlx59.vxindia.veritas.com
assoc:    device=tagmastore-usp0_1b6f type=auto
flags:    autoconfig
device:   path=/dev/vx/dmp/tagmastore-usp0_1b6fs3
devinfo:  publen=2023168 privlen=65536
```

If required, you can use the `-o noecopy` option to turn off Extended Copy Service for each invocation of the `vxplex att`, `cp`, `mv` and `snapstart` commands, and the `vxsd mv` command.

Enabling a disk for Extended Copy Service operation

To enable a disk for Extended Copy Service operation, perform the following tasks in the order listed:

- Install the Hardware Assisted Copy license.
- Enable the Ecopy features in the array. This procedure is vendor-specific.
- Install the vendor ASL that supports the Ecopy feature. contact VITA@veritas.com for vendor ASL information.

Enabling Extended Copy Service for Hitachi arrays

To enable extended copy service for the Hitachi 9900 and 9900V arrays

- Use the following command to create the two files, `/etc/vx/user_pwwn_file` and `/etc/vx/user_luid_file`, that contain identification information for the disks.

  ```
  # /etc/vx/diag.d/vxwwnluid
  ```

  This command must be executed as `root`.

  The `user_pwwn_file` file contains the disk access name and the port world-wide-name (pwwn) for each disk in the array. For the Hitachi arrays, both the source and the destination devices must have entries in the this file. The information for each disk in the array is defined on a single line. The disk access name and PWWN are separated by a single tab character.

  The following are sample entries from the `user_pwwn_file` file:

  ```
  sde     50060e800404040b
  sdf     50060e800404040b
  sdg     50060e800404040b
  ```

  The `user_luid_file` file contains the disk access names and their corresponding LUN numbers in the array. The information for each disk in the array is defined on a single line. The disk access name and the LUN are separated by a single tab character.

  The following are sample entries from the `user_luid_file` file:

  ```
  sde     1
  sdf     2
  sdg     1
  ```

Chapter 4

Administering Dynamic Multi-Pathing

This chapter includes the following topics:

How DMP works

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

Multiported 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 81.

The multi-pathing policy that is used by DMP depends on the characteristics of the disk array.

DMP supports the following standard array types:

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. Usually an A/A-A array behaves like an A/P array rather than an A/A array. However, during failover, an A/A-A array behaves like an A/A array. An ALUA array behaves like an A/A-A array.
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 policy 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.

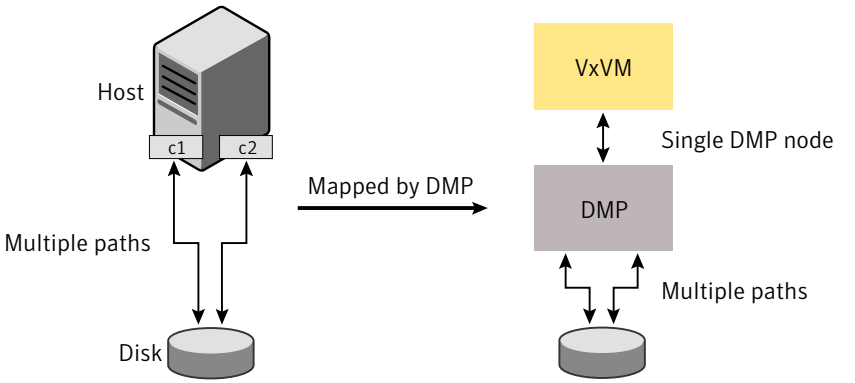
Active/Passive in explicit failover mode or non-autotrespass mode (A/P-F)	The appropriate command must be issued to the array to make the LUNs fail over to the secondary path. This policy supports concurrent I/O and load balancing by having multiple primary paths into a controller. This functionality 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/P-G)	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 policy 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.

VxVM 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 4-1 shows how DMP sets up a node for a disk in a supported disk array.

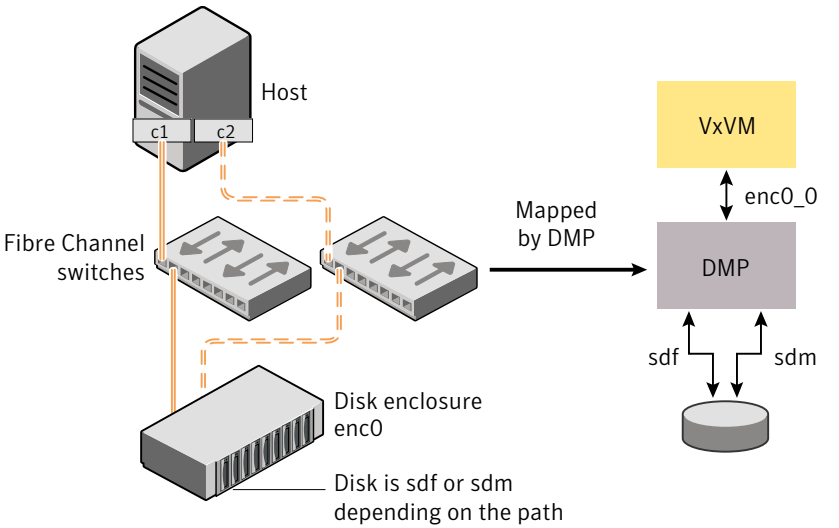
Figure 4-1 How DMP represents multiple physical paths to a disk as one node



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

Figure 4-2 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 4-2 Example of multi-pathing for a disk enclosure in a SAN environment



See "About enclosure-based naming" on page 24.

See "Changing the disk-naming scheme" on page 100.

See "Discovering and configuring newly added disk devices" on page 81.

How DMP monitors I/O on paths

In older releases of VxVM, 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 `vxdmpadm stat` command can be used to provide information about the threads. The names `errord` and `restored` have 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 (typically 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 207.

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 163.

See "I/O throttling" on page 164.

Path failover mechanism

DMP enhances system reliability when used with multiported disk arrays. 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 202.

Subpaths Failover Group (SFG)

An 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 group, 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 IO performance. Currently the criteria followed by DMP to form the subpath 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 205.

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 though IO activity is not there on the paths. 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 limited number of paths within an 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" on page 205.

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 204.

Load balancing

By default, the DMP uses the Minimum Queue policy for load balancing across paths for Active/Active (A/A), Active/Passive (A/P), Active/Passive with explicit failover (A/P-F) and Active/Passive with group failover (A/P-G) disk arrays. Load balancing maximizes I/O throughput by using the total bandwidth of all available paths. I/O is sent down the path which has the minimum outstanding I/Os.

For A/P disk arrays, I/O is sent down the primary paths. If 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 A/P, A/P-F and A/P-G arrays, load balancing is performed across all the currently active paths as is done for A/A arrays.

You can use the `vxdmpadm` command to change the I/O policy for the paths to an enclosure or disk array.

See "Specifying the I/O policy" on page 194.

DMP in a clustered environment

Note: You need an additional license to use the cluster feature of VxVM.

Clustering is only supported for VxVM.

In a clustered environment where Active/Passive type disk arrays are shared by multiple hosts, all nodes in the cluster must access the disk via the same physical storage controller port. Accessing a disk via 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 ASL (and APM, if required) is available for the array, and has been installed on the system.

See "Discovering disks and dynamically adding disk arrays" on page 83.

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 all need to access a disk via the same physical path.

See "How to administer the Device Discovery Layer" on page 86.

See "Configuring array policy modules" on page 209.

About enabling or disabling controllers with shared disk groups

Prior to release 5.0, 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.

Disabling multi-pathing and making devices invisible to VxVM

Use this procedure to prevent a device from being multi-pathed by the VxVM DMP driver (`vxdmp`), or to exclude a device from the view of VxVM.

To disable multi-pathing and 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 but one path to a disk. Only one path is made visible to VxVM.
Option 5	Prevents multi-pathing for all disks on a specified controller by VxVM.
Option 6	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.
Option 7	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.
Option 8	Lists the devices that are currently suppressed or not multi-pathed.

Enabling multi-pathing and making devices visible to VxVM

Use this procedure to re-enable multi-pathing for a device, or to make a device visible to VxVM again.

To enable multi-pathing and 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	Removes a pathgroup definition. (A pathgroup explicitly defines alternate paths to the same disk.) Once a pathgroup has been removed, all paths that were defined in that pathgroup become visible again.
Option 5	Allows multi-pathing of all disks that have paths through the specified controller.
Option 6	Allows multi-pathing of a disk by VxVM.
Option 7	Allows multi-pathing of disks that match a specified Vendor ID and Product ID combination.
Option 8	Lists the devices that are currently suppressed or not multipathed.

About enabling and disabling I/O for controllers and storage processors

DMP allows you to turn off I/O for a controller or the array port of a storage processor so that you can perform administrative operations. This feature can be used for maintenance of HBA controllers on the host, or array ports that are attached to disk arrays supported by VxVM. I/O operations to the controller or 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, after disabling 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 active secondary paths and I/O will continue on them.

After the operation is over, you can use `vxdmpadm` to re-enable the paths through the controllers.

See "Disabling I/O for paths, controllers or array ports" on page 200.

See "Enabling I/O for paths, controllers or array ports" on page 201.

Note: From release 5.0 of VxVM, these operations are supported for controllers that are used to access disk arrays on which cluster-shareable disk groups are configured.

You can also perform certain reconfiguration operations dynamically online.

See "About online dynamic reconfiguration" on page 211.

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 vxdmpadm" on page 173.

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 particular physical disk having multiple physical paths from one of the system's HBA controllers. In 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:    system01
.
.
.
Multipathing information:
numpaths:   2
sdl    state=enabled     type=secondary
sdp    state=disabled    type=primary
```

The `numpaths` line shows that there are 2 paths to the device. The next two lines in the "Multipathing information" section 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_893`, use the following command:

```
# vxdmpadm getsubpaths dmpnodename=emc_clariion0_893
```

Typical output from the `vxdmpadm getsubpaths` command is as follows:

```
NAME      STATE[A]    PATH-TYPE[M] CTLR-NAME  ENCLR-TYPE   ENCLR-NAME     ATTRS
==========================================================================
sdbc      ENABLED(A)  PRIMARY      c3         EMC_CLARiiON  emc_clariion0     -
sdbm      ENABLED     SECONDARY    c3         EMC_CLARiiON  emc_clariion0     -
sdbw      ENABLED(A)  PRIMARY      c3         EMC_CLARiiON  emc_clariion0     -
sdck      ENABLED(A)  PRIMARY      c2         EMC_CLARiiON  emc_clariion0     -
sdcu      ENABLED     SECONDARY    c2         EMC_CLARiiON  emc_clariion0     -
sdde      ENABLED(A)  PRIMARY      c2         EMC_CLARiiON  emc_clariion0     -
```

Setting customized names for DMP nodes

The DMP node name is the meta device name which represents the multiple paths to a disk. The DMP node name is generated from the device name according to the VxVM naming scheme.

See "Disk device naming in VxVM" on page 77.

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 assign DMP nodes from a file

1 Use the script `vxgetdmpnames` to get a sample file populated from the devices in your configuration. The sample file shows the format required and serves as a template to specify your customized names.

2 To assign the names, use the following command:

```
# vxddladm assign names file=pathname
```

To clear custom names

- To clear the names, and use the default OSN or EBN names, use the following command:

```
# vxddladm -c assign names
```

Administering DMP using vxdmpadm

The `vxdmpadm` utility is a command line administrative interface to DMP.

You can use the `vxdmpadm` utility to perform the following tasks:

- Retrieve the name of the DMP device corresponding to a particular path.
- Display the members of a LUN group.
- List all paths under a DMP device node, HBA controller or array port.
- Display information about the HBA controllers on the host.
- Display information about enclosures.
- Display information about array ports that are connected to the storage processors of enclosures.
- Display information about devices that are controlled by third-party multi-pathing drivers.
- Gather I/O statistics for a DMP node, enclosure, path or controller.
- Configure the attributes of the paths to an enclosure.
- Set the I/O policy that is used for the paths to an enclosure.
- Enable or disable I/O for a path, HBA controller or array port on the system.
- Upgrade disk controller firmware.

- Rename an enclosure.
- Configure how DMP responds to I/O request failures.
- Configure the I/O throttling mechanism.
- Control the operation of the DMP path restoration thread.
- Get or set the values of various tunables used by DMP.

The following sections cover these tasks in detail along with sample output.

See "Changing the values of VxVM tunables" on page 507.

See "DMP tunable parameters" on page 516.

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

Retrieving information about a DMP node

The following command displays the DMP node that controls a particular physical path:

```
# vxdmpadm getdmpnode nodename=sdbc
```

The physical path is specified by argument to the `nodename` attribute, which must be a valid path listed in the `/dev` directory.

The command displays output similar to the following:

```
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=enc0
```

```
NAME      STATE    ENCLR-TYPE PATHS  ENBL  DSBL ENCLR-NAME
=============================================================
sdm       ENABLED  ACME       2      2     0    enc0
sdn       ENABLED  ACME       2      2     0    enc0
```

```
sdo       ENABLED  ACME        2       2      0     enc0
sdp       ENABLED  ACME        2       2      0     enc0
```

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 DMP node. The information includes the enclosure name, LUN serial number, port id information, device attributes, etc.

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=enclosure name
```

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 dmpnode.

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 DMP nodes that are in the same LUN group as a specified DMP node:

```
# 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 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 WWN. You can also display paths for devices controlled by third-party drivers.

See "Displaying information about TPD-controlled devices" on page 181.

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)    -          disk_24      disk          c0        -
sda       ENABLED(A)    -          disk_25      disk          c0        -
sdav      ENABLED(A)  PRIMARY      emc_clariion0_1017 emc_clariion0 c3       -
sdbf      ENABLED     SECONDARY    emc_clariion0_1017 emc_clariion0 c3       -
```

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/rdmp` directory:

```
# vxdmpadm getsubpaths dmpnodename=sdu

NAME     STATE[A]   PATH-TYPE[M] CTLR-NAME ENCLR-TYPE ENCLR-NAME ATTRS
=====================================================================
sdu      ENABLED(A) PRIMARY      c2        ACME       enc0       -
sdt      ENABLED    PRIMARY      c1        ACME       enc0       -
```

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 194.

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=c2

NAME  STATE[-]   PATH-TYPE[-] CTLR-NAME ENCLR-TYPE ENCLR-NAME ATTRS
==========================================================================
sdk   ENABLED(A) PRIMARY      sdk       ACME       enc0       -
sdl   ENABLED(A) PRIMARY      sdl       ACME       enc0       -
sdm   DISABLED   SECONDARY    sdm       ACME       enc0       -
sdn   ENABLED    SECONDARY    sdn       ACME       enc0       -
```

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 worldwide name (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=A5

NAME         STATE[A]   PATH-TYPE[M] DMPNODENAME  ENCLR-NAME   CTLR    ATTRS
================================================================================
sdav         ENABLED(A) PRIMARY      emc_clariion0_1017 emc_clariion0 c3        -
sdcd         ENABLED(A) PRIMARY      emc_clariion0_1017 emc_clariion0 c2        -
sdau         ENABLED(A) PRIMARY      emc_clariion0_1018 emc_clariion0 c3        -
sdcc         ENABLED(A) PRIMARY      emc_clariion0_1018 emc_clariion0 c2        -
```

For example, to list subpaths through an array port through the WWN:

```
# vxdmpadm getsubpaths pwwn=50:06:01:61:41:e0:3b:33

NAME         STATE[A]   PATH-TYPE[M] CTLR-NAME  ENCLR-TYPE   ENCLR-NAME    ATTRS
================================================================================
sdav         ENABLED(A)  PRIMARY      c3         EMC_CLARiiON  emc_clariion0     -
sdcd         ENABLED(A)  PRIMARY      c2         EMC_CLARiiON  emc_clariion0     -
sdau         ENABLED(A)  PRIMARY      c3         EMC_CLARiiON  emc_clariion0     -
sdcc         ENABLED(A)  PRIMARY      c2         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=Disk ctlr=c1
```

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]
```

Displaying information about controllers

The following command lists attributes of all HBA controllers on the system:

```
# vxdmpadm listctlr all
```

```
CTLR-NAME       ENCLR-TYPE        STATE          ENCLR-NAME
=============================================================
c1              OTHER             ENABLED        other0
c2              X1                ENABLED        jbod0
c3              ACME              ENABLED        enc0
c4              ACME              ENABLED        enc0
```

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=enc0
```

or

```
# vxdmpadm listctlr type=ACME
```

```
CTLR-NAME          ENCLR-TYPE           STATE         ENCLR-NAME
=============================================================
c2                 ACME                 ENABLED       enc0
c3                 ACME                 ENABLED       enc0
```

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 occurance 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

To display the attributes of a specified enclosure, including its enclosure type, enclosure serial number, status, array type, and number of LUNs, use the following command:

```
# vxdmpadm listenclosure enc0
```

```
ENCLR_NAME ENCLR_TYPE  ENCLR_SNO            STATUS  ARRAY_TYPE    LUN_COUNT
=========================================================================
enc0       A3          60020f20000001a90000 CONNECTED  A/P        30
```

The following command lists attributes for all enclosures in a system:

```
# vxdmpadm listenclosure all
```

```
ENCLR_NAME ENCLR_TYPE ENCLR_SNO             STATUS    ARRAY_TYPE    LUN_COUNT
==========================================================================
Disk       Disk       DISKS                 CONNECTED     Disk        6
ANA0       ACME       508002000001d660      CONNECTED     A/A        57
enc0       A3         60020f20000001a90000 CONNECTED     A/P        30
```

Displaying information about array ports

Use the commands in this section to display information about array ports. The information displayed for an array port includes the name of its enclosure, and its ID and worldwide name (WWN) identifier.

To display the attributes of an array port that is accessible via 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 via 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 TPD-controlled devices

The third-party driver (TPD) coexistence feature allows I/O that is controlled by third-party multi-pathing drivers to bypass 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 TPD-controlled enclosures" on page 102.

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
emcpower10      auto:sliced     disk1       ppdg      online
emcpower11      auto:sliced     disk2       ppdg      online
emcpower12      auto:sliced     disk3       ppdg      online
emcpower13      auto:sliced     disk4       ppdg      online
emcpower14      auto:sliced     disk5       ppdg      online
emcpower15      auto:sliced     disk6       ppdg      online
emcpower16      auto:sliced     disk7       ppdg      online
emcpower17      auto:sliced     disk8       ppdg      online
emcpower18      auto:sliced     disk9       ppdg      online
emcpower19      auto:sliced     disk10      ppdg      online
```

The following command displays the paths that DMP has discovered, and which correspond to the PowerPath-controlled node, `emcpower10`:

```
# vxdmpadm getsubpaths tpdnodename=emcpower10

NAME     TPDNODENAME  PATH-TYPE[-]DMP-NODENAME  ENCLR-TYPE ENCLR-NAME
=================================================================
sdq      emcpower10s2 -           emcpower10    PP_EMC     pp_emc0
sdr      emcpower10s2 -           emcpower10    PP_EMC     pp_emc0
```

Conversely, the next command displays information about the PowerPath node that corresponds to the path, `sdq`, discovered by DMP:

```
# vxdmpadm gettpdnode nodename=sdq

NAME          STATE    PATHS  ENCLR-TYPE    ENCLR-NAME
=================================================================
emcpower10s2 ENABLED  2      PP_EMC        pp_emc0
```

Displaying extended device attributes

Device Discovery Layer (DDL) extended attributes are attributes or flags corresponding to a VxVM or DMP LUN or Disk and which are discovered by DDL. These attributes identify a LUN to a specific hardware category.

The list of categories includes:

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 disk)
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 attributes discovered during device discovery. ASLs furnish this information to DDL through the property DDL_DEVICE_ATTR. 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 via 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               VID          DDL_DEVICE_ATTR
tagmastore-usp0_0a7a HITACHI      std fc RAID_5
tagmastore-usp0_0a7b HITACHI      std fc RAID_5
tagmastore-usp0_0a78 HITACHI      std fc RAID_5
tagmastore-usp0_0a79 HITACHI      std fc RAID_5
tagmastore-usp0_065a HITACHI      hdprclm fc
tagmastore-usp0_065b HITACHI      hdprclm fc
tagmastore-usp0_065c HITACHI      hdprclm fc
tagmastore-usp0_065d HITACHI      hdprclm fc
```

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 at the following URL:

http://seer.entsupport.symantec.com/docs/330441.htm

Suppressing or including devices for VxVM or DMP control

The `vxdmpadm exclude` command suppresses devices from VxVM or DMP based on the criteria that you specify. The devices can be added back into VxVM or DMP

control by using 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 [vxvm | vxdmp] { all | product=VID:PID |
ctlr=[\!]ctlr | dmpnodename=diskname [ path=\!pathname] }

# vxdmpadm include [vxvm | vxdmp] { all | product=VID:PID |
ctlr=[\!]ctlr | dmpnodename=diskname [ path=\!pathname] }
```

where:

all – all devices

product=*VID:PID* – all devices with the specified VID:PID

ctlr=*ctlr* – 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 or controller.

To enable the gathering of statistics, enter this command:

```
# vxdmpadm iostat start [memory=size]
```

To reset the I/O counters to zero, use this command:

```
# vxdmpadm iostat reset
```

The `memory` attribute can be used to limit 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 display the accumulated statistics at regular intervals, use the following command:

```
# vxdmpadm iostat show {all | ctlr=ctlr-name \
  | dmpnodename=dmp-node  \
  | enclosure=enclr-name [portid=array-portid ] \
  | pathname=path-name |  pwwn=array-port-wwn } \
  [interval=seconds [count=N]]
```

This command displays I/O statistics for all paths (`all`), or for a specified controller, DMP node, enclosure, path or port ID. 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.

The `interval` and `count` attributes may be used 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.

To disable the gathering of statistics, enter this command:

```
# vxdmpadm iostat stop
```

Examples of using the vxdmpadm iostat command

The following is 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 kilobytes read and written, on all paths.

```
# vxdmpadm iostat show all
                    cpu usage = 7952us    per cpu memory = 8192b
             OPERATIONS           KBYTES         AVG TIME(ms)
PATHNAME READS     WRITES     READS     WRITES    READS    WRITES
sdf          87          0     44544          0    0.00      0.00
sdk           0          0         0          0    0.00      0.00
sdg          87          0     44544          0    0.00      0.00
sdl           0          0         0          0    0.00      0.00
sdh          87          0     44544          0    0.00      0.00
sdm           0          0         0          0    0.00      0.00
sdi          87          0     44544          0    0.00      0.00
sdn           0          0         0          0    0.00      0.00
```

```
sdj          87           0     44544           0     0.00      0.00
sdo           0           0         0           0     0.00      0.00
sdj          87           0     44544           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 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 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     557056          0      0.00      0.00

# vxdmpadm 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     557056          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
```

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 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
```

Displaying cumulative I/O statistics

Use the `groupby` clause of the `vxdmpadm iostat` command to display cumulative I/O statistics listings per DMP node, controller, array port id, or host-array controller pair and enclosure. If the `groupby` clause is not specified, then the statistics are displayed per path.

To group by DMP node:

```
# vxdmpadm iostat show groupby=dmpnode [all | dmpnodename=dmpnodename
| enclosure=enclr-name]
```

To group by controller:

```
# vxdmpadm 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 iostat show groupby=arrayport [ all | pwwn=array_pwwn
 | enclosure=enclr portid=array-port-id ]
```

For example:

```
# vxdmpadm iostat show groupby=arrayport enclosure=HDS9500-ALUA0 \
portid=1A
```

```
             OPERATIONS        BLOCKS       AVG TIME(ms)
PORTNAME  READS   WRITES  READS  WRITES  READS   WRITES
1A        224      14     54      7       4.20    11.10
```

To group by enclosure:

```
# vxdmpadm iostat show groupby=enclosure [ all | enclosure=enclr ]
```

For example:

```
# vxdmpadm iostat show groupby=enclosure enclosure=EMC_CLARiiON0
```

```
               OPERATIONS        BLOCKS       AVG TIME(ms)
ENCLRNAME    READS  WRITES  READS  WRITES  READS   WRITES
EMC_CLARiiON 0      0       0      0        0.00     0.00
```

You can also filter out entities for which all data entries are zero. This option is especially useful in a cluster environment which 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 -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
```

You can now specify the units in which the statistics data is displayed. 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.The `-u` option accepts the following options:

`k`	Displays throughput in kiloblocks.
`m`	Displays throughput in megablocks.
`g`	Displays throughput in gigablocks.
`bytes`	Displays throughput in exact number of bytes.
`us`	Displays average read/write time in microseconds.

For example: 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
```

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 and are stored in the file `/etc/vx/dmppolicy.info`.

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 194.

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 DMP log file. Also, notification is sent to `vxnotify` clients.

The value set for minimum redundancy level is stored in the `dmppolicy.info` file, and is persistent. 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 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).

Warning: Starting with release 4.1 of VxVM, I/O policies are recorded in the file `/etc/vx/dmppolicy.info`, and are persistent across reboots of the system.

Do not edit this file yourself.

The following policies may be set:

`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
```

`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 size argument to the partitionsize attribute to specify the partition size. 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 516.

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.

`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 190.

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 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. 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.

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   DEFAULT  CURRENT
============================================
HDS9500-ALUA0 no   yes
```

The use_all_paths attribute only applies to A/A-A 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 configure load balancing in a SAN environment where there are multiple primary paths to an Active/Passive device through several SAN switches. As can be seen 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 the dd command is used 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 or array ports

Disabling I/O through a path, HBA controller or array port prevents DMP from issuing I/O requests through the specified path, or the paths that are connected to the specified controller or array port. The command blocks until all pending I/O requests issued through the paths are completed.

Note: From release 5.0 of VxVM, this operation is supported for controllers that are used to access disk arrays on which cluster-shareable disk groups are configured.

To disable I/O for a path, use the following command:

```
# vxdmpadm [-c|-f] disable path=path_name
```

To disable I/O for multiple 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 an HBA controller, use the following command:

```
# vxdmpadm [-c|-f] disable ctlr=ctlr_name
```

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 are examples of using the command 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
```

You can use the -c option to check if there is only a single active path to the disk. If so, the `disable` command fails with an error message unless you use the `-f` option to forcibly disable the path.

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.

Enabling I/O for paths, controllers or array ports

Enabling a controller allows a previously disabled path, HBA controller or array port to accept I/O again. This operation succeeds only if the path, controller or array port 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: From release 5.0 of VxVM, this operation is supported for controllers that are used to access disk arrays on which cluster-shareable disk groups are configured.

To enable I/O for a path, use the following command:

```
# vxdmpadm enable path=path_name
```

To enable I/O for multiple paths, use the following command:

```
# vxdmpadm enable path=path_name1,path_name2,path_nameN
```

To enable I/O for the paths connected to an HBA controller, use the following command:

```
# vxdmpadm enable ctlr=ctlr_name
```

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 [-f] 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
```

Renaming an enclosure

The `vxdmpadm setattr` command can be used to assign a meaningful name to an existing enclosure, for example:

```
# vxdmpadm setattr enclosure enc0 name=GRP1
```

This example changes the name of an enclosure from `enc0` 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
============================================================
other0        OTHER         OTHER_DISKS               CONNECTED
jbod0         X1            X1_DISKS                  CONNECTED
GRP1          ACME          60020f20000001a90000      CONNECTED
```

Configuring the response to I/O failures

You can configure how 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 times for a single path.

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 206.

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 corresponding to `recoveryoption=fixedretry retrycount=5`, for example:

```
# vxdmpadm setattr arraytype A/A recoveryoption=default
```

The above command also has the effect of configuring I/O throttling with the default settings.

See "Configuring the I/O throttling mechanism" on page 204.

Note: The response to I/O failure settings is persistent across reboots of the system.

Configuring the I/O throttling mechanism

By default, 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 206.

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 202.

Note: The I/O throttling settings are persistent across reboots of the system.

Configuring Subpaths Failover Groups (SFG)

The Subpaths Failover Groups (SFG) feature can be turned on or off using the tunable `dmp_sfg_threshold`.

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 which triggers SFG. The default is 1.

```
# vxdmpadm settune dmp_sfg_threshold=N
```

The default value of the tunable is "1" which represents that the feature is on.

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}
```

Configuring Low Impact Path Probing

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 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
```

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 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]   Timebound[60]
HDS9500-ALUA0   Error-Retry      Fixed-Retry[5] Timebound[20]
```

This shows the default and current policy options and their values.

Table 4-1 summarizes the possible recovery option settings for retrying I/O after an error.

Table 4-1 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 4-2 summarizes the possible recovery option settings for throttling I/O.

Table 4-2 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

DMP maintains a kernel thread 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 thread 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 208.

Use the `vxdmpadm start restore` command to configure one of the following restore policies. The policy will remain in effect until the restore thread is stopped or the values are changed using `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 start restore [interval=seconds] 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 start restore [interval=seconds] \
    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 start restore [interval=seconds] \
    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 there is a large number of paths available. The command to configure this policy is:

  ```
  # vxdmpadm start restore interval=seconds \
    policy=check_periodic [period=number]
  ```

  The interval attribute must be specified for this policy. The default number of cycles between running the check_all policy is 10.

The interval attribute specifies how often the path restoration thread examines the paths. For example, after stopping the path restoration thread, the polling interval can be set to 400 seconds using the following command:

```
# vxdmpadm start restore interval=400
```

Starting with the 5.0MP3 release, you can also use the vxdmpadm settune command to change the restore policy, restore interval, and restore period. This method stores the values for these arguments as DMP tunables. The settings are immediately applied and are persistent across reboots. Use the vxdmpadm gettune to view the current settings.

See "DMP tunable parameters" on page 516.

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 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 following command to display the status of the automatic path restoration kernel thread, its polling interval, and the policy that it uses to check the condition of paths:

```
# vxdmpadm stat restored
```

This produces output such as the following:

```
The number of daemons running : 1
The interval of daemon: 300
The policy of daemon: check_disabled
```

Displaying information about the DMP error-handling thread

To display information about the kernel thread that handles DMP errors, use the following command:

```
# vxdmpadm stat errord
```

One daemon should be shown as running.

Configuring array policy modules

An array policy module (APM) is a dynamically loadable kernel module (plug-in for DMP) for use in conjunction with an array. An APM 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 are provided by DMP or by another version of the APM.

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 that is provided by 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.

Specifying the `-r` option allows you to remove an APM that is not currently loaded:

```
# vxdmpadm -r cfgapm module_name
```

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

Chapter 5

Online dynamic reconfiguration

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
- Updating the array controller firmware, also known as a nondisruptive upgrade

Reconfiguring a LUN online that is under DMP control

System administrators and storage administrators may need to modify the set of LUNs provisioned to a server. You can change the LUN configuration dynamically, without performing a reconfiguration reboot on the host.

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.

The operations are as follows:

- Dynamic LUN removal from an existing target ID
  See "Removing LUNs dynamically from an existing target ID" on page 212.

- Dynamic new LUN addition to a new target ID
  See "Adding new LUNs dynamically to a new target ID" on page 214.

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 cleanup the operating system device tree.

The high-level procedure and the VxVM 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 "Cleaning up the operating system device tree after removing LUNs" on page 216.

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 216.

9 Use VxVM 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.

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 VxVM 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 216.

   Repeat step 2 and step 3 until you see that all the LUNs have been added.

5 Use VxVM 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 "Cleaning up the operating system device tree after removing LUNs" on page 216.

```
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
    ```

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.

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.

Legacy storage subsystems contain two controllers for redundancy. An online upgrade is done one controller at a time. DMP fails over all I/O to the second controller while the first controller is undergoing an Online Controller Upgrade. After the first controller has completely staged the code, it reboots, resets, and comes online with the new version of the code. The second controller goes through the same process, and I/O fails over to the first 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 retries 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 retries the I/Os for 300 seconds, or until the I/O succeeds.

To verify which arrays support Online Controller Upgrade or NDU, see the hardware compatibility list (HCL) at the following URL:

http://entsupport.symantec.com/docs/330441

Chapter 6

Creating and administering disk groups

This chapter includes the following topics:

About disk groups

Disk groups are named collections of disks that share a common configuration. Volumes are created within a disk group and are restricted to using disks within that disk group.

Data related to a particular set of applications or a particular group of users may need to be made accessible on another system. These situations include the following:

- A system has failed and its data needs to be moved to other systems.
- The work load must be balanced across a number of systems.

You must place disks in one or more disk groups before VxVM can use the disks for volumes. It is important that you locate data related to particular applications or users on an identifiable set of disks. When you need to move these disks, this lets you move only the application or user data that should be moved. The disk group also provides a single object to move, rather than specifying all objects within the disk group individually.

As system administrator, you can create additional disk groups to arrange your system's disks for different purposes. Many systems only use one disk group, unless they have a large number of disks. You can initialize, reserve, and add disks to disk groups at any time. You do not have to add disks to disk groups until the disks are needed to create VxVM objects.

Veritas Volume Manager's Cross-platform Data Sharing (CDS) feature lets you move VxVM disks and objects between machines that are running under different operating systems. Disk groups may be made compatible with CDS.

For more information about CDS, see the *Veritas Storage Foundation Advanced Features Administrator's Guide*.

When you add a disk to a disk group, you name that disk (for example, `mydg02`). This name identifies a disk for operations such as creating or mirroring a volume. The name also relates directly to the underlying physical disk. If a physical disk is moved to a different target address or to a different controller, the name `mydg02`

continues to refer to it. You can replace disks by first associating a different physical disk with the name of the disk to be replaced and then recovering any volume data that was stored on the original disk (from mirrors or backup copies).

Having disk groups that contain many disks and VxVM objects causes the private region to fill. If you have large disk groups that are expected to contain more than several hundred disks and VxVM objects, you should set up disks with larger private areas. A major portion of a private region provides space for a disk group configuration database that contains records for each VxVM object in that disk group. Because each configuration record is approximately 256 bytes, you can use the configuration database copy size to estimate the number of records that you can create in a disk group. You can obtain the copy size in blocks from the output of the `vxdg list` *`diskgroup`* command. It is the value of the `permlen` parameter on the line starting with the string "`config:`". This value is the smallest of the `len` values for all copies of the configuration database in the disk group. The value of the `free` parameter indicates the amount of remaining free space in the configuration database.

See "Displaying disk group information" on page 228.

One way to overcome the problem of running out of free space is to split the affected disk group into two separate disk groups.

See "Reorganizing the contents of disk groups" on page 262.

See "Backing up and restoring disk group configuration data" on page 278.

Before Veritas Volume Manager (VxVM) 4.0, a system installed with VxVM was configured with a default disk group, `rootdg`. This group had to contain at least one disk. By default, operations were directed to the `rootdg` disk group. From release 4.0 onward, VxVM can function without any disk group having been configured. Only when the first disk is placed under VxVM control must a disk group be configured. Now, you do not have to name any disk group `rootdg`. If you name a disk group `rootdg`, it has no special properties because of this name.

See "Specification of disk groups to commands" on page 222.

Note: Most VxVM commands require superuser or equivalent privileges.

Additionally, before VxVM 4.0, some commands such as `vxdisk` were able to deduce the disk group if the name of an object was uniquely defined in one disk group among all the imported disk groups. Resolution of a disk group in this way is no longer supported for any command.

Specification of disk groups to commands

Many VxVM commands let you specify a disk group using the `-g` option. For example, the following command creates a volume in the disk group, `mktdg`:

```
# vxassist -g mktdg make mktvol 5g
```

The block special device that corresponds to this volume is `/dev/vx/dsk/mktdg/mktvol`.

System-wide reserved disk groups

The following disk group names are reserved, and cannot be used to name any disk groups that you create:

`bootdg`	Specifies the boot disk group. This is an alias for the disk group that contains the volumes that are used to boot the system. VxVM sets `bootdg` to the appropriate disk group if it takes control of the root disk. Otherwise, `bootdg` is set to `nodg` (no disk group).
`defaultdg`	Specifies the default disk group. This is an alias for the disk group name that should be assumed if the `-g` option is not specified to a command, or if the `VXVM_DEFAULTDG` environment variable is undefined. By default, `defaultdg` is set to `nodg` (no disk group).
`nodg`	Specifies to an operation that no disk group has been defined. For example, if the root disk is not under VxVM control, `bootdg` is set to `nodg`.

Warning: Do not try to change the assigned value of `bootdg`. If you change the value, it may render your system unbootable.

If you have upgraded your system, you may find it convenient to continue to configure a disk group named `rootdg` as the default disk group (`defaultdg`). `defaultdg` and `bootdg` do not have to refer to the same disk group. Also, neither the default disk group nor the boot disk group have to be named `rootdg`.

Rules for determining the default disk group

You should use the `-g` option to specify a disk group to 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`. 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`. If this alias is undefined, the following rule is applied.
  See "Displaying and specifying the system-wide default disk group" on page 223.
- 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

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. 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.

Use the following command to specify the name of the disk group that is aliased by `defaultdg`:

```
# vxdctl defaultdg diskgroup
```

If `bootdg` is specified as the argument to this command, the default disk group is set to be the same as the currently defined system-wide boot disk group.

If `nodg` is specified as the argument to the `vxdctl defaultdg` command, the default disk group is undefined.

The specified disk group is not required to exist on the system.

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

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

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 5.1SP1 creates disk groups with version 160.

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 276.

Table 6-1 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 6-1 Disk group version assignments

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 6-1 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

Table 6-1 Disk group version assignments *(continued)*

VxVM release	Introduces disk group version	New features supported	Supports disk group versions
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
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

Table 6-1 Disk group version assignments *(continued)*

VxVM release	Introduces disk group version	New features supported	Supports disk group versions
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

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 231.

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.

You can create a shared disk group.

See "Creating a shared disk group" on page 467.

You can also use the `vxdiskadd` command to create a new disk group:

```
# vxdiskadd sdd
```

where `sdd` is the device name of a disk that is not currently assigned to a disk group. The command dialog is similar to that described for the `vxdiskadm` command.

See "Adding a disk to VxVM" on page 106.

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. In Veritas Volume Manager 4.0 and later releases, 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
```

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 5.1SP1 is 160. 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 5.1SP1 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.

Adding a disk to a disk group

To add a disk to an existing disk group, select `Add or initialize one or more disks` from the main menu of the `vxdiskadm` command.

You can also use the `vxdiskadd` command to add a disk to a disk group. Enter the following:

```
# vxdiskadd sde
```

where `sde` is the device name of a disk that is not currently assigned to a disk group. The command dialog is similar to that described for the `vxdiskadm` command.

See "Adding a disk to VxVM" on page 106.

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 275.

As an alternative to disabling the disk group, you can destroy it.

See "Destroying a disk group" on page 276.

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.

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 269.

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 236.

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.

See "Importing disk groups as shared" on page 467.

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

- Use the following `vxdefault` command to turn off automatic volume recovery.

```
# vxdefault set autostartvolumes off
```

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 a shared disk group. When this situation happened, you had to run the `vxdg reminor` command manually to resolve the minor conflicts. Starting with 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 between shared minor numbers and private minor numbers. For example, you may need 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 package.

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 236.

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 255.

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.

See "Importing disk groups as shared" on page 467.

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 236.

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.

Handling cloned disks with duplicated identifiers

A disk may be copied by creating a hardware snapshot (such as an EMC BCV™ or Hitachi ShadowCopy™) or clone, by using dd or a similar command to replicate the disk, or by building a new LUN from the space that was previously used by a deleted LUN. To avoid the duplicate disk ID condition, the default action of VxVM is to prevent such duplicated disks from being imported.

Advanced disk arrays provide hardware tools that you can use to create clones of existing disks outside the control of VxVM. For example, these disks may have been created as hardware snapshots or mirrors of existing disks in a disk group. As a result, the VxVM private region is also duplicated on the cloned disk. When the disk group containing the original disk is subsequently imported, VxVM detects multiple disks that have the same disk identifier that is defined in the private region. In releases prior to 5.0, if VxVM could not determine which disk was the original, it would not import such disks into the disk group. The duplicated disks would have to be re-initialized before they could be imported.

From release 5.0, a unique disk identifier (UDID) is added to the disk's private region when the disk is initialized or when the disk is imported into a disk group (if this identifier does not already exist). Whenever a disk is brought online, the current UDID value that is known to the Device Discovery Layer (DDL) is compared with the UDID that is set in the disk's private region. If the UDID values do not match, the udid_mismatch flag is set on the disk. This flag can be viewed with the vxdisk list command. This allows a LUN snapshot to be imported on the same host as the original LUN. It also allows multiple snapshots of the same LUN to be simultaneously imported on a single server, which can be useful for off-host backup and processing.

A new set of vxdisk and vxdg operations are provided to handle such disks; either by writing the DDL value of the UDID to a disk's private region, or by tagging a disk and specifying that it is a cloned disk to the vxdg import operation.

The following is sample output from the vxdisk list command showing that disks sdg, sdh and sdi are marked with the udid_mismatch flag:

```
# vxdisk list

DEVICE       TYPE          DISK     GROUP    STATUS
sda          auto:cdsdisk  -        -        online
sdb          auto:cdsdisk  -        -        online
 .
 .
 .
sde          auto:cdsdisk  -        -        online
sdf          auto:cdsdisk  -        -        online
sdg          auto:cdsdisk  -        -        online udid_mismatch
sdh          auto:cdsdisk  -        -        online udid_mismatch
sdi          auto:cdsdisk  -        -        online udid_mismatch
```

Writing a new UDID to a disk

You can use the following command to update the unique disk identifier (UDID) for one or more disks. This is useful when building a new LUN from space previously used by a deleted LUN, for example.

```
# vxdisk [-f] [-g diskgroup ] updateudid disk ...
```

This command 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. The `-f` option must be specified if VxVM has not set the `udid_mismatch` flag for a disk.

For example, the following command updates the UDIDs for the disks `sdg` and `sdh`:

```
# vxdisk updateudid sdg sdh
```

Importing a disk group containing cloned disks

By default, disks on which the `udid_mismatch` flag or the `clone_disk` flag has been set are not imported by the `vxdg import` command unless all disks in the disk group have at least one of these flags set, and no two of the disks have the same UDID. You can then import 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.

If the `clone_disk` flag is set on a disk, this indicates the disk was previously imported into a disk group with the `udid_mismatch` flag set.

The `-o updateid` option can be specified 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.) However, the import fails if multiple copies of one or more cloned disks exist. In this case, you can 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, the following command sets the tag, `my_tagged_disks`, on several disks that are to be imported together:

```
# vxdisk settag my_tagged_disks sdg sdh
```

Alternatively, you can update the UDIDs of the cloned disks.

See "Writing a new UDID to a disk" on page 245.

To check which disks are tagged, use the `vxdisk listtag` command:

```
# vxdisk listtag

DEVICE       NAME                VALUE
sda          -                   -
sdb          -                   -
 .
 .
 .
sde          my_tagged_disks     -
sdf          my_tagged_disks     -
sdg          my_tagged_disks     -
sdh          my_tagged_disks     -
sdi          -                   -
```

The configuration database in a VM disk's private region contains persistent configuration data (or metadata) about the objects in a disk group. This database is consulted by VxVM when the disk group is imported. At least one of the cloned disks that are being imported must contain a copy of the current configuration database in its private region.

You can use the following command to ensure that a copy of the metadata is placed on a disk, regardless of the placement policy for the disk group:

```
# vxdisk [-g diskgroup] set disk keepmeta=always
```

Alternatively, use the following command to place a copy of the configuration database and kernel log on all disks in a disk group that share a given tag:

```
# vxdg [-g diskgroup] set tagmeta=on tag=tagname nconfig=all \
  nlog=all
```

To check which disks in a disk group contain copies of this configuration information, use the `vxdg listmeta` command:

```
# vxdg [-q] listmeta diskgroup
```

The `-q` option can be specified to suppress detailed configuration information from being displayed.

The tagged disks in the disk group may be imported by specifying the tag to the `vxdg import` command in addition to the `-o useclonedev=on` option:

```
# vxdg -o useclonedev=on -o tag=my_tagged_disks import mydg
```

If you have already imported the non-cloned disks in a disk group, you can use the `-n` and `-t` option to specify a temporary name for the disk group containing the cloned disks:

```
# vxdg -t -n clonedg -o useclonedev=on -o tag=my_tagged_disks \
  import mydg
```

See "Renaming a disk group" on page 253.

To remove a tag from a disk, use the `vxdisk rmtag` command, as shown in the following example:

```
# vxdisk rmtag tag=my_tagged_disks sdh
```

See the `vxdisk`(1M) and `vxdg`(1M) manual pages.

Sample cases of operations on cloned disks

The following sections contain examples of operations that can be used with cloned disks:

See "Enabling configuration database copies on tagged disks" on page 248.

See "Importing cloned disks without tags" on page 249.

See "Importing cloned disks with tags" on page 251.

Enabling configuration database copies on tagged disks

In this example, the following commands have been used to tag some of the disks in an Hitachi TagmaStore array:

```
# vxdisk settag TagmaStore-USP0_28 t1=v1
# vxdisk settag TagmaStore-USP0_28 t2=v2
# vxdisk settag TagmaStore-USP0_24 t2=v2
# vxdisk settag TagmaStore-USP0_25 t1=v1
```

These tags can be viewed by using the `vxdisk listtag` command:

```
# vxdisk listtag

DEVICE               NAME    VALUE
TagmaStore-USP0_24   t2      v2
TagmaStore-USP0_25   t1      v1
TagmaStore-USP0_28   t1      v1
TagmaStore-USP0_28   t2      v2
```

The following command ensures that configuration database copies and kernel log copies are maintained for all disks in the disk group `mydg` that are tagged as `t1`:

```
# vxdg -g mydg set tagmeta=on tag=t1 nconfig=all nlog=all
```

The disks for which such metadata is maintained can be seen by using this command:

```
# vxdisk -o alldgs list

DEVICE             TYPE         DISK    GROUP   STATUS
TagmaStore-USP0_10 auto:cdsdisk -       -       online
TagmaStore-USP0_24 auto:cdsdisk mydg02  mydg    online
TagmaStore-USP0_25 auto:cdsdisk mydg03  mydg    online tagmeta
TagmaStore-USP0_26 auto:cdsdisk -       -       online
TagmaStore-USP0_27 auto:cdsdisk -       -       online
TagmaStore-USP0_28 auto:cdsdisk mydg01  mydg    online tagmeta
```

Alternatively, the following command can be used to ensure that a copy of the metadata is kept with a disk:

```
# vxdisk set TagmaStore-USP0_25 keepmeta=always
# vxdisk -o alldgs list

DEVICE             TYPE         DISK    GROUP   STATUS
```

```
TagmaStore-USP0_10 auto:cdsdisk -       -       online
TagmaStore-USP0_22 auto:cdsdisk -       -       online
TagmaStore-USP0_23 auto:cdsdisk -       -       online
TagmaStore-USP0_24 auto:cdsdisk mydg02  mydg    online
TagmaStore-USP0_25 auto:cdsdisk mydg03  mydg    online keepmeta
TagmaStore-USP0_28 auto:cdsdisk mydg01  mydg    online
```

Importing cloned disks without tags

In the first example, cloned disks (ShadowImage™ devices) from an Hitachi TagmaStore array will be imported. The primary (non-cloned) disks, `mydg01`, `mydg02` and `mydg03`, are already imported, and the cloned disks are not tagged.

```
# vxdisk -o alldgs list

DEVICE              TYPE         DISK    GROUP   STATUS
TagmaStore-USP0_3   auto:cdsdisk -       (mydg)  online udid_mismatch
TagmaStore-USP0_23 auto:cdsdisk mydg02   mydg   online
TagmaStore-USP0_25 auto:cdsdisk mydg03   mydg   online
TagmaStore-USP0_30 auto:cdsdisk -       (mydg)  online udid_mismatch
TagmaStore-USP0_31 auto:cdsdisk -       (mydg)  online udid_mismatch
TagmaStore-USP0_32 auto:cdsdisk mydg01   mydg   online
```

To import the cloned disks, they must be assigned a new disk group name, and their UDIDs must be updated:

```
# vxdg -n snapdg -o useclonedev=on -o updateid import mydg
# vxdisk -o alldgs list

DEVICE              TYPE         DISK    GROUP   STATUS
TagmaStore-USP0_3   auto:cdsdisk mydg03  snapdg   online clone_disk
TagmaStore-USP0_23 auto:cdsdisk mydg02   mydg    online
TagmaStore-USP0_25 auto:cdsdisk mydg03   mydg    online
TagmaStore-USP0_30 auto:cdsdisk mydg02   snapdg   online clone_disk
TagmaStore-USP0_31 auto:cdsdisk mydg01   snapdg   online clone_disk
TagmaStore-USP0_32 auto:cdsdisk mydg01   mydg    online
```

Note that the state of the imported cloned disks has changed from `online udid_mismatch` to `online clone_disk`.

In the next example, none of the disks (neither cloned nor non-cloned) have been imported:

```
# vxdisk -o alldgs list
```

```
DEVICE              TYPE          DISK    GROUP   STATUS
TagmaStore-USP0_3   auto:cdsdisk -       (mydg)  online udid_mismatch
TagmaStore-USP0_23  auto:cdsdisk -       (mydg)  online
TagmaStore-USP0_25  auto:cdsdisk -       (mydg)  online
TagmaStore-USP0_30  auto:cdsdisk -       (mydg)  online udid_mismatch
TagmaStore-USP0_31  auto:cdsdisk -       (mydg)  online udid_mismatch
TagmaStore-USP0_32  auto:cdsdisk -       (mydg)  online
```

To import only the cloned disks into the `mydg` disk group:

```
# vxdg -o useclonedev=on -o updateid import mydg
# vxdisk -o alldgs list

DEVICE              TYPE          DISK    GROUP   STATUS
TagmaStore-USP0_3   auto:cdsdisk mydg03  mydg    online clone_disk
TagmaStore-USP0_23  auto:cdsdisk -      (mydg)   online
TagmaStore-USP0_25  auto:cdsdisk -      (mydg)   online
TagmaStore-USP0_30  auto:cdsdisk mydg02  mydg    online clone_disk
TagmaStore-USP0_31  auto:cdsdisk mydg01  mydg    online clone_disk
TagmaStore-USP0_32  auto:cdsdisk -      (mydg)   online
```

In the next example, a cloned disk (BCV device) from an EMC Symmetrix DMX array is to be imported. Before the cloned disk, `EMC0_27`, has been split off from the disk group, the `vxdisk list` command shows that it is in the `error udid_mismatch` state:

```
# vxdisk -o alldgs list
DEVICE              TYPE          DISK    GROUP   STATUS
EMC0_1              auto:cdsdisk EMC0_1  mydg    online
EMC0_27             auto          -       -       error udid_mismatch
```

The `symmir` command is used to split off the BCV device:

```
# /usr/symcli/bin/symmir -g mydg split DEV001
```

After updating VxVM's information about the disk by running the `vxdisk scandisks` command, the cloned disk is in the `online udid_mismatch` state:

```
# vxdisk -o alldgs list
DEVICE              TYPE          DISK    GROUP   STATUS
EMC0_1              auto:cdsdisk EMC0_1  mydg    online
EMC0_27             auto:cdsdisk -       -       online udid_mismatch
```

The following command imports the cloned disk into the new disk group `newdg`, and updates the disk's UDID:

```
# vxdg -n newdg -o useclonedev=on -o updateid import mydg
```

The state of the cloned disk is now shown as `online clone_disk`:

```
# vxdisk -o alldgs list
DEVICE              TYPE          DISK    GROUP  STATUS
EMC0_1              auto:cdsdisk EMC0_1  mydg   online
EMC0_27             auto:cdsdisk EMC0_1  newdg  online clone_disk
```

Importing cloned disks with tags

In this example, cloned disks (BCV devices) from an EMC Symmetrix DMX array will be imported. The primary (non-cloned) disks, `mydg01`, `mydg02` and `mydg03`, are already imported, and the cloned disks with the tag `t1` are to be imported.

```
# 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 mydg03  mydg   online
EMC0_24             auto:cdsdisk -       (mydg)  online udid_mismatch
```

The disks are tagged as follows:

```
# vxdisk listtag

DEVICE              NAME    VALUE
EMC0_4              t2      v2
EMC0_4              t1      v1
EMC0_6              t2      v2
EMC0_8              t1      v1
EMC0_15             t2      v2
EMC0_18             t1      v1
EMC0_24             t1      v1
EMC0_24             t2      v2
```

To import the cloned disks that are tagged as `t1`, they must be assigned a new disk group name, and their UDIDs must be updated:

```
# vxdg -n bcvdg -o useclonedev=on -o tag=t1 -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 mydg03  bcvdg  online clone_disk
EMC0_15         auto:cdsdisk -      (mydg)  online udid_mismatch
EMC0_18         auto:cdsdisk mydg03  mydg   online
EMC0_24         auto:cdsdisk mydg01  bcvdg  online clone_disk
```

As the cloned disk `EMC0_15` is not tagged as `t1`, it is not imported. Note that the state of the imported cloned disks has changed from `online udid_mismatch` to `online clone_disk`.

By default, the state of imported cloned disks is shown as `online clone_disk`. This can be removed by using the `vxdisk set` command as shown here:

```
# vxdisk set EMC0_8 clone=off
# 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 mydg03  bcvdg  online
EMC0_15         auto:cdsdisk -      (mydg)  online udid_mismatch
EMC0_18         auto:cdsdisk mydg03  mydg   online
EMC0_24         auto:cdsdisk mydg01  bcvdg  online clone_disk
```

In the next example, none of the disks (neither cloned nor non-cloned) have been imported:

```
# vxdisk -o alldgs list

DEVICE          TYPE         DISK    GROUP  STATUS
EMC0_4          auto:cdsdisk -       (mydg) online
EMC0_6          auto:cdsdisk -       (mydg) online
EMC0_8          auto:cdsdisk -       (mydg) online udid_mismatch
EMC0_15         auto:cdsdisk -       (mydg) online udid_mismatch
EMC0_18         auto:cdsdisk -       (mydg) online
EMC0_24         auto:cdsdisk -       (mydg) online udid_mismatch
```

To import only the cloned disks that have been tagged as `t1` into the `mydg` disk group:

```
# vxdg -o useclonedev=on -o tag=t1 -o updateid import mydg
# vxdisk -o alldgs list
```

```
DEVICE          TYPE         DISK    GROUP  STATUS
EMC0_4          auto:cdsdisk -       (mydg) online
EMC0_6          auto:cdsdisk -       (mydg) online
EMC0_8          auto:cdsdisk mydg03  mydg   online clone_disk
EMC0_15         auto:cdsdisk -       (mydg) online udid_mismatch
EMC0_18         auto:cdsdisk -       (mydg) online
EMC0_24         auto:cdsdisk mydg01   mydg  online clone_disk
```

As in the previous example, the cloned disk `EMC0_15` is not tagged as `t1`, and so it is not imported.

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
```

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 122.

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).

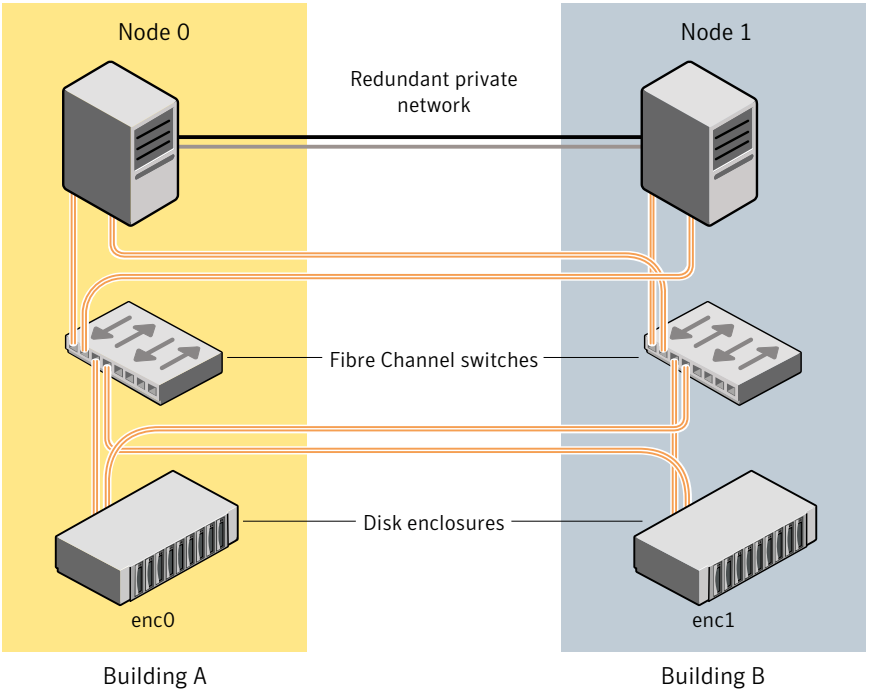
Example of a serial split brain condition in a cluster

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 6-1 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 6-1 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.

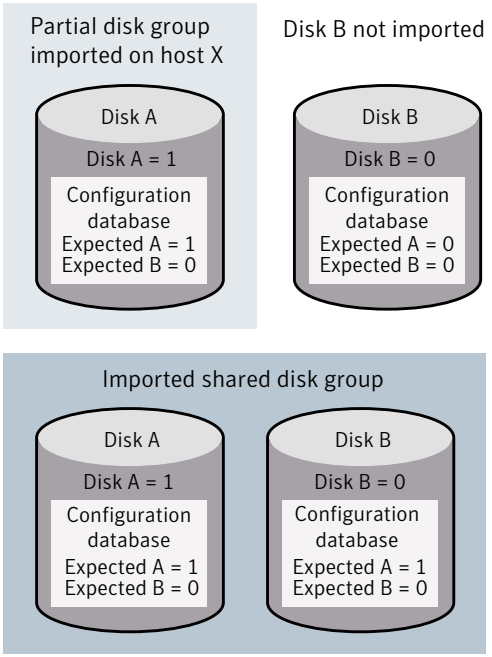
If some disks went missing from the disk group (due to physical disconnection or power failure) and those disks were imported by another host, the serial IDs for the disks in their copies of the configuration database, and also in each disk's private region, are updated separately on that host. When 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 6-2 shows an example of a serial split brain condition that can be resolved automatically by VxVM.

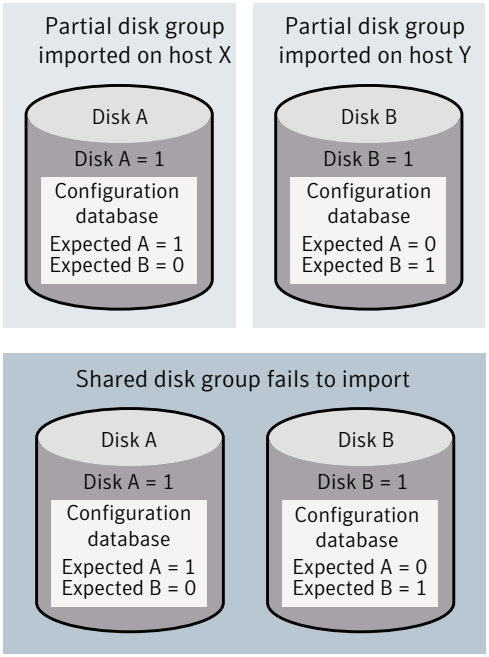
Figure 6-2 Example of a serial split brain condition that can be resolved automatically



- 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 6-3 shows an example of a true serial split brain condition that cannot be resolved automatically by VxVM.

Figure 6-3 Example of a true serial split brain condition that cannot be resolved automatically



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 260.

See "About sites and remote mirrors" on page 477.

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.

Reorganizing the contents of disk groups

There are several circumstances under which you might want to reorganize the contents of your existing disk groups:

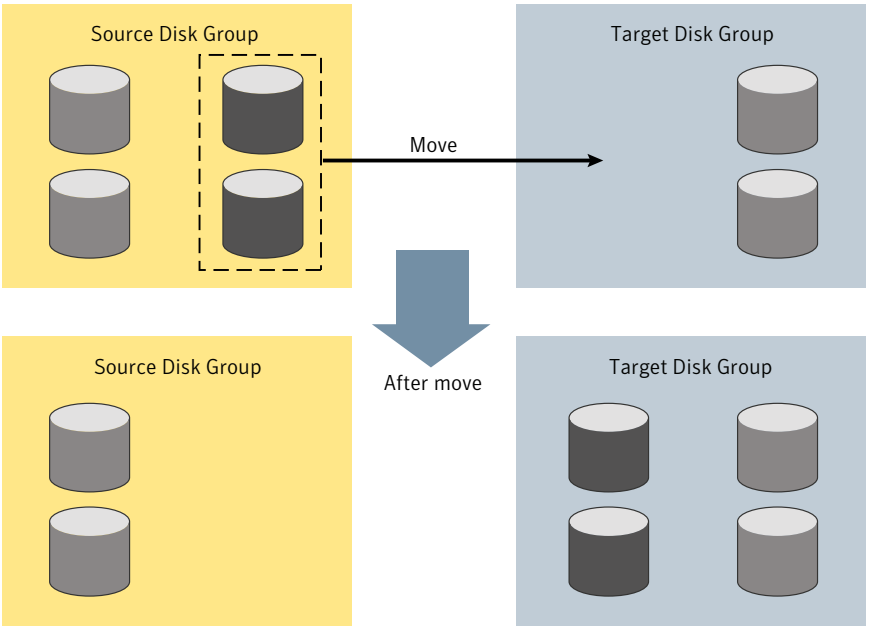
- To group volumes or disks differently as the needs of your organization change. For example, you might want to split disk groups to match the boundaries of separate departments, or to join disk groups when departments are merged.
- To isolate volumes or disks from a disk group, and process them independently 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.
  See "About off-host processing solutions" on page 405.
- 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 6-4 shows the `move` operation.

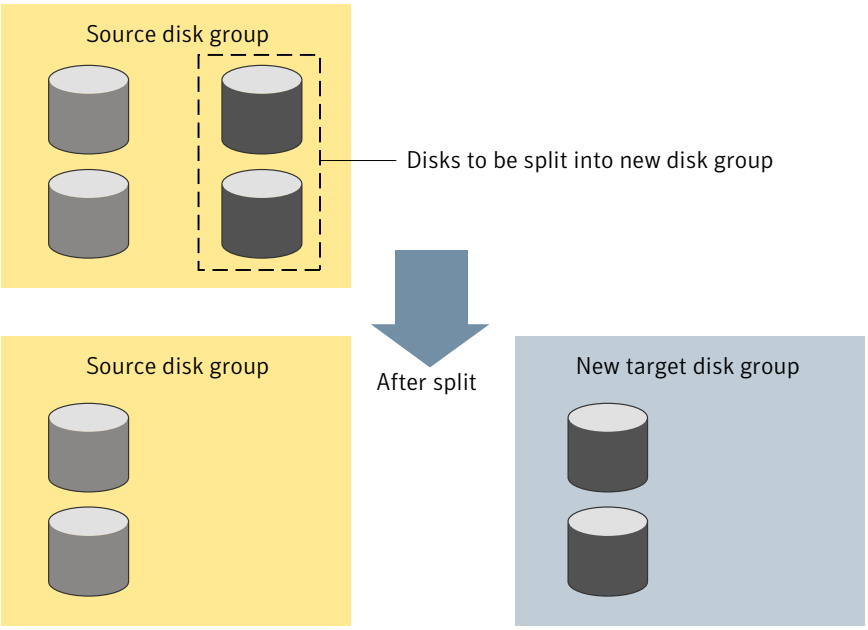
Figure 6-4 Disk group move operation



- 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. An existing deported disk group is destroyed if it has the same name as the target disk group (as is the case for the `vxdg init` command).

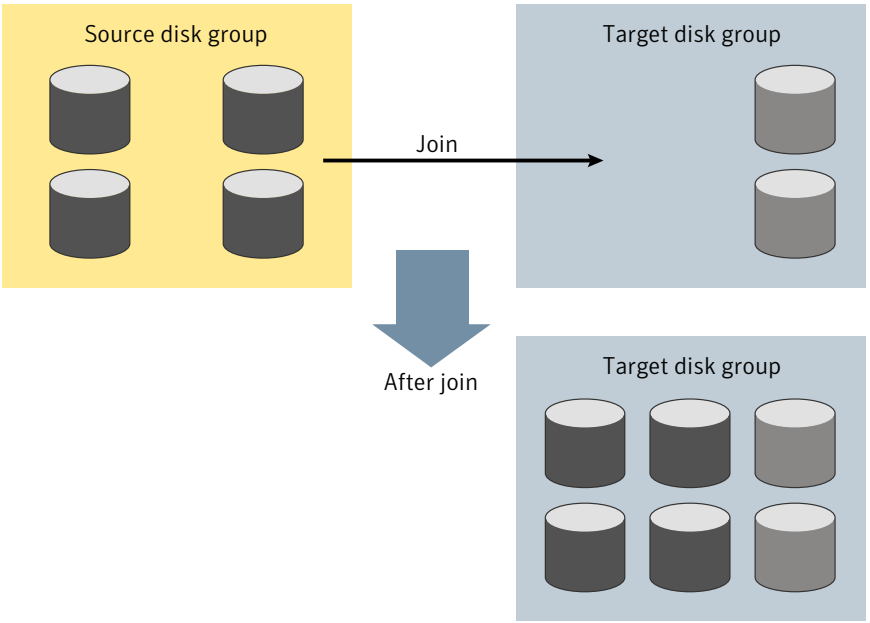
Figure 6-5 shows the `split` operation.

Figure 6-5 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 6-6 shows the `join` operation.

Figure 6-6 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 Volume Manager 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 276.
- 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 236.
- 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 20 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 6-7 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.

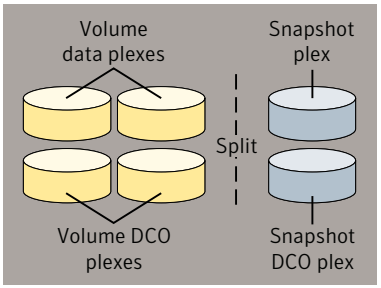
For more information about snapshots and DCO volumes, see the *Veritas Storage Foundation Advanced Features Administrator's Guide*.

See "Specifying storage for version 20 DCO plexes" on page 372.

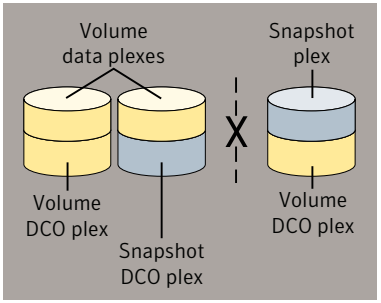
See "FastResync" on page 61.

See "Volume snapshots" on page 59.

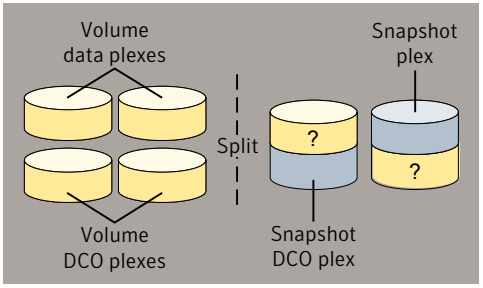
Figure 6-7 Examples of disk groups that can and cannot be split



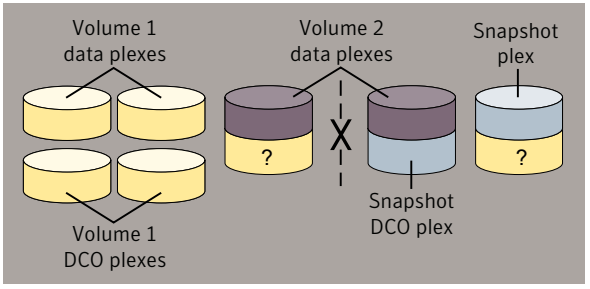
The disk group can be split as the DCO plexes are on dedicated disks, and can therefore accompany the disks that contain the volume data



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 Symantec products. The `TUTIL0` values are temporary; they are not maintained on reboot. The `PUTIL0` values are persistent; they are maintained on reboot.

See "Changing subdisk attributes" on page 291.

```
# 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   -        -        -        -
```

The following commands would also achieve the same result:

```
# vxdg move mydg rootdg mydg01 mydg05
# vxdg move mydg rootdg vol1
```

See "Moving objects between shared disk groups" on page 469.

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 269.

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 Symantec products. The `TUTIL0` values are temporary; they are not maintained on reboot. The `PUTIL0` values are persistent; they are maintained on reboot.

See "Changing subdisk attributes" on page 291.

```
# 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
```

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   -          -         -          -
```

See “Splitting shared disk groups” on page 469.

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 269.

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 Symantec products. The `TUTIL0` values are temporary; they are not maintained on reboot. The `PUTIL0` values are persistent; they are maintained on reboot.

See "Changing subdisk attributes" on page 291.

```
# 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
```

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          -         -          -
```

See "Joining shared disk groups" on page 469.

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
   ```

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 5.1SP1, 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
```

About the configuration daemon in VxVM

The 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.

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 Volume Manager Troubleshooting Guide*.

See the vxconfigbackup(1M) manual page.

See the vxconfigrestore(1M) manual page.

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.

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 storage pool (`st`), 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.
  See "About volume administration" on page 348.
- Renaming any VxVM objects such as volume, disk group, plex, etc.
- Plex attach and detach.
- The `vxconfigbackup` and `vxconfigrestore` command can be used at the cost of losing any intent information

Chapter 7

Creating and administering subdisks and plexes

This chapter includes the following topics:

About subdisks

Subdisks are the low-level building blocks in a Veritas Volume Manager (VxVM) configuration that are required to create plexes and volumes.

See "Creating a volume" on page 311.

Note: Most VxVM commands require superuser or equivalent privileges.

Creating subdisks

Use the `vxmake` command to create VxVM objects, such as subdisks:

```
# vxmake [-g diskgroup] sd subdisk diskname,offset,length
```

where *subdisk* is the name of the subdisk, *diskname* is the disk name, *offset* is the starting point (offset) of the subdisk within the disk, and *length* is the length of the subdisk.

For example, to create a subdisk named `mydg02-01` in the disk group, mydg, that starts at the beginning of disk `mydg02` and has a length of 8000 sectors, use the following command:

```
# vxmake -g mydg sd mydg02-01 mydg02,0,8000
```

Note: As for all VxVM commands, the default size unit is `s`, representing a sector. Add a suffix, such as `k` for kilobyte, `m` for megabyte or `g` for gigabyte, to change the unit of size. For example, `500m` would represent 500 megabytes.

If you intend to use the new subdisk to build a volume, you must associate the subdisk with a plex.

See "Associating subdisks with plexes" on page 287.

Subdisks for all plex layouts (concatenated, striped, RAID-5) are created the same way.

Displaying subdisk information

The `vxprint` command displays information about VxVM objects. To display general information for all subdisks, use this command:

```
# vxprint -st
```

The `-s` option specifies information about subdisks. The `-t` option prints a single-line output record that depends on the type of object being listed.

The following is example output:

```
SD NAME      PLEX      DISK    DISKOFFS LENGTH [COL/]OFF DEVICE   MODE
SV NAME      PLEX      VOLNAME NVOLLAYR LENGTH [COL/]OFF AM/NM   MODE
sd mydg01-01 vol1-01 mydg01 0          102400 0          sdc ENA
sd mydg02-01 vol2-01 mydg02 0          102400 0          sdd ENA
```

You can display complete information about a particular subdisk by using this command:

```
# vxprint [-g diskgroup] -l subdisk
```

For example, the following command displays all information for subdisk `mydg02-01` in the disk group, `mydg`:

```
# vxprint -g mydg -l mydg02-01
```

This command provides the following output:

```
Disk group: mydg

Subdisk:  mydg02-01
info:     disk=mydg02 offset=0 len=205632
assoc:    vol=mvol plex=mvol-02 (offset=0)
flags:    enabled
device:   device=sdd path=/dev/vx/dmp/sdd diskdev=32/68
```

Moving subdisks

Moving a subdisk copies the disk space contents of a subdisk onto one or more other subdisks. If the subdisk being moved is associated with a plex, then the data stored on the original subdisk is copied to the new subdisks. The old subdisk is dissociated from the plex, and the new subdisks are associated with the plex. The association is at the same offset within the plex as the source subdisk. To move a subdisk, use the following command:

```
# vxsd [-g diskgroup] mv old_subdisk new_subdisk [new_subdisk ...]
```

For example, if `mydg03` in the disk group, `mydg`, is to be evacuated, and `mydg12` has enough room on two of its subdisks, use the following command:

```
# vxsd -g mydg mv mydg03-01 mydg12-01 mydg12-02
```

For the subdisk move to work correctly, the following conditions must be met:

- The subdisks involved must be the same size.
- The subdisk being moved must be part of an active plex on an active (`ENABLED`) volume.
- The new subdisk must not be associated with any other plex.

Subdisk can also be moved manually after hot-relocation.

See "Moving relocated subdisks" on page 429.

Splitting subdisks

Splitting a subdisk divides an existing subdisk into two separate subdisks. To split a subdisk, use the following command:

```
# vxsd [-g diskgroup] -s size split subdisk newsd1 newsd2
```

where *subdisk* is the name of the original subdisk, *newsd1* is the name of the first of the two subdisks to be created and *newsd2* is the name of the second subdisk to be created.

The `-s` option is required to specify the size of the first of the two subdisks to be created. The second subdisk occupies the remaining space used by the original subdisk.

If the original subdisk is associated with a plex before the task, upon completion of the split, both of the resulting subdisks are associated with the same plex.

To split the original subdisk into more than two subdisks, repeat the previous command as many times as necessary on the resulting subdisks.

For example, to split subdisk mydg03-02, with size 2000 megabytes into subdisks mydg03-02, mydg03-03, mydg03-04 and mydg03-05, each with size 500 megabytes, all in the disk group, mydg, use the following commands:

```
# vxsd -g mydg -s 1000m split mydg03-02 mydg03-02 mydg03-04
# vxsd -g mydg -s 500m split mydg03-02 mydg03-02 mydg03-03
# vxsd -g mydg -s 500m split mydg03-04 mydg03-04 mydg03-05
```

Joining subdisks

Joining subdisks combines two or more existing subdisks into one subdisk. To join subdisks, the subdisks must be contiguous on the same disk. If the selected subdisks are associated, they must be associated with the same plex, and be contiguous in that plex. To join several subdisks, use the following command:

```
# vxsd [-g diskgroup] join subdisk1 subdisk2 ... new_subdisk
```

For example, to join the contiguous subdisks mydg03-02, mydg03-03, mydg03-04 and mydg03-05 as subdisk mydg03-02 in the disk group, mydg, use the following command:

```
# vxsd -g mydg join mydg03-02 mydg03-03 mydg03-04 mydg03-05 \
  mydg03-02
```

Associating subdisks with plexes

Associating a subdisk with a plex places the amount of disk space defined by the subdisk at a specific offset within the plex. The entire area that the subdisk fills must not be occupied by any portion of another subdisk. There are several ways that subdisks can be associated with plexes, depending on the overall state of the configuration.

If you have already created all the subdisks needed for a particular plex, to associate subdisks at plex creation, use the following command:

```
# vxmake [-g diskgroup] plex plex sd=subdisk,...
```

For example, to create the plex home-1 and associate subdisks mydg02-01, mydg02-00, and mydg02-02 with plex home-1, all in the disk group, mydg, use the following command:

```
# vxmake -g mydg plex home-1 sd=mydg02-01,mydg02-00,mydg02-02
```

Subdisks are associated in order starting at offset 0. If you use this type of command, you do not have to specify the multiple commands needed to create

the plex and then associate each of the subdisks with that plex. In this example, the subdisks are associated to the plex in the order they are listed (after `sd=`). The disk space defined as `mydg02-01` is first, `mydg02-00` is second, and `mydg02-02` is third. This method of associating subdisks is convenient during initial configuration.

Subdisks can also be associated with a plex that already exists. To associate one or more subdisks with an existing plex, use the following command:

```
# vxsd [-g diskgroup] assoc plex subdisk1 [subdisk2 subdisk3 ...]
```

For example, to associate subdisks named `mydg02-01`, `mydg02-00`, and `mydg02-02` with a plex named `home-1`, use the following command:

```
# vxsd -g mydg assoc home-1 mydg02-01 mydg02-00 mydg02-01
```

If the plex is not empty, the new subdisks are added after any subdisks that are already associated with the plex, unless the `-l` option is specified with the command. The `-l` option associates subdisks at a specific offset within the plex.

The `-l` option is required if you previously created a sparse plex (that is, a plex with portions of its address space that do not map to subdisks) for a particular volume, and subsequently want to make the plex complete. To complete the plex, create a subdisk of a size that fits the hole in the sparse plex exactly. Then, associate the subdisk with the plex by specifying the offset of the beginning of the hole in the plex, using the following command:

```
# vxsd [-g diskgroup] -l offset assoc sparse_plex exact_size_subdisk
```

For example, the following command would insert the subdisk, `mydg15-01`, in the plex, `vol10-01`, starting at an offset of 4096 blocks:

```
# vxsd -g mydg -l 4096b assoc vol10-01 mydg15-01
```

Note: The subdisk must be exactly the right size. VxVM does not allow the space defined for two subdisks to overlap within a plex.

For striped or RAID-5 plexes, use the following command to specify a column number and column offset for the subdisk to be added:

```
# vxsd [-g diskgroup] -l column_#/offset assoc plex subdisk ...
```

If only one number is specified with the `-l` option for striped plexes, the number is interpreted as a column number and the subdisk is associated at the end of the column.

For example, the following command would add the subdisk, `mydg11-01`, to the end of column 1 of the plex, `vol02-01`:

```
# vxsd -g mydg -l 1 assoc vol02-01 mydg11-01
```

Alternatively, to add *M* subdisks at the end of each of the *N* columns in a striped or RAID-5 volume, you can use the following form of the `vxsd` command:

```
# vxsd [-g diskgroup] assoc plex subdisk1:0 ... subdiskM:N-1
```

The following example shows how to append three subdisk to the ends of the three columns in a striped plex, `vol-01`, in the disk group, `mydg`:

```
# vxsd -g mydg assoc vol01-01 mydg10-01:0 mydg11-01:1 mydg12-01:2
```

If a subdisk is filling a "hole" in the plex (that is, some portion of the volume logical address space is mapped by the subdisk), the subdisk is considered stale. If the volume is enabled, the association operation regenerates data that belongs on the subdisk. Otherwise, it is marked as stale and is recovered when the volume is started.

Associating log subdisks

Log subdisks are defined and added to a plex that is to become part of a volume on which dirty region logging (DRL) is enabled. DRL is enabled for a volume when the volume is mirrored and has at least one log subdisk.

Warning: Only one log subdisk can be associated with a plex. Because this log subdisk is frequently written, care should be taken to position it on a disk that is not heavily used. Placing a log subdisk on a heavily-used disk can degrade system performance.

See "Dirty region logging" on page 56.

See "Dirty region logging in cluster environments" on page 460.

Log subdisks are ignored as far as the usual plex policies are concerned, and are only used to hold the dirty region log.

Warning: The version 20 DCO volume layout includes space for a DRL. Do not use procedures that are intended for manipulating log subdisks with a volume that has a version 20 DCO volume associated with it.

See "Preparing a volume for DRL and instant snapshots" on page 371.

To add a log subdisk to an existing plex, use the following command:

```
# vxsd [-g diskgroup] aslog plex subdisk
```

where *subdisk* is the name to be used for the log subdisk. The plex must be associated with a mirrored volume before dirty region logging takes effect.

For example, to associate a subdisk named `mydg02-01` with a plex named `vol01-02`, which is already associated with volume `vol01` in the disk group, `mydg`, use the following command:

```
# vxsd -g mydg aslog vol01-02 mydg02-01
```

You can also add a log subdisk to an existing volume with the following command:

```
# vxassist [-g diskgroup] addlog volume disk
```

This command automatically creates a log subdisk within a log plex on the specified disk for the specified volume.

Dissociating subdisks from plexes

To break an established connection between a subdisk and the plex to which it belongs, the subdisk is dissociated from the plex. A subdisk is dissociated when the subdisk is removed or used in another plex. To dissociate a subdisk, use the following command:

```
# vxsd [-g diskgroup] [-o force] dis subdisk
```

For example, to dissociate a subdisk named `mydg02-01` from the plex with which it is currently associated in the disk group, `mydg`, use the following command:

```
# vxsd -g mydg dis mydg02-01
```

You can additionally remove the dissociated subdisks from VxVM control using the following form of the command:

```
# vxsd [-g diskgroup] -o rm dis subdisk
```

Warning: If the subdisk maps a portion of a volume's address space, dissociating it places the volume in DEGRADED mode. In this case, the `dis` operation prints a warning and must be forced using the `-o force` option to succeed. Also, if removing the subdisk makes the volume unusable, because another subdisk in the same stripe is unusable or missing and the volume is not DISABLED and empty, the operation is not allowed.

Removing subdisks

To remove a subdisk, use the following command:

```
# vxedit [-g diskgroup] rm subdisk
```

For example, to remove a subdisk named `mydg02-01` from the disk group, `mydg`, use the following command:

```
# vxedit -g mydg rm mydg02-01
```

Changing subdisk attributes

Warning: To avoid possible data loss, change subdisk attributes with extreme care.

The `vxedit` command changes attributes of subdisks and other VxVM objects. To change subdisk attributes, use the following command:

```
# vxedit [-g diskgroup] set attribute=value ... subdisk ...
```

The subdisk fields you can change with the `vxedit` command include the following:

`name` Subdisk name.

`putiln` Persistent utility field(s) used to manage objects and communication between different commands and Symantec products.

`putiln` field attributes are maintained on reboot. *`putiln`* fields are organized as follows:

- `putil0` is set by VxVM.
- `putil1` is set by other Symantec products such as Veritas Operations Manager (VOM) or the Veritas Enterprise Administrator (VEA) console.
- `putil2` is available for you to set for site-specific purposes.

If a command is stopped in the middle of an operation, these fields may need to be cleaned up.

tutiln Nonpersistent (temporary) utility field(s) used to manage objects and communication between different commands and Symantec products.

tutiln field attributes are not maintained on reboot. *tutiln* fields are organized as follows:

- tutil0 is set by VxVM.
- tutil1 is set by other Symantec products such as Veritas Operations Manager (VOM) or the Veritas Enterprise Administrator (VEA) console..
- tutil2 is available for you to set for site-specific purposes.

If a command is stopped in the middle of an operation, these fields may need to be cleaned up.

len Subdisk length. This value is a standard Veritas Volume Manager length number.

See the vxintro(1M) manual page.

You can only change the length of a subdisk if the subdisk is disassociated. You cannot increase the length of a subdisk to the point where it extends past the end of the disk or it overlaps a reserved disk region on another disk.

comment Comment.

For example, to change the comment field of a subdisk named mydg02-01 in the disk group, mydg, use the following command:

```
# vxedit -g mydg set comment="subdisk comment" mydg02-01
```

To prevent a particular subdisk from being associated with a plex, set the putil0 field to a non-null string, as shown in the following command:

```
# vxedit -g mydg set putil0="DO-NOT-USE" mydg02-01
```

See the vxedit(1M) manual page.

About plexes

Plexes are logical groupings of subdisks that create 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. Because each data plex must reside on different disks from the other plexes, the replication provided by mirroring prevents data loss in the event of a single-point disk-subsystem failure. Multiple data plexes also provide increased data integrity and reliability.

See “About subdisks” on page 284.

See "Creating a volume" on page 311.

Note: Most VxVM commands require superuser or equivalent privileges.

Creating plexes

Use the `vxmake` command to create VxVM objects, such as plexes. When creating a plex, identify the subdisks that are to be associated with it:

To create a plex from existing subdisks, use the following command:

```
# vxmake [-g diskgroup] plex plex sd=subdisk1[,subdisk2,...]
```

For example, to create a concatenated plex named `vol01-02` from two existing subdisks named `mydg02-01` and `mydg02-02` in the disk group, `mydg`, use the following command:

```
# vxmake -g mydg plex vol01-02 sd=mydg02-01,mydg02-02
```

Creating a striped plex

To create a striped plex, you must specify additional attributes. For example, to create a striped plex named `pl-01` in the disk group, `mydg`, with a stripe width of 32 sectors and 2 columns, use the following command:

```
# vxmake -g mydg plex pl-01 layout=stripe stwidth=32 ncolumn=2 \
  sd=mydg01-01,mydg02-01
```

To use a plex to build a volume, you must associate the plex with the volume.

See "Attaching and associating plexes" on page 298.

Displaying plex information

Listing plexes helps identify free plexes for building volumes. Use the plex (`-p`) option to the `vxprint` command to list information about all plexes.

To display detailed information about all plexes in the system, use the following command:

```
# vxprint -lp
```

To display detailed information about a specific plex, use the following command:

```
# vxprint [-g diskgroup] -l plex
```

The `-t` option prints a single line of information about the plex. To list free plexes, use the following command:

```
# vxprint -pt
```

The following section describes the meaning of the various plex states that may be displayed in the STATE field of `vxprint` output.

Plex states

Plex states reflect whether or not plexes are complete and are consistent copies (mirrors) of the volume contents. VxVM utilities automatically maintain the plex state. However, if a volume should not be written to because there are changes to that volume and if a plex is associated with that volume, you can modify the state of the plex. For example, if a disk with a particular plex located on it begins to fail, you can temporarily disable that plex.

A plex does not have to be associated with a volume. A plex can be created with the `vxmake plex` command and be attached to a volume later.

VxVM utilities use plex states to:

- indicate whether volume contents have been initialized to a known state
- determine if a plex contains a valid copy (mirror) of the volume contents
- track whether a plex was in active use at the time of a system failure
- monitor operations on plexes

This section explains the individual plex states in detail.

See the *Veritas Volume Manager Troubleshooting Guide*.

Table 7-1shows the states that may be associated with a plex.

Table 7-1 Plex states

State	Description
ACTIVE	A plex can be in the ACTIVE state in the following ways: ■ when the volume is started and the plex fully participates in normal volume I/O (the plex contents change as the contents of the volume change) ■ when the volume is stopped as a result of a system crash and the plex is ACTIVE at the moment of the crash In the latter case, a system failure can leave plex contents in an inconsistent state. When a volume is started, VxVM does the recovery action to guarantee that the contents of the plexes marked as ACTIVE are made identical. On a system that is running well, ACTIVE should be the most common state you see for any volume plexes.
CLEAN	A plex is in a CLEAN state when it is known to contain a consistent copy (mirror) of the volume contents and an operation has disabled the volume. As a result, when all plexes of a volume are clean, no action is required to guarantee that the plexes are identical when that volume is started.
DCOSNP	This state indicates that a data change object (DCO) plex attached to a volume can be used by a snapshot plex to create a DCO volume during a snapshot operation.
EMPTY	Volume creation sets all plexes associated with the volume to the EMPTY state to indicate that the plex is not yet initialized.
IOFAIL	The IOFAIL plex state is associated with persistent state logging. When the `vxconfigd` daemon detects an uncorrectable I/O failure on an ACTIVE plex, it places the plex in the IOFAIL state to exclude it from the recovery selection process at volume start time. This state indicates that the plex is out-of-date with respect to the volume, and that it requires complete recovery. It is likely that one or more of the disks associated with the plex should be replaced.
LOG	The state of a dirty region logging (DRL) or RAID-5 log plex is always set to LOG.

Table 7-1 Plex states *(continued)*

State	Description
OFFLINE	The `vxmend off` task indefinitely detaches a plex from a volume by setting the plex state to OFFLINE. Although the detached plex maintains its association with the volume, changes to the volume do not update the OFFLINE plex. The plex is not updated until the plex is put online and reattached with the `vxplex att` task. When this occurs, the plex is placed in the STALE state, which causes its contents to be recovered at the next `vxvol start` operation.
SNAPATT	This state indicates a snapshot plex that is being attached by the snapstart operation. When the attach is complete, the state for the plex is changed to SNAPDONE. If the system fails before the attach completes, the plex and all of its subdisks are removed.
SNAPDIS	This state indicates a snapshot plex that is fully attached. A plex in this state can be turned into a snapshot volume with the `vxplex snapshot` command. If the system fails before the attach completes, the plex is dissociated from the volume. See the `vxplex`(1M) manual page.
SNAPDONE	The SNAPDONE plex state indicates that a snapshot plex is ready for a snapshot to be taken using `vxassist snapshot`.
SNAPTMP	The SNAPTMP plex state is used during a `vxassist snapstart` operation when a snapshot is being prepared on a volume.
STALE	If there is a possibility that a plex does not have the complete and current volume contents, that plex is placed in the STALE state. Also, if an I/O error occurs on a plex, the kernel stops using and updating the contents of that plex, and the plex state is set to STALE. A `vxplex att` operation recovers the contents of a STALE plex from an ACTIVE plex. Atomic copy operations copy the contents of the volume to the STALE plexes. The system administrator can force a plex to the STALE state with a `vxplex det` operation.
TEMP	Setting a plex to the TEMP state eases some plex operations that cannot occur in a truly atomic fashion. For example, attaching a plex to an enabled volume requires copying volume contents to the plex before it can be considered fully attached. A utility sets the plex state to TEMP at the start of such an operation and to an appropriate state at the end of the operation. If the system fails for any reason, a TEMP plex state indicates that the operation is incomplete. A later `vxvol start` dissociates plexes in the TEMP state.

Table 7-1 Plex states *(continued)*

State	Description
TEMPRM	A TEMPRM plex state is similar to a TEMP state except that at the completion of the operation, the TEMPRM plex is removed. Some subdisk operations require a temporary plex. Associating a subdisk with a plex, for example, requires updating the subdisk with the volume contents before actually associating the subdisk. This update requires associating the subdisk with a temporary plex, marked TEMPRM, until the operation completes and removes the TEMPRM plex. If the system fails for any reason, the TEMPRM state indicates that the operation did not complete successfully. A later operation dissociates and removes TEMPRM plexes.
TEMPRMSD	The TEMPRMSD plex state is used by `vxassist` when attaching new data plexes to a volume. If the synchronization operation does not complete, the plex and its subdisks are removed.

Plex condition flags

Table 7-2 shows the plex condition flags that `vxprint` may display in the STATE field.

Table 7-2 Plex condition flags

Condition flag	Description
IOFAIL	The plex was detached as a result of an I/O failure detected during normal volume I/O. The plex is out-of-date with respect to the volume, and in need of complete recovery. However, this condition also indicates a likelihood that one of the disks in the system should be replaced.
NODAREC	No physical disk was found for one of the subdisks in the plex. This implies either that the physical disk failed, making it unrecognizable, or that the physical disk is no longer attached through a known access path. The plex cannot be used until this condition is fixed, or the affected subdisk is dissociated.
NODEVICE	A physical device could not be found corresponding to the disk ID in the disk media record for one of the subdisks associated with the plex. The plex cannot be used until this condition is fixed, or the affected subdisk is dissociated.

Table 7-2 Plex condition flags *(continued)*

Condition flag	Description
RECOVER	A disk corresponding to one of the disk media records was replaced, or was reattached too late to prevent the plex from becoming out-of-date with respect to the volume. The plex required complete recovery from another plex in the volume to synchronize its contents.
REMOVED	Set in the disk media record when one of the subdisks associated with the plex is removed. The plex cannot be used until this condition is fixed, or the affected subdisk is dissociated.

Plex kernel states

The plex kernel state indicates the accessibility of the plex to the volume driver which monitors it.

No user intervention is required to set these states; they are maintained internally. On a system that is operating properly, all plexes are enabled.

Table 7-3 shows the possible plex kernel states.

Table 7-3 Plex kernel states

Kernel state	Description
DETACHED	Maintenance is being performed on the plex. Any write request to the volume is not reflected in the plex. A read request from the volume is not satisfied from the plex. Plex operations and `ioctl` function calls are accepted.
DISABLED	The plex is offline and cannot be accessed.
ENABLED	The plex is online. A write request to the volume is reflected in the plex. A read request from the volume is satisfied from the plex. If a plex is sparse, this is indicated by the `SPARSE` modifier being displayed in the output from the `vxprint -t` command.

Attaching and associating plexes

A plex becomes a participating plex for a volume by attaching it to a volume. (Attaching a plex associates it with the volume and enables the plex for use.) To attach a plex to an existing volume, use the following command:

```
# vxplex [-g diskgroup] att volume plex
```

For example, to attach a plex named vol01-02 to a volume named vol01 in the disk group, mydg, use the following command:

```
# vxplex -g mydg att vol01 vol01-02
```

If the volume does not already exist, associate one or more plexes to the volume when you create the volume, using the following command:

```
# vxmake [-g diskgroup] -U usetype vol volume plex=plex1[,plex2...]
```

For example, to create a mirrored, fsgen-type volume named home, and to associate two existing plexes named home-1 and home-2 with home, use the following command:

```
# vxmake -g mydg -U fsgen vol home plex=home-1,home-2
```

You can also use the command vxassist mirror *volume* to add a data plex as a mirror to an existing volume.

Taking plexes offline

Once a volume has been created and placed online (ENABLED), VxVM can temporarily disconnect plexes from the volume. This is useful, for example, when the hardware on which the plex resides needs repair or when a volume has been left unstartable and a source plex for the volume revive must be chosen manually.

Resolving a disk or system failure includes taking a volume offline and attaching and detaching its plexes. The two commands used to accomplish disk failure resolution are vxmend and vxplex.

To take a plex OFFLINE so that repair or maintenance can be performed on the physical disk containing subdisks of that plex, use the following command:

```
# vxmend [-g diskgroup] off plex
```

If a disk fails (for example, it has a head crash), use the vxmend command to take offline all plexes that have associated subdisks on the affected disk. For example, if plexes vol01-02 and vol02-02 in the disk group, mydg, had subdisks on a drive to be repaired, use the following command to take these plexes offline:

```
# vxmend -g mydg off vol01-02 vol02-02
```

This command places vol01-02 and vol02-02 in the OFFLINE state, and they remain in that state until it is changed. The plexes are not automatically recovered on rebooting the system.

Detaching plexes

To temporarily detach one data plex in a mirrored volume, use the following command:

```
# vxplex [-g diskgroup] det plex
```

For example, to temporarily detach a plex named `vol01-02` in the disk group, `mydg`, and place it in maintenance mode, use the following command:

```
# vxplex -g mydg det vol01-02
```

This command temporarily detaches the plex, but maintains the association between the plex and its volume. However, the plex is not used for I/O. A plex detached with the preceding command is recovered at system reboot. The plex state is set to `STALE`, so that if a `vxvol start` command is run on the appropriate volume (for example, on system reboot), the contents of the plex is recovered and made `ACTIVE`.

When the plex is ready to return as an active part of its volume, it can be reattached to the volume.

See "Reattaching plexes" on page 300.

Reattaching plexes

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 Volume Manager Troubleshooting Guide*.

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 the disk groups imported on the node are monitored. If the automatic reattachment functionality is disabled on a master node, the feature is disabled 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 495.

Moving plexes

Moving a plex copies the data content from the original plex onto a new plex. To move a plex, use the following command:

```
# vxplex [-g diskgroup] mv original_plex new_plex
```

For a move task to be successful, the following criteria must be met:

- The old plex must be an active part of an active (`ENABLED`) volume.
- The new plex must be at least the same size or larger than the old plex.
- The new plex must not be associated with another volume.

The size of the plex has several implications:

- If the new plex is smaller or more sparse than the original plex, an incomplete copy is made of the data on the original plex. If an incomplete copy is desired, use the `-o force` option to `vxplex`.
- If the new plex is longer or less sparse than the original plex, the data that exists on the original plex is copied onto the new plex. Any area that is not on the original plex, but is represented on the new plex, is filled from other complete plexes associated with the same volume.
- If the new plex is longer than the volume itself, then the remaining area of the new plex above the size of the volume is not initialized and remains unused.

Copying volumes to plexes

This task copies the contents of a volume onto a specified plex. The volume to be copied must not be enabled. The plex cannot be associated with any other volume. To copy a plex, use the following command:

```
# vxplex [-g diskgroup] cp volume new_plex
```

After the copy task is complete, *new_plex* is not associated with the specified volume *volume*. The plex contains a complete copy of the volume data. The plex that is being copied should be the same size or larger than the volume. If the plex being copied is larger than the volume, an incomplete copy of the data results. For the same reason, *new_plex* should not be sparse.

Dissociating and removing plexes

When a plex is no longer needed, you can dissociate it from its volume and remove it as an object from VxVM. You might want to remove a plex for the following reasons:

- to provide free disk space
- to reduce the number of mirrors in a volume so you can increase the length of another mirror and its associated volume. When the plexes and subdisks are removed, the resulting space can be added to other volumes
- to remove a temporary mirror that was created to back up a volume and is no longer needed
- to change the layout of a plex

To save the data on a plex to be removed, the configuration of that plex must be known. Parameters from that configuration (stripe unit size and subdisk ordering)

are critical to the creation of a new plex to contain the same data. Before a plex is removed, you must record its configuration.

See "Displaying plex information" on page 293."

To dissociate a plex from the associated volume and remove it as an object from VxVM, use the following command:

```
# vxplex [-g diskgroup] -o rm dis plex
```

For example, to dissociate and remove a plex named `vol01-02` in the disk group, `mydg`, use the following command:

```
# vxplex -g mydg -o rm dis vol01-02
```

This command removes the plex `vol01-02` and all associated subdisks.

Alternatively, you can first dissociate the plex and subdisks, and then remove them with the following commands:

```
# vxplex [-g diskgroup] dis plex
# vxedit [-g diskgroup] -r rm plex
```

When used together, these commands produce the same result as the `vxplex -o rm dis` command. The `-r` option to `vxedit rm` recursively removes all objects from the specified object downward. In this way, a plex and its associated subdisks can be removed by a single `vxedit` command.

Changing plex attributes

Warning: To avoid possible data loss, change plex attributes with extreme care.

The `vxedit` command changes the attributes of plexes and other Volume Manager objects. To change plex attributes, use the following command:

```
# vxedit [-g diskgroup] set attribute=value ... plex
```

Plex fields that can be changed using the `vxedit` command include:

- `name`
- *`putiln`*
- *`tutiln`*
- `comment`

The following example command sets the comment field, and also sets `tutil2` to indicate that the subdisk is in use:

```
# vxedit -g mydg set comment="plex comment" tutil2="u" vol01-02
```

To prevent a particular plex from being associated with a volume, set the `putil0` field to a non-null string, as shown in the following command:

```
# vxedit -g mydg set putil0="DO-NOT-USE" vol01-02
```

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

Chapter 8

Creating volumes

This chapter includes the following topics:

- Initializing and starting a volume
- Accessing a volume
- Using rules and persistent attributes to make volume allocation more efficient

About volume creation

Volumes are logical devices that appear as physical disk partition devices to data management systems. Volumes enhance recovery from hardware failure, data availability, performance, and storage configuration.

Volumes are created to take advantage of the VxVM concept of virtual disks. A file system can be placed on the volume to organize the disk space with files and directories. In addition, you can configure applications such as databases to organize data on volumes.

Disks and disk groups must be initialized and defined to VxVM before volumes can be created from them.

See "About disk management" on page 76.

See "About disk groups" on page 220.

Types of volume layouts

VxVM allows you to create volumes with the following layout types:

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 35.
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 37.

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 40.
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 43.
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 41.
Layered Volume	A volume constructed from other volumes. Non-layered volumes are constructed by mapping their subdisks to VM 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. Examples of layered volumes are striped-mirror and concatenated-mirror volumes. See "Layered volumes" on page 48. 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 42. 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.

Supported volume logs and maps

Veritas Volume Manager supports the use of the following types of logs and maps with volumes:

- FastResync Maps are used to perform quick and efficient resynchronization of mirrors.
  See "FastResync" on page 61.

  These maps are supported either in memory (Non-Persistent FastResync), or on disk as part of a DCO volume (Persistent FastResync). Two types of DCO volume are supported:

  - Version 0 DCO volumes only support Persistent FastResync for the traditional third-mirror break-off type of volume snapshot.
    See "Version 0 DCO volume layout" on page 64.
    See "Creating a volume with a version 0 DCO volume" on page 325.
  - Version 20 DCO volumes, introduced in VxVM 4.0, support DRL logging (see below) and Persistent FastResync for full-sized and space-optimized instant volume snapshots.
    See "Version 20 DCO volume layout" on page 64.
    See "Creating a volume with a version 20 DCO volume" on page 328.
    See "Enabling FastResync on a volume" on page 385.

- Dirty region logs allow the fast recovery of mirrored volumes after a system crash.
  See "Dirty region logging" on page 56.
  These logs are supported either as DRL log plexes, or as part of a version 20 DCO volume. Refer to the following sections for information on creating a volume on which DRL is enabled:
  See "Creating a volume with dirty region logging enabled" on page 328.
  See "Creating a volume with a version 20 DCO volume" on page 328.

- RAID-5 logs are used to prevent corruption of data during recovery of RAID-5 volumes.
  See "RAID-5 logging" on page 48.
  These logs are configured as plexes on disks other than those that are used for the columns of the RAID-5 volume.
  See "Creating a RAID-5 volume" on page 333.

Creating a volume

You can create volumes using an advanced approach or an assisted approach. Each method uses different tools. You may switch between the advanced and the assisted approaches at will.

Note: Most VxVM commands require superuser or equivalent privileges.

Advanced approach

The advanced approach consists of a number of commands that typically require you to specify detailed input. These commands use a "building block" approach that requires you to have a detailed knowledge of the underlying structure and components to manually perform the commands necessary to accomplish a certain task. Advanced operations are performed using several different VxVM commands.

To create a volume using the advanced approach, perform the following steps in the order specified:

- Create subdisks using `vxmake sd`.
  See "Creating subdisks" on page 284.
- Create plexes using `vxmake plex`, and associate subdisks with them.
  See "Creating plexes" on page 293.
  See "Associating subdisks with plexes" on page 287.
- Associate plexes with the volume using `vxmake vol`.
- Initialize the volume using `vxvol start` or `vxvol init zero`.
  See "Initializing and starting a volume created using vxmake" on page 339.

The steps to create the subdisks and plexes, and to associate the plexes with the volumes can be combined by using a volume description file with the `vxmake` command.

See "Creating a volume using a vxmake description file" on page 337.

See "Creating a volume using vxmake" on page 336.

Assisted approach

The assisted approach takes information about what you want to accomplish and then performs the necessary underlying tasks. This approach requires only minimal input from you, but also permits more detailed specifications.

Assisted operations are performed primarily through the `vxassist` command. `vxassist` creates the required plexes and subdisks using only the basic attributes

of the desired volume as input. Additionally, the vxassist command can modify existing volumes while automatically modifying any underlying or associated objects.

The vxassist command uses default values for many volume attributes, unless you provide specific values. It does not require you to have a thorough understanding of low-level VxVM concepts, vxassist does not conflict with other VxVM commands or preclude their use. Objects created by vxassist are compatible and inter-operable with objects created by other VxVM commands and interfaces.

Using vxassist

You can use the vxassist utility to create and modify volumes. Specify the basic requirements for volume creation or modification, and vxassist performs the necessary tasks.

The advantages of using vxassist rather than the advanced approach include:

- Most actions require that you enter only one command rather than several.
- You are required to specify only minimal information to vxassist. If necessary, you can specify additional parameters to modify or control its actions.
- Operations result in a set of configuration changes that either succeed or fail as a group, rather than individually. System crashes or other interruptions do not leave intermediate states that you have to clean up. If vxassist finds an error or an exceptional condition, it exits after leaving the system in the same state as it was prior to the attempted operation.

The vxassist utility helps you perform the following tasks:

- Creating volumes.
- Creating mirrors for existing volumes.
- Growing or shrinking existing volumes.
- Backing up volumes online.
- Reconfiguring a volume's layout online.

vxassist obtains most of the information it needs from sources other than your input. vxassist obtains information about the existing objects and their layouts from the objects themselves.

For tasks requiring new disk space, vxassist 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 takes this form:

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

where *keyword* selects the task to perform. The first argument after a vxassist keyword, *volume*, is a volume name, which is followed by a set of desired volume attributes. For example, the keyword make allows you to create a new volume:

```
# vxassist [options] make volume length [attributes]
```

The length of the volume can be specified in sectors, kilobytes, megabytes, or gigabytes by using a suffix character of s, k, m, or g. If no suffix is specified, the size is assumed to be in sectors.

See the vxintro(1M) manual page.

Additional attributes can be specified as appropriate, depending on the characteristics that you wish the volume to have. Examples are stripe unit width, number of columns in a RAID-5 or stripe volume, number of mirrors, number of logs, and log type.

By default, the vxassist command creates volumes in a default disk group according to a set of rules.

See "Rules for determining the default disk group" on page 222.

To use a different disk group, specify the *-g diskgroup* option to vxassist.

A large number of vxassist keywords and attributes are available for use.

See the vxassist(1M) manual page.

The simplest way to create a volume is to use default attributes.

See "Creating a volume on any disk" on page 316.

More complex volumes can be created with specific attributes by controlling how vxassist uses the available storage space.

See "Creating a volume on specific disks" on page 317.

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 the SmartMove™ feature while attaching a plex

The SmartMove™ feature reduces the time and I/O required to attach or reattach a plex to an existing VxVM volume, in the specific case where a VxVM volume has a VxFS file system mounted on it. The SmartMove feature uses the VxFS information to detect free extents and avoid copying them.

The SmartMove feature is enabled by default.

The SmartMove feature takes effect when a plex is attached or reattached using the `vxplex`, `vxsd`, or `vxassist` commands.

Note: The file system must be mounted to get the benefits of the SmartMove™ feature.

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.

The SmartMove feature enables migration from a traditional LUN to a thinly provisioned LUN, removing unused space in the process.

For more information, see the section on migrating to thin provisioning in the *Veritas Storage Foundation™ Advanced Features Administrator's Guide*.

Discovering the maximum size of a volume

To find out how large a volume you can create within a disk group, use the following form of the `vxassist` command:

```
# vxassist [-g diskgroup] maxsize layout=layout [attributes]
```

For example, to discover the maximum size RAID-5 volume with 5 columns and 2 logs that you can create within the disk group, `dgrp`, enter the following command:

```
# vxassist -g dgrp maxsize layout=raid5 nlog=2
```

You can use storage attributes if you want to restrict the disks that `vxassist` uses when creating volumes.

See "Creating a volume on specific disks" on page 317.

The maximum size of a VxVM volume that you can create is 256TB.

Disk group alignment constraints on volumes

Certain constraints apply to the length of volumes and to the numeric values of size attributes that apply to volumes. If a volume is created in a disk group that is compatible with the Cross-platform Data Sharing (CDS) feature, the volume's length and the values of volume attributes that define the sizes of objects such as logs or stripe units, must be an integer multiple of the alignment value of 16 blocks (8 kilobytes) . If the disk group is not compatible with the CDS feature, the volume's length and attribute size values must be multiples of 1 block (512 bytes).

To discover the value in blocks of the alignment that is set on a disk group, use this command:

```
# vxprint -g diskgroup -G -F %align
```

By default, `vxassist` automatically rounds up the volume size and attribute size values to a multiple of the alignment value. (This is equivalent to specifying the attribute `dgalign_checking=round` as an additional argument to the `vxassist` command.)

If you specify the attribute `dgalign_checking=strict` to `vxassist`, the command fails with an error if you specify a volume length or attribute size value that is not a multiple of the alignment value for the disk group.

Creating a volume on any disk

By default, the `vxassist make` command creates a concatenated volume that uses one or more sections of disk space. On a fragmented disk, this allows you to put together a volume larger than any individual section of free disk space available.

To change the default layout, edit the definition of the `layout` attribute defined in the `/etc/default/vxassist` file.

If there is not enough space on a single disk, `vxassist` creates a spanned volume. A spanned volume is a concatenated volume with sections of disk space spread across more than one disk. A spanned volume can be larger than any disk on a system, since it takes space from more than one disk.

To create a concatenated, default volume, use the following form of the `vxassist` command:

```
# vxassist [-b] [-g diskgroup] make volume length
```

Specify the `-b` option if you want to make the volume immediately available for use.

See "Initializing and starting a volume" on page 338.

For example, to create the concatenated volume `voldefault` with a length of 10 gigabytes in the default disk group:

```
# vxassist -b make voldefault 10g
```

Creating a volume on specific disks

VxVM automatically selects the disks on which each volume resides, unless you specify otherwise. 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.

See "Initializing and starting a volume" on page 338.

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 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 `dgrp07` and `dgrp08` when calculating the maximum size of RAID-5 volume that `vxassist` can create using the disks in the disk group `dg`:

```
# vxassist -b -g dgrp maxsize layout=raid5 nlog=2 \!dgrp07 \!dgrp08
```

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 320.

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

`vxassist` also lets you select disks based on disk tags. The following command only includes disks that have a `tier1` disktag.

```
# vxassist -g dg3 make vol3 1g disktag:tier1
```

Creating a volume on SSD devices

This section explains how to create a volume on Solid State Disk (SSD) device.

You must upgrade the disk group to version 150 or higher for SSD support. To upgrade the disk group, use the following command:

```
# vxdg upgrade diskgroup
```

where *`diskgroup`* is the name of the disk group to which the disk belongs.

The allocation behavior of the `vxassist` command changes with the presence of SSD devices in a disk group.

Note: If the disk group version is less than 150, the `vxassist` command does not honor media type of the device for making allocations.

The `vxassist` command allows you to specify Hard Disk Drive (HDD) or SSD devices for allocation using the `mediatype` attribute. For example, to create a volume `myvol` of size 1g on SSD disks in `mydg`, use the following command:

```
# vxassist -g mydg make myvol 1g mediatype:ssd
```

For example, to create a volume `myvol` of size 1g on HDD disks in `mydg`, use the following command:

```
# vxassist -g mydg make myvol 1g mediatype:hdd
```

If neither `mediatype:hdd` nor `mediatype:ssd` is specified, then `mediatype:hdd` is considered as default selection type which means only the HDD devices present in the disk group are considered for allocation.

If a mix of SSD devices and HDD devices are specified, the allocation is done only on HDD devices unless mediatype:ssd is explicitly specified. For example:

```
enclr1 : enclosure having all SSD devices
enclr2 : enclosure having all HDD devices
enclr3 : enclosure having mix of SSD and HDD devices
```

In the following command, volume *`myvol`* of size 1G is allocated on devices from `enclr2` array (only HDD devices):

```
# vxassist -g mydg make myvol 1G enclr:enclr1 enclr:enclr2
```

In order to create a volume on SSD devices from `enclr1` enclosure, following command should be used:

```
# vxassist -g mydg make myvol 1G enclr:enclr1 mediatype:ssd
```

If `enclr3` is only specified, only hdd devices present in enclr3 are considered for allocation.

In the following two commands, volume myvol of size 1G is allocated on HDD devices from enclr3 array:

```
# vxassist -g mydg make myvol 1G enclr:enclr3 mediatype:hdd
# vxasisst -g mydg make myvol 1G enclr:enclr3
```

In order to allocate a volume on SSD devices from enclr3 enclosure, following command should be used:

```
# vxassist -g mydg make myvol 1G enclr:enclr3 mediatype:ssd
```

The allocation fails, if the command is specified in one of the following two ways:

```
# vxassist -g mydg make myvol 1G enclr:enclr1 mediatype:hdd
```

In the above case, volume `myvol` cannot be created as there are no HDD devices in `enclr1` enclosure.

```
# vxassist -g mydg make myvol 1G enclr:enclr2 mediatype:ssd
```

In the above case, volume myvol cannot be created as there are no SSD devices in `enclr2` enclosure.

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.

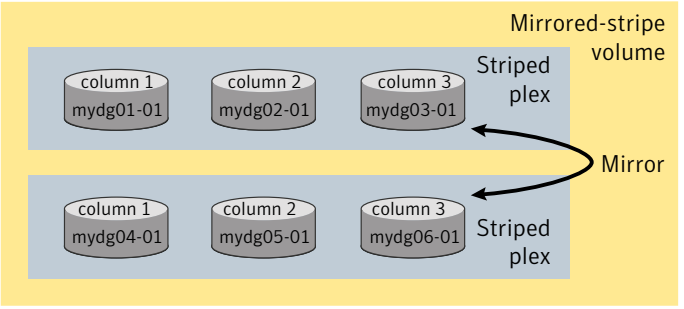
For example, the following command creates a mirrored-stripe volume with 3 columns and 2 mirrors on 6 disks in the disk group, `mydg`:

```
# 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 8-1 shows an example of using ordered allocation to create a mirrored-stripe volume.

Figure 8-1 Example of using ordered allocation to create a mirrored-stripe volume



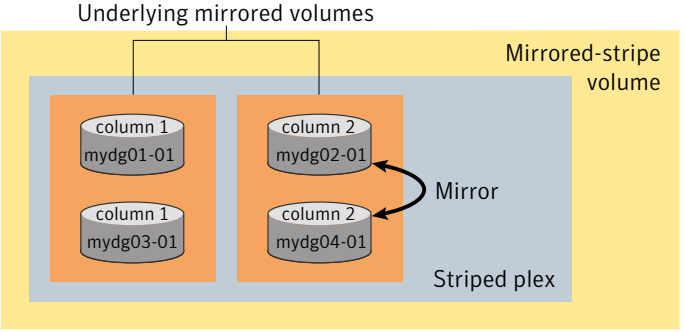
For layered volumes, `vxassist` applies the same rules to allocate storage as for non-layered volumes. For example, the following command creates a striped-mirror volume with 2 columns:

```
# 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 8-2 shows an example of using ordered allocation to create a striped-mirror volume.

Figure 8-2 Example of using ordered allocation to create a striped-mirror volume



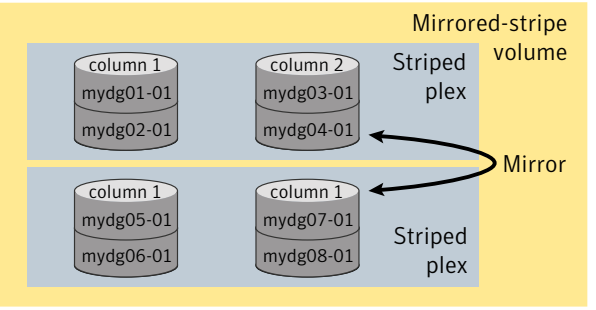
Additionally, you can use the `col_switch` attribute to specify how to concatenate space on the disks into columns. For example, the following command creates a mirrored-stripe volume with 2 columns:

```
# 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 8-3 shows an example of using concatenated disk space to create a mirrored-stripe volume.

Figure 8-3 Example of using concatenated disk space to create a mirrored-stripe volume



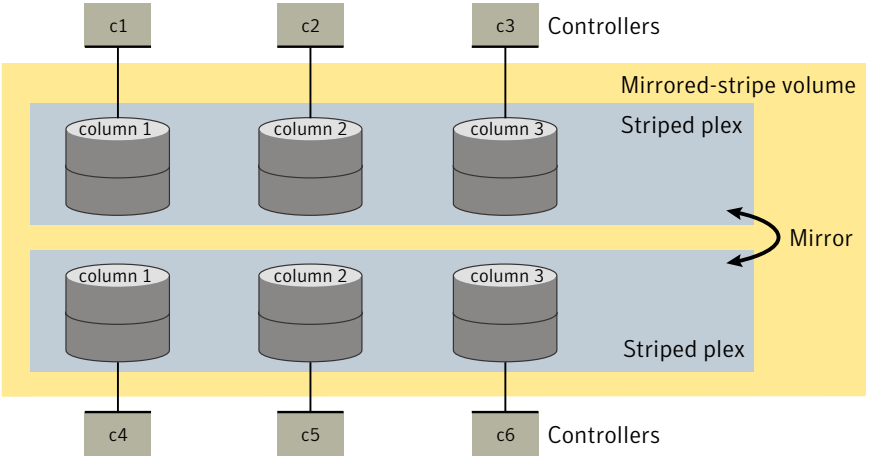
Other storage specification classes for controllers, enclosures, targets and trays can be used with ordered allocation. For example, the following command creates a 3-column mirrored-stripe volume between specified controllers:

```
# 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 8-4 shows an example of using storage allocation to create a mirrored-stripe volume across controllers.

Figure 8-4 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.

See "Mirroring across targets, controllers or enclosures" on page 331.

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, use 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.

See "Initializing and starting a volume" on page 338.

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
```

To create a volume with 3 instead of the default of 2 mirrors, modify the command to read:

```
# 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 concatenated-mirror volume, use 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.

See "Initializing and starting a volume" on page 338.

Alternatively, first create a concatenated volume, and then mirror it.

See "Adding a mirror to a volume " on page 367.

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, use 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.

See "Initializing and starting a volume" on page 338.

Creating a volume with a version 0 DCO volume

If a data change object (DCO) and DCO volume are associated with a volume, this allows Persistent FastResync to be used with the volume.

See "How persistent FastResync works with snapshots" on page 65.

The version 0 data change object (DCO) and DCO volume layout was introduced in VxVM 3.2. The version 0 layout supports traditional (third-mirror) snapshots, but not full-sized instant snapshots, space-optimized instant snapshots nor DRL configured within the DCO volume.

See "Determining the DCO version number" on page 374.

See "Version 0 DCO volume layout" on page 64.

See "Version 20 DCO volume layout" on page 64.

See "Creating a volume with a version 20 DCO volume" on page 328.

To perform fast resynchronization of mirrors after a system crash or reboot, you must also enable dirty region logging (DRL) on a mirrored volume.

For more information about snapshots and DCO volumes, see the *Veritas Storage Foundation Advanced Features Administrator's Guide*.

Note: You need a license to use the Persistent FastResync feature. 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. However, without a license, only full resynchronization can be performed.

See "How persistent FastResync works with snapshots" on page 65.

To create a volume with an attached version 0 DCO object and volume

1 Ensure that the disk group has been upgraded to at least version 90. Use the following command to check the version of a disk group:

```
# vxdg list diskgroup
```

To upgrade a disk group to the latest version, use the following command:

```
# vxdg upgrade diskgroup
```

See "Upgrading the disk group version" on page 276.

2 Use the following command to create the volume (you may need to specify additional attributes to create a volume with the desired characteristics):

```
# vxassist [-g diskgroup] make volume length layout=layout \
  logtype=dco [ndcomirror=number] [dcolen=size] \
  [fastresync=on] [other 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 data plexes in the volume. For example, specify `ndcomirror=3` when creating a 3-way mirrored volume.

The default size of each plex is 132 blocks unless you use the `dcolen` attribute to specify a different size. If specified, the size of the plex must be a multiple of 33 blocks from 33 up to a maximum of 2112 blocks.

By default, FastResync is not enabled on newly created volumes. Specify the `fastresync=on` attribute if you want to enable FastResync on the volume. If a DCO object and DCO volume are associated with the volume, Persistent FastResync is enabled; otherwise, Non-Persistent FastResync is enabled.

3 To enable DRL or sequential DRL logging on the newly created volume, use the following command:

```
# vxvol [-g diskgroup] set logtype=drl|drlseq volume
```

If you use ordered allocation when creating a mirrored volume on specified storage, you can use the optional `logdisk` attribute to specify on which disks dedicated 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 [-g diskgroup] -o ordered make volume length \
  layout=mirror logtype=log_type logdisk=disk[,disk,...] \
  storage_attributes
```

If you do not specify the `logdisk` attribute, `vxassist` locates the logs in the data plexes of the volume.

See "Specifying ordered allocation of storage to volumes" on page 320.

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

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

Creating a volume with a version 20 DCO volume

To create a volume with an attached version 20 DCO object and volume

1. Ensure that the disk group has been upgraded to the latest version. Use the following command to check the version of a disk group:

   ```
   # vxdg list diskgroup
   ```

   To upgrade a disk group to the most recent version, use the following command:

   ```
   # vxdg upgrade diskgroup
   ```

   See "Upgrading the disk group version" on page 276.

2. Use the following command to create the volume (you may need to specify additional attributes to create a volume with the desired characteristics):

   ```
   # vxassist [-g diskgroup] make volume length layout=layout \
     logtype=dco dcoversion=20 [drl=on|sequential|off] \
     [ndcomirror=number] [fastresync=on] [other attributes]
   ```

   Set the value of the `drl` attribute to `on` if dirty region logging (DRL) is to be used with the volume (this is the default setting). For a volume that will be written to sequentially, such as a database log volume, set the value to `sequential` to enable sequential DRL. The DRL logs are created in the DCO volume. The redundancy of the logs is determined by the number of mirrors that you specify using the `ndcomirror` attribute.

   By default, Persistent FastResync is not enabled on newly created volumes. Specify the `fastresync=on` attribute if you want to enable Persistent FastResync on the volume.

   See "Determining the DCO version number" on page 374.

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

Creating a volume with dirty region logging enabled

Dirty region logging (DRL), if enabled, speeds recovery of mirrored volumes after a system crash. To enable DRL on a volume that is created within a disk group with a version number between 20 and 100, specify the `logtype=drl` attribute to the `vxassist make` command as shown in this example usage:

```
# vxassist [-g diskgroup] make volume length layout=layout \
  logtype=drl [nlog=n] [loglen=size] [other attributes]
```

The `nlog` attribute can be used to specify the number of log plexes to add. By default, one log plex is added. The `loglen` attribute specifies the size of the log, where each bit represents one region in the volume. For example, the size of the log would need to be 20K for a 10GB volume with a region size of 64 kilobytes.

For example, to create a mirrored 10GB volume, `vol02`, with two log plexes in the disk group, `mydg`, use the following command:

```
# vxassist -g mydg make vol02 10g layout=mirror logtype=drl \
  nlog=2 nmirror=2
```

Sequential DRL limits the number of dirty regions for volumes that are written to sequentially, such as database replay logs. To enable sequential DRL on a volume that is created within a disk group with a version number between 70 and 100, specify the `logtype=drlseq` attribute to the `vxassist make` command.

```
# vxassist [-g diskgroup] make volume length layout=layout \
  logtype=drlseq [nlog=n] [other attributes]
```

It is also possible to enable the use of Persistent FastResync with this volume.

See "Creating a volume with a version 0 DCO volume" on page 325.

Note: Operations on traditional DRL log plexes are usually applicable to volumes that are created in disk groups with a version number of less than 110. If you enable DRL or sequential DRL on a volume that is created within a disk group with a version number of 110 or greater, the DRL logs are usually created within the plexes of a version 20 DCO volume.

See "Creating a volume with a version 20 DCO volume" on page 328.

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 37.

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.

See "Initializing and starting a volume" on page 338.

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 striped-mirror 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.

See "Initializing and starting a volume" on page 338.

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 367.

Creating a striped-mirror volume

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

See "Initializing and starting a volume" on page 338.

By default, 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`.

Mirroring across targets, controllers or enclosures

To create a volume whose mirrored data plexes lie on different controllers (also known as disk duplexing) or in different enclosures, use the `vxassist` command as described in this section.

In the following command, the attribute `mirror=target` specifies that volumes should be mirrored between targets on different controllers.

```
# vxassist [-b] [-g diskgroup] make volume length \
  layout=layout mirror=target [attributes]
```

Specify the `-b` option if you want to make the volume immediately available for use.

See "Initializing and starting a volume" on page 338.

The attribute `mirror=ctlr` specifies that disks in one mirror should not be on the same controller as disks in other mirrors within the same volume:

```
# vxassist [-b] [-g diskgroup] make volume length \
  layout=layout mirror=ctlr [attributes]
```

Note: Both paths of an active/passive array are not considered to be on different controllers when mirroring across controllers.

The following command creates a mirrored volume with two data plexes in the disk group, `mydg`:

```
# vxassist -b -g mydg make volspec 10g layout=mirror nmirror=2 \
  mirror=ctlr ctlr:c2 ctlr:c3
```

The disks in one data plex are all attached to controller `c2`, and the disks in the other data plex are all attached to controller `c3`. This arrangement ensures continued availability of the volume should either controller fail.

The attribute `mirror=enclr` specifies that disks in one mirror should not be in the same enclosure as disks in other mirrors within the same volume.

The following command creates a mirrored volume with two data plexes:

```
# vxassist -b make -g mydg volspec 10g layout=mirror nmirror=2 \
  mirror=enclr enclr:enc1 enclr:enc2
```

The disks in one data plex are all taken from enclosure `enc1`, and the disks in the other data plex are all taken from enclosure `enc2`. This arrangement ensures continued availability of the volume should either enclosure become unavailable.

There are other ways in which you can control how volumes are laid out on the specified storage.

See "Specifying ordered allocation of storage to volumes" on page 320.

Mirroring across media types (SSD and HDD)

This section describes how to mirror across media types (SSD and HDD).

To create a volume with a HDD plex and a SSD plex

1 Create a volume with media type HDD:

```
# vxassist -g mydg make myvol 1G mediatype:hdd
```

2 Add a mirror to the volume with media type SSD:

```
# vxassist -g mydg mirror myvol mediatype:ssd
```

Note: `mirror=mediatype` is not supported.

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: 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. Both approaches are described below.

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 43.

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.

See "Initializing and starting a volume" on page 338.

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  volumelength \
  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 320.

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

It is possible to add more logs to a RAID-5 volume at a later time.

See “Adding a RAID-5 log” on page 394.

Creating tagged volumes

Volume tags are used to implement the SmartTier feature of the Storage Foundation software.

See the *Veritas Storage Foundation Advanced Features Administrator’s Guide*.

You can use the `tag` attribute with the `vxassist make` command to set a named tag and optional tag value on a volume, for example:

```
# vxassist -b -g mydg make volmir 5g layout=mirror tag=mirvol=5g
```

To list the tags that are associated with a volume, use this command:

```
# vxassist [-g diskgroup] listtag volume
```

If you do not specify a volume name, the tags of all volumes and vsets in the disk group are listed.

The following is an example of listtag output:

```
# vxassist -g dgl listtag vol
TY   NAME              DISKGROUP         TAG
================================================
v    vol               dg1               Symantec
```

To list the volumes that have a specified tag name, use this command:

```
# vxassist [-g diskgroup] list tag=tagname
```

Tag names and tag values are case-sensitive character strings of up to 256 characters. Tag names can consist of letters (A through Z and a through z), numbers (0 through 9), dashes (-), underscores (_) or periods (.) from the ASCII character set. A tag name must start with either a letter or an underscore. Tag values can consist of any character from the ASCII character set with a decimal value from 32 through 127. If a tag value includes any spaces, use the `vxassist settag` command to set the tag on the newly created volume.

Dotted tag hierarchies are understood by the `list` operation. For example, the listing for `tag=a.b` includes all volumes that have tag names that start with `a.b`.

The tag names `site`, `udid` and `vdid` are reserved and should not be used. To avoid possible clashes with future product features, it is recommended that tag names do not start with any of the following strings: `asl`, `be`, `isp`, `nbu`, `sf`, `symc`, `vx`, or `vxvm.attr`.

See "Setting tags on volumes" on page 380.

You can use the `tier` attribute with the `vxassist make` command to set the `vxfs.placement_class.` tag on the volume being created. This attribute lets you set the volume tier that SmartTier will use. The following command adds `tier1` to the `vxfs.placement_class tag`.

```
# vxassist  -g dg3 make vol4 5g tier=tier2
```

The following command confirms that the vxfs.placement_class tag has been updated.

```
# vxassist  -g dg3 listtag

TY  NAME             DISKGROUP        TAG
============================================================
v   vol4             dg3              vxfs.placement_class.tier2
```

Creating a volume using vxmake

As an alternative to using vxassist, you can create a volume using the vxmake command to arrange existing subdisks into plexes, and then to form these plexes into a volume.

See "Creating subdisks" on page 284.

The example given in this section is to create a RAID-5 volume using vxmake.

Creating a RAID-5 plex for a RAID-5 volume is similar to creating striped plexes, except that the layout attribute is set to raid5. Subdisks can be implicitly associated in the same way as with striped plexes. For example, to create a four-column RAID-5 plex with a stripe unit size of 32 sectors, use the following command:

```
# vxmake -g mydg plex raidplex layout=raid5 stwidth=32 \
  sd=mydg00-01,mydg01-00,mydg02-00,mydg03-00
```

Note that because four subdisks are specified, but the number of columns is not specified, the vxmake command assumes a four-column RAID-5 plex and places one subdisk in each column. Striped plexes are created using the same method except that the layout is specified as stripe. If the subdisks are to be created and added later, use the following command to create the plex:

```
# vxmake -g mydg plex raidplex layout=raid5 ncolumn=4 stwidth=32
```

If no subdisks are specified, the ncolumn attribute must be specified. Subdisks can be added to the plex later using the vxsd assoc command.

See "Associating subdisks with plexes" on page 287.

If each column in a RAID-5 plex is to be created from multiple subdisks which may span several physical disks, you can specify to which column each subdisk should be added. For example, to create a three-column RAID-5 plex using six subdisks, use the following form of the vxmake command:

```
# vxmake -g mydg plex raidplex layout=raid5 stwidth=32 \
  sd=mydg00-00:0,mydg01-00:1,mydg02-00:2,mydg03-00:0, \
  mydg04-00:1,mydg05-00:2
```

This command stacks subdisks `mydg00-00` and `mydg03-00` consecutively in column 0, subdisks `mydg01-00` and `mydg04-00` consecutively in column 1, and subdisks `mydg02-00` and `mydg05-00` in column 2. Offsets can also be specified to create sparse RAID-5 plexes, as for striped plexes.

Log plexes may be created as default concatenated plexes by not specifying a layout, for example:

```
# vxmake -g mydg plex raidlog1 sd=mydg06-00
  # vxmake -g mydg plex raidlog2 sd=mydg07-00
```

The following command creates a RAID-5 volume, and associates the prepared RAID-5 plex and RAID-5 log plexes with it:

```
# vxmake -g mydg -Uraid5 vol raidvol \
  plex=raidplex,raidlog1,raidlog2
```

Each RAID-5 volume has one RAID-5 plex where the data and parity are stored. Any other plexes associated with the volume are used as RAID-5 log plexes to log information about data and parity being written to the volume.

After creating a volume using `vxmake`, you must initialize it before it can be used.

See "Initializing and starting a volume" on page 338.

Creating a volume using a vxmake description file

You can use the `vxmake` command to add a new volume, plex or subdisk to the set of objects managed by VxVM. `vxmake` adds a record for each new object to the VxVM configuration database. You can create records either by specifying parameters to `vxmake` on the command line, or by using a file which contains plain-text descriptions of the objects. The file can also contain commands for performing a list of tasks. Use the following form of the command to have `vxmake` read the file from the standard input:

```
# vxmake [-g diskgroup] < description_file
```

Alternatively, you can specify the file to `vxmake` using the `-d` option:

```
# vxmake [-g diskgroup] -d description_file
```

The following sample description file defines a volume, `db`, with two plexes, `db-01` and `db-02`:

```
#rty #name       #options
sd   mydg03-01   disk=mydg03 offset=0 len=10000
sd   mydg03-02   disk=mydg03 offset=25000 len=10480
sd   mydg04-01   disk=mydg04 offset=0 len=8000
sd   mydg04-02   disk=mydg04 offset=15000 len=8000
sd   mydg04-03   disk=mydg04 offset=30000 len=4480
plex db-01       layout=STRIPE ncolumn=2 stwidth=16k
                 sd=mydg03-01:0/0,mydg03-02:0/10000,mydg04-01:1/0,
mydg04-02:1/8000,mydg04-03:1/16000
sd   ramd1-01    disk=ramd1 len=640
                 comment="Hot spot for dbvol"
plex db-02       sd=ramd1-01:40320
vol  db          usetype=gen plex=db-01,db-02
                 readpol=prefer prefname=db-02
                 comment="Uses mem1 for hot spot in last 5m"
```

The subdisk definition for plex, `db-01`, must be specified on a single line. It is shown here split across two lines because of space constraints.

The first plex, `db-01`, is striped and has five subdisks on two physical disks, `mydg03` and `mydg04`. The second plex, `db-02`, is the preferred plex in the mirror, and has one subdisk, `ramd1-01`, on a volatile memory disk.

For detailed information about how to use `vxmake`, refer to the `vxmake`(1M) manual page.

After creating a volume using `vxmake`, you must initialize it before it can be used.

See "Initializing and starting a volume created using vxmake" on page 339.

Initializing and starting a volume

If you create a volume using the `vxassist` command, `vxassist` initializes and starts the volume automatically unless you specify the attribute `init=none`.

When creating a volume, you can make it immediately available for use by specifying the `-b` option to the `vxassist` command, as shown here:

```
# vxassist -b [-g diskgroup] make volume length layout=mirror
```

The `-b` option makes VxVM carry out any required initialization as a background task. It also greatly speeds up the creation of striped volumes by initializing the columns in parallel.

As an alternative to the `-b` option, you can specify the `init=active` attribute to make a new volume immediately available for use. In this example, `init=active`

is specified to prevent VxVM from synchronizing the empty data plexes of a new mirrored volume:

```
# vxassist [-g diskgroup] make volume length layout=mirror \
  init=active
```

Warning: There is a very small risk of errors occurring when the `init=active` attribute is used. Although written blocks are guaranteed to be consistent, read errors can arise in the unlikely event that `fsck` attempts to verify uninitialized space in the file system, or if a file remains uninitialized following a system crash. If in doubt, use the `-b` option to `vxassist` instead.

This command writes zeroes to the entire length of the volume and to any log plexes. It then makes the volume active. You can also zero out a volume by specifying the attribute `init=zero` to `vxassist`, as shown in this example:

```
# vxassist [-g diskgroup] make volume length layout=raid5 \
  init=zero
```

You cannot use the `-b` option to make this operation a background task.

Initializing and starting a volume created using vxmake

A volume may be initialized by running the `vxvol` command if the volume was created by the `vxmake` command and has not yet been initialized, or if the volume has been set to an uninitialized state.

To initialize and start a volume, use the following command:

```
# vxvol [-g diskgroup] start volume
```

The following command can be used to enable a volume without initializing it:

```
# vxvol [-g diskgroup] init enable volume
```

This allows you to restore data on the volume from a backup before using the following command to make the volume fully active:

```
# vxvol [-g diskgroup] init active volume
```

If you want to zero out the contents of an entire volume, use this command to initialize it:

```
# vxvol [-g diskgroup] init zero volume
```

Accessing a volume

As soon as a volume has been created and initialized, it is available for use as a virtual disk partition by the operating system for the creation of a file system, or by application programs such as relational databases and other data management software.

Creating a volume in a disk group sets up block and character (raw) device files that can be used to access the volume:

`/dev/vx/dsk/`*`dg`*`/`*`vol`*	block device file for volume *vol* in disk group *dg*
`/dev/vx/rdsk/`*`dg`*`/`*`vol`*	character device file for volume *vol* in disk group *dg*

The pathnames include a directory named for the disk group. Use the appropriate device node to create, mount and repair file systems, and to lay out databases that require raw partitions.

As the `rootdg` disk group no longer has special significance, VxVM only creates volume device nodes for this disk group in the `/dev/vx/dsk/rootdg` and `/dev/vx/rdsk/rootdg` directories. VxVM does not create device nodes in the `/dev/vx/dsk` or `/dev/vx/rdsk` directories for the `rootdg` disk group.

Using rules and persistent attributes 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.

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.

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, it's 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.

Understanding persistent attributes

The `vxassist` command also 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. Useful attributes include 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
- `addlog`

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 even 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 which has proven too restrictive or discard it to allow a needed allocation to succeed.

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 which runs across several lines.

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

By default, a rule file is loaded from `/etc/default/vxsf_rules`. You can override this location in `/etc/default/vxassist`. You can also specify additional rule files on the command line.

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 tier1. The application of rule `tier1` was successful.

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

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'll use the following `vxsf_rules` files. 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 disk (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  -        -       -
```

Chapter 9

Administering volumes

This chapter includes the following topics:

About volume administration

Veritas Volume Manager (VxVM) lets you perform common maintenance tasks on volumes. These include the following:

- Displaying volume information
- Monitoring tasks
- Resizing volumes
- Adding and removing logs
- Adding and removing mirrors
- Removing volumes
- Changing the layout of volumes without taking them offline

Note: To use most VxVM commands, you need superuser or equivalent privileges.

Displaying volume information

You can use the `vxprint` command to display information about how a volume is configured.

To display the volume, plex, and subdisk record information for all volumes in the system, use the following command:

```
# vxprint -hvt
```

You can also apply the `vxprint` command to a single disk group:

```
# vxprint -g mydg -hvt
```

This example produces the following output:

```
V   NAME      RVG/VSET/CO KSTATE    STATE     LENGTH     READPOL      PREFPLEX    UTYPE
PL  NAME      VOLUME      KSTATE    STATE     LENGTH     LAYOUT       NCOL/WID    MODE
SD  NAME      PLEX        DISK      DISKOFFS  LENGTH     [COL/]OFF    DEVICE      MODE
SV  NAME      PLEX        VOLNAME   NVOLLAYR  LENGTH     [COL/]OFF    AM/NM       MODE
SC  NAME      PLEX        CACHE     DISKOFFS  LENGTH     [COL/]OFF    DEVICE      MODE
DC  NAME      PARENTVOL   LOGVOL
SP  NAME      SNAPVOL     DCO

v   pubs      -           ENABLED   ACTIVE    22880      SELECT       -           fsgen
pl  pubs-01   pubs        ENABLED   ACTIVE    22880      CONCAT       -           RW
sd  mydg11-01 pubs-01     mydg11    0         22880      0            sdg         ENA

v   voldef    -           ENABLED   ACTIVE    20480      SELECT       -           fsgen
pl  voldef-01 voldef      ENABLED   ACTIVE    20480      CONCAT       -           RW
sd  mydg12-02 voldef-0    mydg12    0         20480      0            sdh         ENA
```

Here `v` is a volume, `pl` is a plex, and `sd` is a subdisk. The first few lines indicate the headers that match each type of output line that follows. Each volume is listed along with its associated plexes and subdisks.

You can ignore the headings for sub-volumes (`SV`), storage caches (`SC`), data change objects (`DCO`) and snappoints (`SP`) in the sample output. No such objects are associated with the volumes that are shown.

To display volume-related information for a specific volume, use the following command:

```
# vxprint [-g diskgroup] -t volume
```

For example, to display information about the volume, `voldef`, in the disk group, `mydg`, use the following command:

```
# vxprint -g mydg -t voldef
```

This example produces the following output:

```
V   NAME      RVG/VSET/CO KSTATE    STATE     LENGTH     READPOL      PREFPLEX    UTYPE

v   voldef    -           ENABLED   ACTIVE    20480      SELECT       -           fsgen
```

If you enable enclosure-based naming, `vxprint` shows enclosure-based names for the disk devices rather than OS-based names.

The output from the `vxprint` command includes information about the volume state.

See "Volume states" on page 350.

Volume states

Table 9-1 shows the volume states that may be displayed by VxVM commands such as `vxprint`.

Table 9-1 Volume states

Volume state	Description
ACTIVE	The volume has been started (the kernel state is currently ENABLED) or was in use (the kernel state was ENABLED) when the machine was rebooted. If the volume is ENABLED, the state of its plexes at any moment is not certain (because the volume is in use). If the volume is DISABLED, the plexes cannot be guaranteed to be consistent, but are made consistent when the volume is started. For a RAID-5 volume, if the volume is DISABLED, parity cannot be guaranteed to be synchronized.
CLEAN	The volume is not started (the kernel state is DISABLED) and its plexes are synchronized. For a RAID-5 volume, its plex stripes are consistent and its parity is good.
EMPTY	The volume contents are not initialized. When the volume is EMPTY, the kernel state is always DISABLED.
INVALID	The contents of an instant snapshot volume no longer represent a true point-in-time image of the original volume.
NEEDSYNC	You must resynchronize the volume the next time it is started. A RAID-5 volume requires a parity resynchronization.
REPLAY	The volume is in a transient state as part of a log replay. A log replay occurs when it becomes necessary to use logged parity and data. This state is only applied to RAID-5 volumes.

Table 9-1 Volume states *(continued)*

Volume state	Description
SYNC	The volume is either in read-writeback recovery mode (the kernel state is ENABLED) or was in read-writeback mode when the machine was rebooted (the kernel state is DISABLED). With read-writeback recovery, plex consistency is recovered by reading data from blocks of one plex and writing the data to all other writable plexes. If the volume is ENABLED, the plexes are being resynchronized through the read-writeback recovery. If the volume is DISABLED, the plexes were being resynchronized through read-writeback when the machine rebooted and still need to be synchronized. For a RAID-5 volume, the volume is either undergoing a parity resynchronization (the kernel state is ENABLED) or was having its parity resynchronized when the machine was rebooted (the kernel state is DISABLED).

The interpretation of these states during volume startup is modified by the persistent state log for the volume (for example, the DIRTY/CLEAN flag). If the clean flag is set, an ACTIVE volume was not written to by any processes or was not even open at the time of the reboot; therefore, it can be considered CLEAN. In any case, the clean flag is always set when the volume is marked CLEAN.

Volume kernel states

The volume kernel state indicates the accessibility of the volume. The volume kernel state lets a volume have an offline (DISABLED), maintenance (DETACHED), or online (ENABLED) mode of operation.

You do not set these states; they are maintained internally. On a system that is operating properly, all volumes are ENABLED.

Table 9-2 shows the volume kernel states that can be defined.

Table 9-2 Volume kernel state

Volume kernel state	Description
DETACHED	Maintenance is being performed on the volume. The volume cannot be read from or written to, but certain plex operations and `ioctl` function calls are accepted.
DISABLED	The volume is offline and cannot be accessed.
ENABLED	The volume is online and can be read from or written to.

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.

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 392.

About SF Thin Reclamation feature

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.
- You can trigger reclamation for a disk, disk group, or enclosure.
- You can trigger reclamation for a VxFS file system.

Reclamation of storage on thin reclamation arrays

Storage Foundation enables reclamation of storage on thin reclamation arrays.

See "How reclamation on a deleted volume works" on page 356.

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 `thinonly` or `thinrclm` attributes on the host.

See "Identifying thin and thin reclamation LUNs " on page 356.

See "Identifying thin and thin reclamation LUNs " on page 356.

See "How reclamation on a deleted volume works" on page 356.

See "Thin Reclamation of a disk, a disk group, or an enclosure" on page 357.

See "Thin Reclamation of a file system" on page 358.

See "Triggering space reclamation" on page 359.

Identifying thin and thin reclamation LUNs

You can only perform Thin Reclamation on LUNS which have the `thinrclm` attribute. VxVM automatically discovers LUNs that support Thin Reclamation from capable storage arrays. To identify devices that are known to have the `thinonly` or `thinrclm` attributes on a host, use the `vxdisk -o thin list` command.

To identify LUNs

- To identify LUNs that are `thin` or `thinrclm` type the following command:

```
# vxdisk -o thin list
DEVICE          SIZE(mb)      PHYS_ALLOC(mb)  GROUP     TYPE
hitachi_usp0_065a 10000         84          -         thinrclm
hitachi_usp0_065b 10000         110         -         thinrclm
hitachi_usp0_065c 10000         74          -         thinrclm
hitachi_usp0_065d 10000         50          -         thinrclm
.
.
.
hitachi_usp0_0660 10000         672         thindg    thinrclm
```

In the above 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 that the array supports thin reclamation.

How reclamation on a deleted volume works

Storage that is no longer in use, needs to be reclaimed by the array. 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, the reclaim operation is made asynchronous. When a volume is deleted the space previously used by the volume is tracked for later asynchronous reclamation. This asynchronous reclamation is handled by `vxrelocd` (or recovery) daemon.

By default, the `vxrelocd` daemon runs everyday at 22:10 hours and reclaims storage on the deleted volume that are one day old.

To perform the reclaim operation during less critical time of the system, control the time of the reclaim operation by using the following tunables:

`reclaim_on_delete_wait_period`	The storage space that is used by the deleted volume is reclaimed after `reclaim_on_delete_wait_period` days. The value of the tunable can be anything between `-1` to `367`. The default is set to `1`, which means the volume is deleted the next day. The storage is reclaimed immediately if the value is `-1`. The storage space is not reclaimed automatically, if the value is greater than `366`. It can only be reclaimed manually using `vxdisk reclaim` command.
`reclaim_on_delete_start_time`	This tunable specifies the time of the day that the reclaim on the deleted volume is performed. The default time is set to `22:10`. This value can be changed to any time of the day.

You can change the tunables using the `vxdefault` command.

Thin Reclamation of a disk, a disk group, or an 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.

Use the `-o full` option of the `vxdisk reclaim` command to also reclaim disk space in unmarked space on the disks.

You can only perform Thin Reclamation on LUNS which have the `thinrclm` attribute.

See "Identifying thin and thin reclamation LUNs " on page 356.

Example of reclamation for disks. The following example triggers reclamation on LUNs `disk1` and `disk2`:

```
# vxdisk reclaim disk1 disk2
```

In the above example, suppose the `disk1` contains a VxVM volume `vol1` with a VxFS file system. If the VxFS file system is not mounted, the command skips reclamation for `disk1`.

To reclaim space on `disk1`, use the following command:

```
# vxdisk -o full reclaim disk1
```

The above command reclaims unused space on `disk1` that is outside of the `vol1`. The reclamation skips the `vol1` volume, since the VxFS file system is not mounted, but it scans the rest of the disk for unused space.

Example of reclamation for disk groups. The following example triggers reclamation on the disk group `oradg`:

```
# vxdisk reclaim oradg
```

Example of reclamation for an enclosure. The following example triggers reclamation on the enclosure=EMC_CLARiiON0:

```
# vxdisk reclaim  EMC_CLARiiON0
```

You can also trigger Thin Reclamation on a VxFS file system.

Thin Reclamation takes considerable amount of time when you reclaim thin storage on a large number of LUNs or an enclosure or disk group.

See "Monitoring Thin Reclamation using the vxtask command" on page 360.

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 or the `vxfs_ts_reclaim` API. 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, Symantec 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) and `vxfs_ts_reclaim`(3) manual pages.

Thin Reclamation is only supported on file systems mounted on a VxVM volume.

The following example performs aggressive reclamation of free storage to the Thin Storage LUN on a VxFS file system mounted at `/mnt1`:

```
# /opt/VRTS/bin/fsadm -R /mnt1
```

Veritas File System also supports reclamation of a portion of the file system using the `vxfs_ts_reclaim`() API.

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.

Triggering space reclamation

This section describes how to trigger space reclamation.

To trigger space reclamation

1. Ensure you mounted the VxFS file system.

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

   If you need to mount the VxFS file system, see the `mount_vxfs`(1M) manual page.

2. Use the `fsadm` command to trigger space reclamation:

   ```
   # /opt/VRTS/bin/fsadm -t vxfs -R /<VxFS_mount_point>
   ```

   where `<VxFS_mount_point>` is the name of the VxFS file system mount point.

Note: If the VxFS file system is not mounted you will receive an error message. For example: Disk 3pardata0_110 : Skipped. No VxFS file system found.

Monitoring Thin Reclamation using the vxtask command

This section describes how to monitor thin reclamation using the `vxtask` command.

To monitor thin reclamation

1 To initiate thin reclamation, use the following command:

```
# vxdisk reclaim diskgroup
```

For example:

```
# vxdisk reclaim dg100
```

2 To monitor the reclamation status, run the following command in another session:

```
# vxtask list

TASKID PTID TYPE/STATE PCT   PROGRESS
171    RECLAIM/R     00.00% 0/41875931136/0 RECLAIM vol100 dg100
```

The `vxdisk reclaim diskgroup` command runs in another session while you run the `vxtask list` command.

Using SmartMove with Thin Provisioning

This section describes how to use SmartMove with Thin Provisioning that improves the synchronization performance and uses thin storage efficiently.

To use SmartMove with Thin Provisioning

1 Mount the volume as the VxFS file system type. For example:

```
# mount -t vxfs /dev/vx/dsk/oradg/oravol1 /oravol1
```

2 Run the following command:

```
# sync
```

3 Mirror the volume. For example:

```
# vxassist -g oradg mirror oravol1
```

See the *Veritas Storage Foundation Advanced Features Administrator's Guide* for more information on Thin Provisioning and SmartMove.

Admin operations on an unmounted VxFS thin volume

A thin volume is a volume composed of one or more thin LUNs. If a thin volume is not mounted on a VxFS file system, any resynchronization, synchronization, or refresh operation on the volume, plex, or subdisk performs a full synchronization and allocates storage on the unused space of the volume. Only a mounted VxFS file system can use SmartMove to assist with optimized administrative operations on thin volumes.

By default, commands that mirror, take snapshots, or attach a plex may fail with an error message.

Some commands use the `-f` option and others use the `-o force` option to force the command. The command manual page contains details of the force option to be used.

Note: The full new plex or volume allocates physical storage on thin LUNs and will not be a thin/optimized operation.

Stopping a volume

Stopping a volume renders it unavailable to the user, and changes the volume kernel state from ENABLED or DETACHED to DISABLED. If the volume cannot be disabled, it remains in its current state. To stop a volume, use the following command:

```
# vxvol [-g diskgroup] [-f] stop volume ...
```

To stop all volumes in a specified disk group, use the following command:

```
# vxvol [-g diskgroup] [-f] stopall
```

Warning: If you use the `-f` option to forcibly disable a volume that is currently open to an application, the volume remains open, but its contents are inaccessible. I/O operations on the volume fail, and this may cause data loss. You cannot deport a disk group until all its volumes are closed.

If you need to prevent a closed volume from being opened, use the `vxvol maint` command, as described in the following section.

Putting a volume in maintenance mode

If all mirrors of a volume become STALE, you can place the volume in maintenance mode. Before you put the volume in maintenance mode, make sure the volume is stopped or it is in the DISABLED state. Then you can view the plexes while the volume is DETACHED and determine which plex to use for reviving the others. To place a volume in maintenance mode, use the following command:

```
# vxvol [-g diskgroup] maint volume
```

To assist in choosing the revival source plex, use vxprint to list the stopped volume and its plexes.

To take a plex offline, (in this example, vol01-02 in the disk group, mydg), use the following command:

```
# vxmend -g mydg off vol01-02
```

Make sure that all the plexes are offline except for the one that you will use for revival. The plex from which you will revive the volume should be placed in the STALE state. The vxmend on command can change the state of an OFFLINE plex of a DISABLED volume to STALE. For example, to put the plex vol101-02 in the STALE state, use the following command:

```
# vxmend -g mydg on vol101-02
```

Running the vxvol start command on the volume then revives the volume with the specified plex. Because you are starting the volume from a stale plex, you must specify the force option (-f).

By using the procedure above, you can enable the volume with each plex, and you can decide which plex to use to revive the volume.

After you specify a plex for revival, and you use the procedure above to enable the volume with the specified plex, put the volume back into the DISABLED state and put all the other plexes into the STALE state using the vxmend on command. Now, you can recover the volume.

See "Starting a volume" on page 362.

Starting a volume

Starting a volume makes it available for use, and changes the volume state from DISABLED or DETACHED to ENABLED. To start a DISABLED or DETACHED volume, use the following command:

```
# vxvol [-g diskgroup] start volume ...
```

If a volume cannot be enabled, it remains in its current state.

To start all DISABLED or DETACHED volumes in a disk group, enter the following:

```
# vxvol -g diskgroup startall
```

To start a DISABLED volume, enter the following:

```
# vxrecover -g diskgroup -s volume ...
```

To start all DISABLED volumes, enter the following:

```
# vxrecover -s
```

To prevent any recovery operations from being performed on the volumes, additionally specify the `-n` option to `vxrecover`.

Resizing a volume

Resizing a volume changes its size. For example, if a volume is too small for the amount of data it needs to store, you can increase its length . To resize a volume, use one of the following commands: `vxresize` (preferred), `vxassist`, or `vxvol`.

Note: You cannot use VxVM commands, Storage Foundation Manager (SFM), or VEA to resize a volume or any underlying file system on an encapsulated `root` disk. This is because the underlying disk partitions also need to be reconfigured. If you need to resize the volumes on the root disk, you must first unencapsulate the root disk.

If you increase a volume's size, the `vxassist` command automatically locates available disk space. The `vxresize` command lets you optionally specify the LUNs or disks to use to increase the size of a volume. The `vxvol` command requires that you have previously ensured that there is sufficient space available in the plexes of the volume to increase its size. The `vxassist` and `vxresize` commands free unused space for use by the disk group. For the `vxvol` command, you must do this yourself. To determine how much you can increase a volume, use the following command:

```
# vxassist [-g diskgroup] maxgrow volume
```

When you resize a volume, you can specify the length of a new volume in sectors, kilobytes, megabytes, or gigabytes. The unit of measure is added as a suffix to the length (`s`, `m`, `k`, or `g`). If you do not specify a unit, sectors are assumed. The `vxassist`

command also lets you specify an increment by which to change the volume's size.

Warning: If you use `vxassist` or `vxvol` to resize a volume, do not shrink it below the size of the file system on it. If you do not shrink the file system first, you risk unrecoverable data loss. If you have a `VxFS` file system, shrink the file system first, and then shrink the volume. For other file systems, you may need to back up your data so that you can later recreate the file system and restore its data.

Resizing volumes with vxresize

Use the `vxresize` command to resize a volume containing a file system. Although you can use other commands to resize volumes containing file systems, `vxresize` offers the advantage of automatically resizing certain types of file system as well as the volume.

Table 9-3 shows which operations are permitted and whether you need to unmount the file system before you resize it.

Table 9-3 Permitted resizing operations on file systems

	Full-VxFS	ext2, ext3, reiserfs
Mounted file system	Grow and shrink	Not allowed
Unmounted file system	Not allowed	Grow and shrink

For example, the following command resizes a volume from 1 GB to 10 GB. The volume is `homevol` in the disk group `mydg`, and contains a VxFS file system. The command uses spare disks `mydg10` and `mydg11`.

```
# vxresize -g mydg -b -F vxfs -t homevolresize homevol 10g mydg10 mydg11
```

The `-b` option specifies that this operation runs in the background. To monitor its progress, specify the task tag `homevolresize` with the `vxtask` command.

When you use `vxresize`, note the following restrictions:

- vxresize works with VxFS and `ext2`, `ext3`, and `reiserfs` file systems only. It does not work with other file system types, or if no file system is configured on a volume.
- In some situations, when you resize large volumes, `vxresize` may take a long time to complete.

- If you resize a volume with a usage type other than FSGEN or RAID5, you can lose data. If such an operation is required, use the `-f` option to forcibly resize the volume.
- You cannot resize a volume that contains plexes with different layout types. Attempting to do so results in the following error message:

  ```
  VxVM vxresize ERROR V-5-1-2536 Volume volume has different
  organization in each mirror
  ```

  To resize such a volume successfully, you must first reconfigure it so that each data plex has the same layout.

Note: If you enter an incorrect volume size, do not try to stop the `vxresize` operation by entering Crtl-C. Let the operation complete and then rerun `vxresize` with the correct value.

For more information about the `vxresize` command, see the `vxresize`(1M) manual page.

Resizing volumes with vxassist

The following modifiers are used with the `vxassist` command to resize a volume:

`growto`	Increases the volume size to a specified length.
`growby`	Increases the volume size by a specified amount.
`shrinkto`	Reduces the volume size to a specified length.
`shrinkby`	Reduces the volume size by a specified amount.

Extending to a given length

To extend a volume to a specific length, use the following command:

```
# vxassist [-b] [-g diskgroup] growto volume length
```

If you specify the `-b` option, growing the volume is a background task.

For example, to extend `volcat` to 2000 sectors, use the following command:

```
# vxassist -g mydg growto volcat 2000
```

If you want the subdisks to be grown using contiguous disk space, and you previously performed a relayout on the volume, also specify the attribute `layout=nodiskalign` to the `growto` command.

Extending by a given length

To extend a volume by a specific length, use the following command:

```
# vxassist [-b] [-g diskgroup] growby volume length
```

If you specify `-b` option, growing the volume is a background task.

For example, to extend `volcat` by 100 sectors, use the following command:

```
# vxassist -g mydg growby volcat 100
```

If you want the subdisks to be grown using contiguous disk space, and you previously performed a relayout on the volume, also specify the attribute `layout=nodiskalign` to the `growby` command .

Shrinking to a given length

To shrink a volume to a specific length, use the following command:

```
# vxassist [-g diskgroup] shrinkto volume length
```

For example, to shrink `volcat` to 1300 sectors, use the following command:

```
# vxassist -g mydg shrinkto volcat 1300
```

Warning: Do not shrink the volume below the current size of the file system or database using the volume. You can safely use the `vxassist shrinkto` command on empty volumes.

Shrinking by a given length

To shrink a volume by a specific length, use the following command:

```
# vxassist [-g diskgroup] shrinkby volume length
```

For example, to shrink `volcat` by 300 sectors, use the following command:

```
# vxassist -g mydg shrinkby volcat 300
```

Warning: Do not shrink the volume below the current size of the file system or database using the volume. You can safely use the `vxassist shrinkby` command on empty volumes.

Resizing volumes with vxvol

To change the length of a volume , use the following command:

```
# vxvol [-g diskgroup] set len=length volume
```

For example, to change the length of the volume `vol01`, in the disk group `mydg`, to 100000 sectors, use the following command:

```
# vxvol -g mydg set len=100000 vol01
```

Note: You cannot use the `vxvol set len` command to increase the size of a volume unless the needed space is available in the volume's plexes. When you reduce the volume's size using the `vxvol set len` command, the freed space is not released into the disk group's free space pool.

If a volume is active and you reduce its length, you must force the operation using the `-o force` option to `vxvol`. This precaution ensures that space is not removed accidentally from applications using the volume.

You can change the length of logs using the following command:

```
# vxvol [-g diskgroup] set loglen=length log_volume
```

Warning: Sparse log plexes are not valid. They must map the entire length of the log. If increasing the log length makes any of the logs invalid, the operation is not allowed. Also, if the volume is not active and is dirty (for example, if it has not been shut down cleanly), you cannot change the log length. If you are decreasing the log length, this feature avoids losing any of the log contents. If you are increasing the log length, it avoids introducing random data into the logs.

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 VM 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 mydg01 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)
   ```

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 317.

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
```

Adding logs and maps to volumes

Veritas Volume Manager supports the following types of volume logs and maps:

- FastResync Maps improve performance and reduce I/O during mirror resynchronization. These maps can be either in memory (Non-Persistent) or on disk (Persistent) as part of a DCO volume.
  See "FastResync" on page 61.
  See "Enabling FastResync on a volume" on page 385.

  Two types of DCO volumes are supported:

  - Version 0 DCO volumes only support Persistent FastResync for the traditional third-mirror break-off type of volume snapshot.
    See "Version 0 DCO volume layout" on page 64.
  - Version 20 DCO volumes, introduced in VxVM 4.0, combine DRL logging (see below) and Persistent FastResync for full-sized and space-optimized instant volume snapshots.

See "Version 20 DCO volume layout" on page 64.
See "Preparing a volume for DRL and instant snapshots" on page 371.

- Dirty Region Logs let you quickly recover mirrored volumes after a system crash. These logs can be either DRL log plexes, or part of a version 20 DCO volume.
  See "Dirty region logging" on page 56.
  See "Adding traditional DRL logging to a mirrored volume" on page 377.
  See "Preparing a volume for DRL and instant snapshots" on page 371.
- RAID-5 logs prevent corruption of data during recovery of RAID-5 volumes. These logs are configured as plexes on disks other than those that are used for the columns of the RAID-5 volume.
  See "RAID-5 logging" on page 48.
  See "Adding a RAID-5 log" on page 394.

Preparing a volume for DRL and instant snapshots

You can add a version 20 data change object (DCO) and DCO volume to an existing volume if the disk group version number is 110 or greater. You can also simultaneously create a new volume, a DCO and DCO volume, and enable DRL as long as the disk group version is 110 or greater.

See "Determining the DCO version number" on page 374.

See "Creating a volume with a version 20 DCO volume" on page 328.

See "Upgrading existing volumes to use version 20 DCOs" on page 378.

Note: You need a license key to use the DRL and FastResync feature. If you do not have a license key, you can configure a DCO object and DCO volume so that snap objects are associated with the original and snapshot volumes. However, without a license key, only full resynchronization can be performed.

See "How persistent FastResync works with snapshots" on page 65.

To add a version 20 DCO and DCO volume to a volume, use the following command :

```
# vxsnap [-g diskgroup] prepare volume [ndcomirs=number] \
  [regionsize=size] [drl=on|sequential|off] \
  [storage_attribute ...]
```

The `ndcomirs` attribute specifies the number of DCO plexes that are created in the DCO volume. You should configure as many DCO plexes as there are data and

snapshot plexes in the volume. The DCO plexes are used to set up a DCO volume for any snapshot volume that you subsequently create from the snapshot plexes. For example, specify `ndcomirs=5` for a volume with 3 data plexes and 2 snapshot plexes.

The value of the `regionsize` attribute specifies the size of the tracked regions in the volume. A write to a region is tracked by setting a bit in the change map. The default value is `64k` (64KB). A smaller value requires more disk space for the change maps, but the finer granularity provides faster resynchronization.

To enable DRL logging on the volume, specify `drl=on` (this is the default). For sequential DRL, specify `drl=sequential`. If you do not need DRL, specify `drl=off`.

You can also 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 20 DCO plexes" on page 372.

The `vxsnap prepare` command automatically enables Persistent FastResync on the volume. Persistent FastResync is also set automatically on any snapshots that are generated from a volume on which this feature is enabled.

If the volume is a RAID-5 volume, it is converted to a layered volume that can be used with instant snapshots and Persistent FastResync.

See "Using a DCO and DCO volume with a RAID-5 volume" on page 373.

By default, a version 20 DCO volume contains 32 per-volume maps. If you require more maps, you can use the `vxsnap addmap` command to add them.

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

Specifying storage for version 20 DCO plexes

If you move the disks that contain volumes and their snapshots into different disk groups, you must ensure that the disks that contain their DCO plexes can accompany them. You can use storage attributes to specify which disks to use for the DCO plexes. (If you do not want to use dirty region logging (DRL) with a volume, you can specify the same disks as those on which the volume is configured, assuming that space is available on the disks). For example, to add a DCO object and mirrored DCO volume with plexes on `disk05` and `disk06` to the volume, `myvol`, use the following command:

```
# vxsnap -g mydg prepare myvol ndcomirs=2 alloc=disk05,disk06
```

To view the details of the DCO object and DCO volume that are associated with a volume, use the `vxprint` command. The following is example `vxprint -vh` 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    foo-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 you need to relocate DCO plexes to different disks, you can use the `vxassist move` command. 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 267.

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

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

Using a DCO and DCO volume with a RAID-5 volume

You can add a DCO and DCO volume to a RAID-5 volume. This lets you use Persistent FastResync on the volume for fast resynchronization of snapshots on returning them to their original volume. However, this procedure has the side effect of converting the RAID-5 volume into a special type of layered volume. You can create space-optimized instant snapshots of such a volume, and you can add mirrors that may be broken off as full-sized instant snapshots. You cannot relayout or resize such a volume unless you convert it back to a pure RAID-5 volume.

To convert a volume back to a RAID-5 volume, remove any snapshot plexes from the volume, and dissociate the DCO and DCO volume from the layered volume. You can then perform relayout and resize operations on the resulting non-layered RAID-5 volume.

See "Removing support for DRL and instant snapshots from a volume" on page 376.

To allow Persistent FastResync to be used with the RAID-5 volume again, re-associate the DCO and DCO volume.

See "Preparing a volume for DRL and instant snapshots" on page 371.

Warning: Dissociating a DCO and DCO volume disables FastResync on the volume. A full resynchronization of any remaining snapshots is required when they are snapped back.

Determining the DCO version number

To use the instant snapshot and DRL-enabled DCO features, you must use a version 20 DCO, rather than version 0 DCO.

To find out the version number of a DCO that is associated with a volume

1. Use the `vxprint` command on the volume to discover the name of its DCO. Enter the following:

```
# DCONAME=`vxprint [-g diskgroup] -F%dco_name volume`
```

2. Use the `vxprint` command on the DCO to determine its version number. Enter the following:

```
# vxprint [-g diskgroup] -F%version $DCONAME
```

Determining if DRL is enabled on a volume

To determine if DRL (configured using a version 20 DCO) is enabled on a volume

1. Use the `vxprint` command on the volume to discover the name of its DCO. Enter the following:

   ```
   # DCONAME=`vxprint [-g diskgroup] -F%dco_name volume`
   ```

2. To determine if DRL is enabled on the volume, enter the following command with the volume's DCO:

   ```
   # vxprint [-g diskgroup] -F%drl $DCONAME
   ```

   If this command displays `on`, DRL is enabled.

3. If DRL is enabled, enter the following command with the DCO to determine if sequential DRL is enabled:

   ```
   # vxprint [-g diskgroup] -F%sequentialdrl $DCONAME
   ```

   If this command displays `on`, sequential DRL is enabled.

   You can also use the following command with the volume:

   ```
   # vxprint [-g diskgroup] -F%log_type volume
   ```

   This displays the logging type as `REGION` for `DRL`, `DRLSEQ` for sequential DRL, or `NONE` if DRL is not enabled.

   If the number of active mirrors in the volume is less than 2, DRL logging is not performed even if DRL is enabled on the volume.

   See "Determining if DRL logging is active on a volume" on page 376.

Determining if DRL logging is active on a volume

To determine if DRL logging (configured using a version 20 DCO) is active on a mirrored volume

1. Use the following `vxprint` commands to discover the name of the volume's DCO volume:

```
# DCONAME=`vxprint [-g diskgroup] -F%dco_name volume`
# DCOVOL=`vxprint [-g diskgroup] -F%parent_vol $DCONAME`
```

2. Use the `vxprint` command on the DCO volume to find out if DRL logging is active:

```
# vxprint [-g diskgroup] -F%drllogging $DCOVOL
```

   This command returns `on` if DRL logging is enabled.

Disabling and re-enabling DRL

To disable DRL (configured using a version 20 DCO) on a volume, enter the following:

```
# vxvol [-g diskgroup] set drl=off volume
```

To re-enable DRL on a volume, enter the following:

```
# vxvol [-g diskgroup] set drl=on volume
```

To re-enable sequential DRL on a volume, enter the following:

```
# vxvol [-g diskgroup] set drl=sequential volume
```

You can use these commands to change the DRL policy on a volume by first disabling and then re-enabling DRL as required. If a data change map (DCM, used with Veritas Volume Replicator) is attached to a volume, DRL is automatically disabled .

Removing support for DRL and instant snapshots from a volume

To remove support for DRL and instant snapshot operation from a volume, use the following command to remove the DCO and DCO volume that are associated with the volume:

```
#  vxsnap [-g diskgroup] unprepare volume
```

This command also has the effect of disabling FastResync tracking on the volume.

Note: If the volume is part of a snapshot hierarchy, this command fails .

Adding traditional DRL logging to a mirrored volume

A traditional DRL log is configured within a DRL plex. A version 20 DCO volume cannot be used in conjunction with a DRL plex. The version 20 DCO volume layout includes space for a DRL log.

See "Preparing a volume for DRL and instant snapshots" on page 371.

To put dirty region logging (DRL) into effect for a mirrored volume, you must add a log subdisk to that volume. Only one log subdisk can exist per plex.

To add DRL logs to an existing volume, use the following command:

```
# vxassist [-b] [-g diskgroup] addlog volume logtype=drl \
   [nlog=n] [loglen=size]
```

If specified, the `-b` option makes adding the new logs a background task.

The `nlog` attribute specifies the number of log plexes to add. By default, one log plex is added. The `loglen` attribute specifies the size of the log, where each bit represents one region in the volume. For example, a 10 GB volume with a 64 KB region size needs a 20K log.

For example, to add a single log plex for the volume `vol03` in the disk group `mydg`, use the following command:

```
# vxassist -g mydg addlog vol03 logtype=drl
```

When you use the `vxassist` command to add a log subdisk to a volume, a log plex is created by default to contain the log subdisk. If you do not want a log plex, include the keyword `nolog` in the layout specification.

For a volume that will be written to sequentially, such as a database log volume, use the following `logtype=drlseq` attribute to specify that sequential DRL will be used:

```
# vxassist -g mydg addlog volume logtype=drlseq [nlog=n]
```

After you create the plex containing a log subdisk, you can treat it as a regular plex. You can add subdisks to the log plex. If you need to, you can remove the log plex and log subdisk.

See "Removing a traditional DRL log" on page 378.

Removing a traditional DRL log

You can use the `vxassist remove log` command to remove a traditional DRL log that is configured within a DRL plex. The command will not remove a DRL log that is configured within a version 20 DCO.

To remove a traditional DRL log

- Type the following command:

  ```
  # vxassist [-g diskgroup] remove log volume logtype=drl [nlog=n]
  ```

  By default, the `vxassist` command removes one log. Use the optional attribute `nlog=`*n* to specify the number of logs that are to remain after the operation completes.

  You can use storage attributes to specify the storage from which a log will be removed. For example, to remove a log on disk `mydg10` from volume vol01, enter the following command.

  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 log vol01 \!mydg10 logtype=drl
  ```

Upgrading existing volumes to use version 20 DCOs

You can upgrade a volume created before VxVM 4.0 to take advantage of new features such as instant snapshots and DRL logs that are configured within the DCO volume. You must upgrade the version of the disk groups, remove snapshots and version 0 DCOs that are associated with volumes in the disk groups, and configure the volumes with version 20 DCOs.

The plexes of the DCO volume require persistent storage space on disk to be available. To make room for the DCO plexes, you may need to add extra disks to the disk group, or reconfigure volumes to free up space in the disk group. You can also add disk space by using the disk group move feature to bring in spare disks from a different disk group.

See "Reorganizing the contents of disk groups" on page 262.

The `vxsnap prepare` command automatically enables FastResync on the volume and on any snapshots that are generated from it.

If the volume is a RAID-5 volume, it is converted to a layered volume that can be used with snapshots and FastResync.

To upgrade an existing disk group and the volumes that it contains

1 Upgrade the disk group that contains the volume to the latest version before performing the remainder of the procedure described in this section. To check the version of a disk group, use the following command :

```
# vxdg list diskgroup
```

To upgrade a disk group to the latest version, use the following command:

```
# vxdg upgrade diskgroup
```

See "Upgrading the disk group version" on page 276.

2 To discover which volumes in the disk group have version 0 DCOs associated with them, use the following command:

```
# vxprint [-g diskgroup] -F "%name" -e "v_hasdcolog"
```

This command assumes that the volumes can only have version 0 DCOs as the disk group has just been upgraded.

See "Determining the DCO version number" on page 374.

To upgrade each volume within the disk group, repeat the following steps as required.

3 If the volume to be upgraded has a traditional DRL plex or subdisk (that is, the DRL logs are not held in a version 20 DCO volume), use the following command to remove this:

```
# vxassist [-g diskgroup] remove log volume [nlog=n]
```

To specify the number, *n*, of logs to be removed, use the optional attribute *nlog=n* . By default, the `vxassist` command removes one log.

4 For a volume that has one or more associated snapshot volumes, use the following command to reattach and resynchronize each snapshot:

```
# vxassist [-g diskgroup] snapback snapvol
```

If FastResync was enabled on the volume before the snapshot was taken, the data in the snapshot plexes is quickly resynchronized from the original volume. If FastResync was not enabled, a full resynchronization is performed.

5 To turn off FastResync for the volume, use the following command :

```
# vxvol [-g diskgroup] set fastresync=off volume
```

6 To dissociate a version 0 DCO object, DCO volume and snap objects from the volume, use the following command:

```
# vxassist [-g diskgroup] remove log volume logtype=dco
```

7 To upgrade the volume, use the following command:

```
# vxsnap [-g diskgroup] prepare volume [ndcomirs=number] \
  [regionsize=size] [drl=on|sequential|off] \
  [storage_attribute ...]
```

The `ndcomirs` attribute specifies the number of DCO plexes that are created in the DCO volume. You should configure as many DCO plexes as there are data and snapshot plexes in the volume. The DCO plexes are used to set up a DCO volume for any snapshot volume that you subsequently create from the snapshot plexes. For example, specify `ndcomirs=5` for a volume with 3 data plexes and 2 snapshot plexes.

The `regionsize` attribute specifies the size of the tracked regions in the volume. A write to a region is tracked by setting a bit in the change map. The default value is `64k` (64KB). A smaller value requires more disk space for the change maps, but the finer granularity provides faster resynchronization.

To enable DRL logging on the volume, specify `drl=on` (this is the default setting). If you need sequential DRL, specify `drl=sequential`. If you do not need DRL, specify `drl=off`.

To define the disks that can or cannot be used for the plexes of the DCO volume, you can also specify `vxassist`-style storage attributes.

Setting tags on volumes

Volume tags implement Storage Foundation's SmartTier feature. You can also apply tags to vsets using the same `vxvm` command syntax as shown below.

See the *Veritas Storage Foundation Advanced Features Administrator's Guide*.

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.

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`.

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`.

Changing the read policy for mirrored volumes

VxVM offers the choice of the following read policies on the data plexes in a mirrored volume:

`round`	Reads each plex in turn in "round-robin" fashion for each nonsequential I/O detected. Sequential access causes only one plex to be accessed. This approach takes advantage of the drive or controller read-ahead caching policies.
`prefer`	Reads first from a plex that has been named as the preferred plex.
`select`	Chooses a default policy based on plex associations to the volume. If the volume has an enabled striped plex, the `select` option defaults to preferring that plex; otherwise, it defaults to round-robin. For disk group versions 150 or higher and if there is a SSD based plex available, it will be preferred over other plexes.
`siteread`	Reads preferentially from plexes at the locally defined site. This method is the default policy for volumes in disk groups where site consistency has been enabled. For disk group versions 150 or higher and if the local site has a SSD based plex, it will be preferred. See "About site consistency" on page 481.
`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
```

See "Volume read policies" on page 499.

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.

Moving volumes from a VM 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)
```

Enabling FastResync on a volume

The recommended method for enabling FastResync on a volume with a version 20 DCO is to use the `vxsnap prepare` command.

See "Preparing a volume for DRL and instant snapshots" on page 371.

Note: To use this feature, you need a FastResync license.

FastResync quickly and efficiently resynchronizes stale mirrors. When you use FastResync with operations such as backup and decision support, it also increases the efficiency of the VxVM snapshot mechanism.

See "FastResync" on page 61.

You can enable the following versions of FastResync on a volume:

- Persistent FastResync holds copies of the FastResync maps on disk. If a system is rebooted, you can use these copies to quickly recover mirrored volumes. To use this form of FastResync, you must first associate a version 0 or a version 20 data change object (DCO) and DCO volume with the volume.
  See "Upgrading existing volumes to use version 20 DCOs" on page 378.
  See "Preparing a volume for DRL and instant snapshots" on page 371.
- Non-Persistent FastResync holds the FastResync maps in memory. These maps do not survive on a system that is rebooted.

By default, FastResync is not enabled on newly-created volumes. If you want to enable FastResync on a volume that you create, specify the `fastresync=on` attribute to the `vxassist make` command.

Note: You cannot configure Persistent and Non-Persistent FastResync on a volume. If a DCO is associated with the volume, Persistent FastResync is used. Otherwise, Non-Persistent FastResync is used.

To turn on FastResync for an existing volume, specify `fastresync=on` to the `vxvol` command as follows:

```
# vxvol [-g diskgroup] set fastresync=on volume
```

To use FastResync with a snapshot, you must enable FastResync before the snapshot is taken, and it must remain enabled until after the snapback is completed.

Checking whether FastResync is enabled on a volume

To check whether FastResync is enabled on a volume, use the following command:

```
# vxprint [-g diskgroup] -F%fastresync volume
```

If FastResync is enabled, the command returns `on`; otherwise, it returns `off`.

If FastResync is enabled, to check whether it is Non-Persistent or Persistent FastResync, use the following command:

```
# vxprint [-g diskgroup] -F%hasdcolog volume
```

If Persistent FastResync is enabled, the command returns `on`; otherwise, it returns `off`.

To list all volumes on which Non-Persistent FastResync is enabled, use the following command.

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"
```

To list all volumes on which Persistent FastResync is enabled, use the following command:

```
# vxprint [-g diskgroup] -F "%name" -e "v_fastresync=on \
  && v_hasdcolog"
```

Disabling FastResync

Use the `vxvol` command to turn off Persistent or Non-Persistent FastResync for an existing volume, as follows:

```
# vxvol [-g diskgroup] set fastresync=off volume
```

Turning off FastResync releases all tracking maps for the specified volume. All subsequent reattaches do not use the FastResync facility, but perform a full resynchronization of the volume. The full resynchronization occurs even if you turn on FastResync later.

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.

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 387.

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 391.

Permitted relayout transformations

Table 9-4 shows the supported relayout transformations for concatenated volumes.

Table 9-4 Supported relayout transformations for concatenated volumes

Relayout to	From concat
concat	No.
concat-mirror	No. Add a mirror, and then use `vxassist convert` instead.

Table 9-4 Supported relayout transformations for concatenated volumes *(continued)*

Relayout to	From concat
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.
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 9-5 shows the supported relayout transformations for concatenated-mirror volumes.

Table 9-5 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 9-6 shows the supported relayout transformations for RAID-5 volumes.

Table 9-6 Supported relayout transformations for mirrored-stripe volumes

Relayout to	From mirror-stripe
concat	Yes.
concat-mirror	Yes.

Table 9-6 Supported relayout transformations for mirrored-stripe volumes *(continued)*

Relayout to	From mirror-stripe
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. Otherwise, use `vxassist convert`.

Table 9-7 shows the supported relayout transformations for mirror-concatenated volumes.

Table 9-7 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 9-8 shows the supported relayout transformations for mirrored-stripe volumes.

Table 9-8 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.
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 9-9 shows the supported relayout transformations for unmirrored stripe and layered striped-mirror volumes.

Table 9-9 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 392.

See "Controlling the progress of a relayout" on page 392.

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 353.

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

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

Converting between layered and non-layered volumes

The `vxassist convert` command transforms volume layouts between layered and non-layered forms. The command has the following syntax

```
# vxassist [-b] [-g diskgroup] convert volume [layout=layout] \
  [convert_options]
```

If you specify the `-b` option, the conversion of the volume is a background task.

The following conversion layouts are supported:

`stripe-mirror`	Mirrored-stripe to striped-mirror
`mirror-stripe`	Striped-mirror to mirrored-stripe
`concat-mirror`	Mirrored-concatenated to concatenated-mirror
`mirror-concat`	Concatenated-mirror to mirrored-concatenated

You can use volume conversion before or after you perform an online relayout to achieve more transformations than would otherwise be possible. During relayout process, a volume may also be converted into an intermediate layout. For example, to convert a volume from a 4-column mirrored-stripe to a 5-column mirrored-stripe, first use `vxassist relayout` to convert the volume to a 5-column striped-mirror as follows:

```
# vxassist -g mydg relayout vol1 ncol=5
```

When the relayout finishes, use the `vxassist convert` command to change the resulting layered striped-mirror volume to a non-layered mirrored-stripe:

```
# vxassist -g mydg convert vol1 layout=mirror-stripe
```

Note: If the system crashes during relayout or conversion, the process continues when the system is rebooted. However, if the system crashes during the first stage of a two-stage relayout and conversion, only the first stage finishes. To complete the operation, you must run `vxassist convert` manually.

Adding a RAID-5 log

You can only have one RAID-5 plex per RAID-5 volume. Additional plexes become RAID-5 log plexes, which log information about data and parity being written to the volume. When you create a RAID-5 volume using the `vxassist` command, a log plex is created for that volume by default.

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
```

Adding a RAID-5 log using vxplex

You can also add a RAID-5 log using the `vxplex` command. For example, to attach the RAID-5 log plex `r5log`, to the RAID-5 volume `r5vol`, in the disk group `mydg`, use the following command:

```
# vxplex -g mydg att r5vol r5log
```

The attach operation can only proceed if the size of the new log is large enough to hold all the data on the stripe. If the RAID-5 volume already contains logs, the new log length is the minimum of each individual log length. The reason is that the new log is a mirror of the old logs.

If the RAID-5 volume is not enabled, the new log is marked as BADLOG and is enabled when the volume is started. However, the contents of the log are ignored.

If the RAID-5 volume is enabled and has other enabled RAID-5 logs, the new log's contents are synchronized with the other logs.

If the RAID-5 volume currently has no enabled logs, the new log is zeroed before it is enabled.

Removing a RAID-5 log

To identify the plex of the RAID-5 log, use the following command:

```
# vxprint [-g diskgroup] -ht volume
```

where *volume* is the name of the RAID-5 volume. For a RAID-5 log, the output lists a plex with a `STATE` field entry of `LOG`.

To dissociate and remove a RAID-5 log and any associated subdisks from an existing volume, use the following command:

```
# vxplex [-g diskgroup] -o rm dis plex
```

For example, to dissociate and remove the log plex `volraid-02` from `volraid` in the disk group `mydg`, use the following command:

```
# vxplex -g mydg -o rm dis volraid-02
```

You can also remove a RAID-5 log with the `vxassist` command, as follows:

```
# vxassist [-g diskgroup] remove log volume [nlog=n]
```

By default, the `vxassist` command removes one log. To specify the number of logs that remain after the operation, use the optional attribute `nlog=`*n*.

Note: When you remove a log and it leaves less than two valid logs on the volume, a warning is printed and the operation is stopped. You can force the operation by specifying the `-f` option with `vxplex` or `vxassist`.

Chapter 10

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.

For more information on SmartTier, see the *Veritas Storage Foundation Advanced Features Administrator's Guide*.

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 "Raw device node access to component volumes" on page 401.
- 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 Storage Foundation Advanced Features Administrator's 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 rmvol myvset vol1
# vxvset -g mydg 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        -
```

Raw device node access to 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 Symantec 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.

See "Enabling raw device access when creating a volume set" on page 402.

See "Displaying the raw device access settings for a volume set" on page 403.

See "Controlling raw device access for an existing volume set" on page 403.

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 403.

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 11

Configuring off-host processing

This chapter includes the following topics:

About off-host processing solutions

Off-host processing lets you implement the following activities:

Data backup	As the requirement for 24 x 7 availability becomes essential for many businesses, organizations cannot afford the downtime involved in backing up critical data offline. By taking a snapshot of the data, and backing up from this snapshot, business-critical applications can continue to run without extended down time or impacted performance.
Decision support analysis and reporting	Because snapshots hold a point-in-time copy of a production database, you can set up a replica of the database using the snapshots. Operations such as decision support analysis and business reporting do not require access to up-to-the-minute information. They can use a database copy that is running on a host other than the primary. When required, the database copy can quickly be synchronized 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 gives developers, system testers and QA groups 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. By restoring the database table files from a snapshot copy, the database can be recovered more quickly than by full restoration from tape or other backup media.

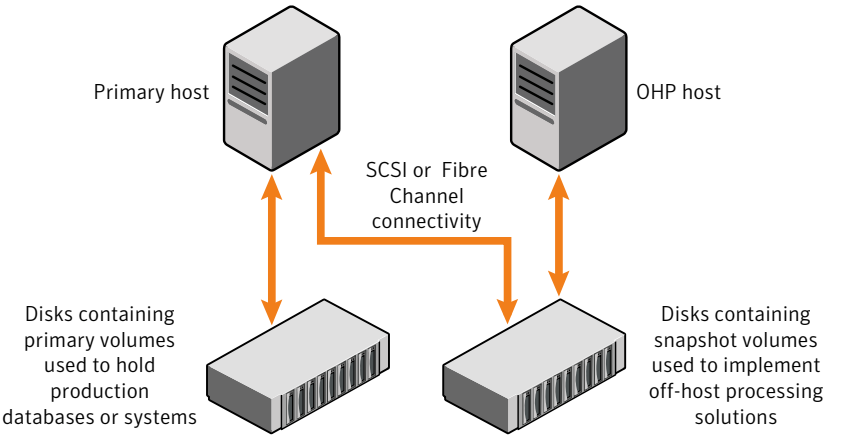
Using linked break-off snapshots makes off-host processing simpler.

For more information about snapshots, see the *Veritas Storage Foundation Advanced Features Administrator's Guide*.

Implemention of off-host processing solutions

Figure 11-1 shows an example implementation of off-host processing.

Figure 11-1 Example implementation of off-host processing



By accessing snapshot volumes from a lightly-loaded host (shown here as the off-host processing (OHP) host), CPU- and I/O-intensive operations for online backup and decision support do not degrade the performance of the primary host that is performing the main production activity (such as running a database). If you also place the snapshot volumes on disks that are attached to different host controllers than the disks in the primary volumes, it is possible to avoid contending with the primary host for I/O resources.

The following sections describe how you can apply off-host processing to implement regular online backup of a volume in a private disk group, and to set up a replica of a production database for decision support. The following applications are outlined:

See "Implementing off-host online backup" on page 407.

See "Implementing decision support" on page 411.

These applications use the Persistent FastResync feature of VxVM in conjunction with linked break-off snapshots.

A volume snapshot represents the data that exists in a volume at a given time. As such, VxVM does not have any knowledge of data that is cached by the overlying file system, or by applications such as databases that have files open in the file system. If you set the `fsgen` volume usage type on a volume that contains a Veritas File System (VxFS), intent logging of the file system metadata ensures the internal consistency of the file system that is backed up. For other file system types, depending on the intent logging capabilities of the file system, there may be potential inconsistencies between in-memory data and the data in the snapshot image.

For databases, you must also use a suitable mechanism to ensure the integrity of tablespace data when the volume snapshot is taken. Most modern database software provides the facility to temporarily suspend file system I/O. 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 when you take the snapshot.

Implementing off-host online backup

This section describes a procedure for implementing off-host online backup for a volume in a private disk group. It outlines how to set up a regular backup cycle. It is beyond the scope of this guide to describe how to configure a database to use this procedure, or how to perform the backup itself.

To back up a volume in a private disk group

1 On the primary host, use the following command to see if the volume is associated with a version 20 data change object (DCO) and DCO volume that allow instant snapshots and Persistent FastResync to be used with the volume:

```
# vxprint -g volumedg -F%instant volume
```

If the volume can be used for instant snapshot operations, this command returns `on`; otherwise, it returns `off`.

If the volume was created under VxVM 4.0 or a later release, and it is not associated with a new-style DCO object and DCO volume, add a version 20 DCO and DCO volume.

See "Preparing a volume for DRL and instant snapshots" on page 371.

If the volume was created before release 4.0 of VxVM, and has any attached snapshot plexes, or is associated with any snapshot volumes, upgrade the volume to use a version 20 DCO.

See "Upgrading existing volumes to use version 20 DCOs" on page 378.

2 On the primary host, use the following command to check whether FastResync is enabled on the volume:

```
# vxprint -g volumedg -F%fastresync volume
```

If FastResync is enabled, this command returns `on`; otherwise, it returns `off`.

If FastResync is disabled, enable it using the following command on the primary host:

```
# vxvol -g volumedg set fastresync=on volume
```

3 On the primary host, create a new volume in a separate disk group for use as the snapshot volume.

For more information about snapshots, see the *Veritas Storage Foundation Advanced Features Administrator's Guide*.

It is recommended that a snapshot disk group is dedicated to maintaining only those disks that are used for off-host processing.

4 On the primary host, link the snapshot volume in the snapshot disk group to the data volume. Enter the following:

```
# vxsnap -g volumedg -b addmir volume mirvol=snapvol \
  mirdg=snapvoldg
```

You can use the `vxsnap snapwait` command to wait for synchronization of the linked snapshot volume to complete. Enter the following:

```
# vxsnap -g volumedg snapwait volume mirvol=snapvol \
  mirdg=snapvoldg
```

This step sets up the snapshot volumes, and starts tracking changes to the original volumes.

When you are ready to create a backup, go to step 5.

5 On the primary host, suspend updates to the volume that contains the database tables. A database may have a hot backup mode that lets you do this by temporarily suspending writes to its tables.

6 On the primary host, create the snapshot volume, *snapvol*, by running the following command:

```
# vxsnap -g volumedg make \
  source=volume/snapvol=snapvol/snapdg=snapvoldg
```

If a database spans more than one volume, you can specify all the volumes and their snapshot volumes using one command, as follows:

```
# vxsnap -g dbasedg make \
  source=vol1/snapvol=snapvol1/snapdg=sdg \
  source=vol2/snapvol=snapvol2/snapdg=sdg \
  source=vol3/snapvol=snapvol3/snapdg=sdg
```

7 On the primary host, if you temporarily suspended updates to a volume in step 5, release all the database tables from hot backup mode.

8 On the primary host, deport the snapshot volume's disk group using the following command:

```
# vxdg deport snapvoldg
```

9 On the OHP host where the backup is to be performed, use the following command to import the snapshot volume's disk group:

```
# vxdg import snapvoldg
```

10 The snapshot volume is initially disabled following the import. On the OHP host, use the following commands to recover and restart the snapshot volume:

```
# vxrecover -g snapvoldg -m snapvol
# vxvol -g snapvoldg start snapvol
```

11 On the OHP host, back up the snapshot volume. If you need to remount the file system in the volume to back it up, first run `fsck` on the volume. The following are sample commands for checking and mounting a file system:

```
# fsck -t vxfs /dev/vx/dsk/snapvoldg/snapvol
# mount -t vxfs /dev/vx/dsk/snapvoldg/snapvol mount_point
```

At this point, back up the file system and use the following command to unmount it:

```
# umount mount_point
```

12 On the OHP host, use the following command to deport the snapshot volume's disk group:

```
# vxdg deport snapvoldg
```

13 On the primary host, re-import the snapshot volume's disk group using the following command:

```
# vxdg import snapvoldg
```

14 The snapshot volume is initially disabled following the import. Use the following commands on the primary host to recover and restart the snapshot volume:

```
# vxrecover -g snapvoldg -m snapvol
# vxvol -g snapvoldg start snapvol
```

15 On the primary host, reattach the snapshot volume to its original volume using the following command:

```
# vxsnap -g snapvoldg reattach snapvol source=vol \
  sourcedg=volumedg
```

For example, to reattach the snapshot volumes `svol1`, `svol2` and `svol3`:

```
# vxsnap -g sdg reattach svol1 \
  source=vol1 sourcedg=dbasedg \
  svol2 source=vol2 sourcedg=dbasedg \
  svol3 source=vol3 sourcedg=dbasedg
```

You can use the `vxsnap snapwait` command to wait for synchronization of the linked snapshot volume to complete:

```
# vxsnap -g volumedg snapwait volume mirvol=snapvol
```

Repeat step 5 through step 15 each time that you need to back up the volume.

Implementing decision support

This section describes a procedure for implementing off-host decision support for a volume in a private disk group. The intention is to present an outline of how to set up a replica database. It is beyond the scope of this guide to describe how to configure a database to use this procedure.

To set up a replica database using the table files that are configured within a volume in a private disk group

1 Use the following command on the primary host to see if the volume is associated with a version 20 data change object (DCO) and DCO volume that allow instant snapshots and Persistent FastResync to be used with the volume:

```
# vxprint -g volumedg -F%instant volume
```

This command returns `on` if the volume can be used for instant snapshot operations; otherwise, it returns `off`.

If the volume was created under VxVM 4.0 or a later release, and it is not associated with a new-style DCO object and DCO volume, it must be prepared.

See "Preparing a volume for DRL and instant snapshots" on page 371.

If the volume was created before release 4.0 of VxVM, and has any attached snapshot plexes, or is associated with any snapshot volumes, it must be upgraded.

See "Upgrading existing volumes to use version 20 DCOs" on page 378.

2 Use the following command on the primary host to check whether FastResync is enabled on a volume:

```
# vxprint -g volumedg -F%fastresync volume
```

This command returns `on` if FastResync is enabled; otherwise, it returns `off`.

If FastResync is disabled, enable it using the following command on the primary host:

```
# vxvol -g volumedg set fastresync=on volume
```

3 Prepare the OHP host to receive the snapshot volume that contains the copy of the database tables. This may involve setting up private volumes to contain any redo logs, and configuring any files that are used to initialize the database.

4 On the primary host, create a new volume in a separate disk group for use as the snapshot volume.

For more information about snapshots, see the *Veritas Storage Foundation Advanced Features Administrator's Guide*.

It is recommended that a snapshot disk group is dedicated to maintaining only those disks that are used for off-host processing.

5 On the primary host, link the snapshot volume in the snapshot disk group to the data volume:

```
# vxsnap -g volumedg -b addmir volume mirvol=snapvol \
  mirdg=snapvoldg
```

You can use the `vxsnap snapwait` command to wait for synchronization of the linked snapshot volume to complete:

```
# vxsnap -g volumedg snapwait volume mirvol=snapvol \
  mirdg=snapvoldg
```

This step sets up the snapshot volumes, and starts tracking changes to the original volumes.

When you are ready to create a replica database, proceed to step 6.

6 On the primary host, suspend updates to the volume that contains the database tables. A database may have a hot backup mode that allows you to do this by temporarily suspending writes to its tables.

7 Create the snapshot volume, *snapvol*, by running the following command on the primary host:

```
# vxsnap -g volumedg make \
  source=volume/snapvol=snapvol/snapdg=snapvoldg
```

If a database spans more than one volume, you can specify all the volumes and their snapshot volumes using one command, as shown in this example:

```
# vxsnap -g dbasedg make \
  source=vol1/snapvol=snapvol1/snapdg=sdg \
  source=vol2/snapvol=snapvol2/snapdg=sdg \
  source=vol3/snapvol=snapvol3/snapdg=sdg
```

This step sets up the snapshot volumes ready for the backup cycle, and starts tracking changes to the original volumes.

8 On the primary host, if you temporarily suspended updates to a volume in step 6, release all the database tables from hot backup mode.

9 On the primary host, deport the snapshot volume's disk group using the following command:

```
# vxdg deport snapvoldg
```

10 On the OHP host where the replica database is to be set up, use the following command to import the snapshot volume's disk group:

```
# vxdg import snapvoldg
```

11 The snapshot volume is initially disabled following the import. Use the following commands on the OHP host to recover and restart the snapshot volume:

```
# vxrecover -g snapvoldg -m snapvol
# vxvol -g snapvoldg start snapvol
```

12 On the OHP host, check and mount the snapshot volume. The following are sample commands for checking and mounting a file system:

```
# fsck -t vxfs /dev/vx/dsk/snapvoldg/snapvol
# mount -t vxfs /dev/vx/dsk/snapvoldg/snapvol mount_point
```

13 On the OHP host, use the appropriate database commands to recover and start the replica database for its decision support role.

At a later time, you can resynchronize the snapshot volume' s data with the primary database.

To refresh the snapshot plexes from the original volume

1 On the OHP host, shut down the replica database, and use the following command to unmount the snapshot volume:

```
# umount mount_point
```

2 On the OHP host, use the following command to deport the snapshot volume's disk group:

```
# vxdg deport snapvoldg
```

3 On the primary host, re-import the snapshot volume's disk group using the following command:

```
# vxdg import snapvoldg
```

4 The snapshot volume is initially disabled following the import. Use the following commands on the primary host to recover and restart the snapshot volume:

```
# vxrecover -g snapvoldg -m snapvol
# vxvol -g snapvoldg start snapvol
```

5 On the primary host, reattach the snapshot volume to its original volume using the following command:

```
# vxsnap -g snapvoldg reattach snapvol source=vol \
  sourcedg=volumedg
```

For example, to reattach the snapshot volumes `svol1`, `svol2` and `svol3`:

```
# vxsnap -g sdg reattach svol1 \
  source=vol1 sourcedg=dbasedg \
  svol2 source=vol2 sourcedg=dbasedg \
  svol3 source=vol3 sourcedg=dbasedg
```

You can use the `vxsnap snapwait` command to wait for synchronization of the linked snapshot volume to complete:

```
# vxsnap -g volumedg snapwait volume mirvol=snapvol
```

You can then proceed to create the replica database, from step 6 in the previous procedure.

See "To set up a replica database using the table files that are configured within a volume in a private disk group" on page 412.

Chapter 12

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 418.

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 147.

For more information about recovering volumes and their data after hardware failure, see the *Veritas Volume Manager 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 421.
  See "Complete disk failure mail messages" on page 422.
  See "Modifying the behavior of hot-relocation" on page 434.
- `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.

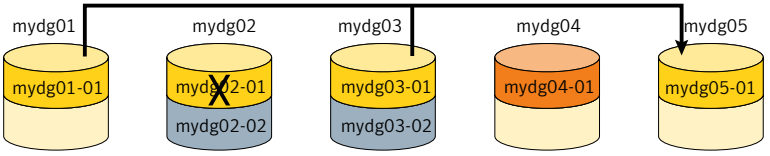
Figure 12-1 shows the hot-relocation process in the case of the failure of a single subdisk of a RAID-5 volume.

Figure 12-1 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 434.

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 147.

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 421.

If the problem is not a cabling error, replace the disk.

See "Removing and replacing disks" on page 147.

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 424.

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 425.

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 426.

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 427.

See "Making a disk available for hot-relocation use" on page 428.

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 428.

After a successful relocation, remove and replace the failed disk.

See "Removing and replacing disks" on page 147.

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 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. Otherwise, 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 VM 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 VM 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 147.

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 vxdiskadm

When a disk has replaced following a failure, you can use the `vxdiskadm` command move the hot-relocated subdisks back to the disk where they originally resided.

To move the relocated subdisks using vxdiskadm

1. Select `Unrelocate subdisks back to a disk` from the `vxdiskadm` main menu.
2. This option prompts for the original disk media name first.

   Enter the disk media name where the hot-relocated subdisks originally resided at the following prompt:

   ```
   Enter the original disk name [<disk>,list,q,?]
   ```

   If there are no hot-relocated subdisks in the system, `vxdiskadm` displays `Currently there are no hot-relocated disks`, and asks you to press Return to continue.
3. You are next asked if you want to move the subdisks to a destination disk other than the original disk.

   ```
   Unrelocate to a new disk [y,n,q,?] (default: n)
   ```

4. If moving subdisks to their original offsets is not possible, you can choose to unrelocate the subdisks forcibly to the specified disk, but not necessarily to the same offsets.

   ```
   Use -f option to unrelocate the subdisks if moving to the exact
   offset fails? [y,n,q,?] (default: n)
   ```

5. If you entered `y` at step 4 to unrelocate the subdisks forcibly, enter `y` or press Return at the following prompt to confirm the operation:

   ```
   Requested operation is to move all the subdisks which were
   hot-relocated from mydg10 back to mydg10 of disk group mydg.
   Continue with operation? [y,n,q,?] (default: y)
   ```

   A status message is displayed at the end of the operation.

   ```
   VxVM INFO V-5-2-954 Unrelocate to disk mydg10 is complete.
   ```

   As an alternative to this procedure, use either the `vxassist` command or the `vxunreloc` command directly.

   See "Moving relocated subdisks using vxassist" on page 431.

   See "Moving relocated subdisks using vxunreloc" on page 431.

Moving relocated subdisks using vxassist

You can use the `vxassist` command to move and unrelocate subdisks. For example, to move the relocated subdisks on `mydg05` belonging to the volume `home` back to `mydg02`, enter the following command.

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 home \!mydg05 mydg02
```

Here, `\!mydg05` specifies the current location of the subdisks, and `mydg02` specifies where the subdisks should be relocated.

If the volume is enabled, subdisks within detached or disabled plexes, and detached log or RAID-5 subdisks, are moved without recovery of data.

If the volume is not enabled, subdisks within STALE or OFFLINE plexes, and stale log or RAID-5 subdisks, are moved without recovery. If there are other subdisks within a non-enabled volume that require moving, the relocation fails.

For enabled subdisks in enabled plexes within an enabled volume, data is moved to the new location, without loss of either availability or redundancy of the volume.

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 /etc/init.d/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 13

Administering cluster functionality (CVM)

This chapter includes the following topics:

Overview of clustering

Tightly-coupled cluster systems are common in the realm of enterprise-scale mission-critical data processing. The primary advantage of clusters is protection against hardware failure. Should the primary node fail or otherwise become unavailable, applications can continue to run by transferring their execution to standby nodes in the cluster. This ability to provide continuous availability of service by switching to redundant hardware is commonly termed failover.

Another major advantage of clustered systems is their ability to reduce contention for system resources caused by activities such as backup, decision support and report generation. Businesses can derive enhanced value from their investment in cluster systems by performing such operations on lightly loaded nodes in the cluster rather than on the heavily loaded nodes that answer requests for service. This ability to perform some operations on the lightly loaded nodes is commonly termed load balancing.

Overview of cluster volume management

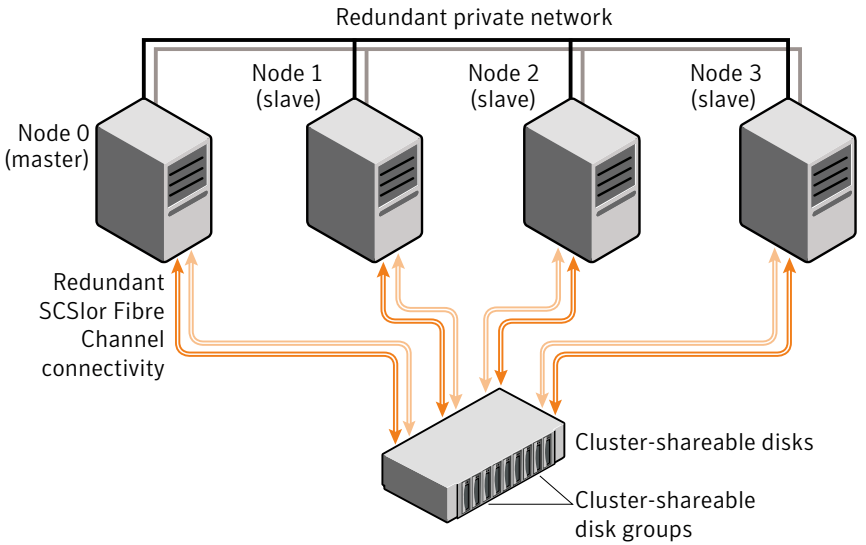
Over the past several years, parallel applications using shared data access have become increasingly popular. Examples of commercially available applications include Oracle Real Application Clusters™ (RAC), Sybase Adaptive Server®, and Informatica Enterprise Cluster Edition. In addition, the semantics of Network File System (NFS), File Transfer Protocol (FTP), and Network News Transfer Protocol (NNTP) allow these workloads to be served by shared data access clusters. Finally, numerous organizations have developed internal applications that take advantage of shared data access clusters.

The cluster functionality of VxVM (CVM) works together with the cluster monitor daemon that is provided by VCS or by the host operating system. The cluster monitor informs VxVM of changes in cluster membership. Each node starts up independently and has its own cluster monitor plus its own copies of the operating system and VxVM/CVM. When a node joins a cluster, it gains access to shared disk groups and volumes. When a node leaves a cluster, it loses access to these shared objects. A node joins a cluster when you issue the appropriate command on that node.

Warning: The CVM functionality of VxVM is supported only when used in conjunction with a cluster monitor that has been configured correctly to work with VxVM.

Figure 13-1 shows a simple cluster arrangement consisting of four nodes with similar or identical hardware characteristics (CPUs, RAM and host adapters), and configured with identical software (including the operating system).

Figure 13-1 Example of a 4-node CVM cluster



To the cluster monitor, all nodes are the same. VxVM objects configured within shared disk groups can potentially be accessed by all nodes that join the cluster. However, the CVM functionality of VxVM requires that one node act as the master node; all other nodes in the cluster are slave nodes. Any node is capable of being the master node, and it is responsible for coordinating certain VxVM activities.

In this example, node 0 is configured as the CVM master node and nodes 1, 2 and 3 are configured as CVM slave nodes. The nodes are fully connected by a private network and they are also separately connected to shared external storage (either disk arrays or JBODs: just a bunch of disks) via SCSI or Fibre Channel in a Storage Area Network (SAN).

In this example, each node has two independent paths to the disks, which are configured in one or more cluster-shareable disk groups. Multiple paths provide resilience against failure of one of the paths, but this is not a requirement for cluster configuration. Disks may also be connected by single paths.

The private network allows the nodes to share information about system resources and about each other's state. Using the private network, any node can recognize which other nodes are currently active, which are joining or leaving the cluster, and which have failed. The private network requires at least two communication channels to provide redundancy against one of the channels failing. If only one channel were used, its failure would be indistinguishable from node failure—a condition known as network partitioning.

You can run commands that configure or reconfigure VxVM objects on any node in the cluster. These tasks include setting up shared disk groups, creating and reconfiguring volumes, and performing snapshot operations.

The first node to join a cluster performs the function of master node. If the master node leaves a cluster, one of the slave nodes is chosen to be the new master.

Private and shared disk groups

The following types of disk groups are defined:

Private disk group	Belongs to only one node. A private disk group can only be imported by one system. LUNs in a private disk group may be physically accessible from one or more systems, but access is restricted to only one system at a time. The boot disk group (usually aliased by the reserved disk group name `bootdg`) is always a private disk group.
Shared disk group	Can be shared by all nodes. A shared (or cluster-shareable) disk group is imported by all cluster nodes. LUNs in a shared disk group must be physically accessible from all systems that may join the cluster.

In a CVM cluster, most disk groups are shared. LUNs in a shared disk group are accessible from all nodes in a cluster, allowing applications on multiple cluster nodes to simultaneously access the same LUN. A volume in a shared disk group can be simultaneously accessed by more than one node in the cluster, subject to license key and disk group activation mode restrictions.

You can use the `vxdg` command to designate a disk group as cluster-shareable.

See "Importing disk groups as shared" on page 467.

When a disk group is imported as cluster-shareable for one node, each disk header is marked with the cluster ID. As each node subsequently joins the cluster, it recognizes the disk group as being cluster-shareable and imports it. In contrast, a private disk group's disk headers are marked with the individual node's host name. As system administrator, you can import or deport a shared disk group at any time; the operation takes place in a distributed fashion on all nodes.

Each LUN is marked with a unique disk ID. When cluster functionality for VxVM starts on the master, it imports all shared disk groups (except for any that do not have the `autoimport` attribute set). When a slave tries to join a cluster, the master sends it a list of the disk IDs that it has imported, and the slave checks to see if it can access them all. If the slave cannot access one of the listed disks, it abandons its attempt to join the cluster. If it can access all of the listed disks, it joins the cluster and imports the same shared disk groups as the master. When a node

leaves the cluster gracefully, it deports all its imported shared disk groups, but they remain imported on the surviving nodes.

Reconfiguring a shared disk group is performed with the cooperation of all nodes. Configuration changes to the disk group are initiated by the master, and happen simultaneously on all nodes and the changes are identical. Such changes are atomic in nature, which means that they either occur simultaneously on all nodes or not at all.

Whether all members of the cluster have simultaneous read and write access to a cluster-shareable disk group depends on its activation mode setting.

See "Activation modes of shared disk groups" on page 441.

The data contained in a cluster-shareable disk group is available as long as at least one node is active in the cluster. The failure of a cluster node does not affect access by the remaining active nodes. Regardless of which node accesses a cluster-shareable disk group, the configuration of the disk group looks the same.

Warning: Applications running on each node can access the data on the VM disks simultaneously. VxVM does not protect against simultaneous writes to shared volumes by more than one node. It is assumed that applications control consistency (by using Veritas Cluster File System or a distributed lock manager, for example).

Activation modes of shared disk groups

A shared disk group must be activated on a node in order for the volumes in the disk group to become accessible for application I/O from that node. The ability of applications to read from or to write to volumes is dictated by the activation mode of a shared disk group. Valid activation modes for a shared disk group are `exclusivewrite`, `readonly`, `sharedread`, `sharedwrite`, and `off` (inactive).

The default activation mode for shared disk groups is `sharedwrite`.

Special uses of clusters, such as high availability (HA) applications and off-host backup, can use disk group activation to explicitly control volume access from different nodes in the cluster

Table 13-1 describes the activation modes.

Table 13-1 Activation modes for shared disk groups

Activation mode	Description
`exclusivewrite (ew)`	The node has exclusive write access to the disk group. No other node can activate the disk group for write access.

Table 13-1 Activation modes for shared disk groups *(continued)*

Activation mode	Description
`readonly (ro)`	The node has read access to the disk group and denies write access for all other nodes in the cluster. The node has no write access to the disk group. Attempts to activate a disk group for either of the write modes on other nodes fail.
`sharedread (sr)`	The node has read access to the disk group. The node has no write access to the disk group, however other nodes can obtain write access.
`sharedwrite (sw)`	The node has write access to the disk group. Attempts to activate the disk group for shared read and shared write access succeed. Attempts to activate the disk group for exclusive write and read-only access fail.
`off`	The node has neither read nor write access to the disk group. Query operations on the disk group are permitted.

Table 13-2 summarizes the allowed and conflicting activation modes for shared disk groups.

Table 13-2 Allowed and conflicting activation modes

Disk group activated in cluster as...	Attempt to activate disk group on another node as...			
	exclusive-write	readonly	sharedread	sharedwrite
exclusivewrite	Fails	Fails	Succeeds	Fails
readonly	Fails	Succeeds	Succeeds	Fails
sharedread	Succeeds	Succeeds	Succeeds	Succeeds
sharedwrite	Fails	Fails	Succeeds	Succeeds

Shared disk groups can be automatically activated in a specified mode when the disk group is created or imported. To control automatic activation of shared disk groups, create a defaults file `/etc/default/vxdg` containing the following lines:

```
enable_activation=true
default_activation_mode=activation-mode
```

The *activation-mode* is one of `exclusivewrite`, `readonly`, `sharedread`, `sharedwrite`, or `off`.

When a shared disk group is created or imported, it is activated in the specified mode. When a node joins the cluster, all shared disk groups accessible from the node are activated in the specified mode.

The activation mode of a disk group controls volume I/O from different nodes in the cluster. It is not possible to activate a disk group on a given node if it is activated in a conflicting mode on another node in the cluster. When enabling activation using the defaults file, it is recommended that the file be consistent on all nodes in the cluster as in Table 13-2. Otherwise, the results of activation are unpredictable.

If the defaults file is edited while the `vxconfigd` daemon is already running, run the `/sbin/vxconfigd -k -x syslog` command on all nodes to restart the process.

If the default activation mode is anything other than `off`, an activation following a cluster join, or a disk group creation or import can fail if another node in the cluster has activated the disk group in a conflicting mode.

To display the activation mode for a shared disk group, use the `vxdg list` *`diskgroup`* command.

See "Listing shared disk groups" on page 466.

You can also use the `vxdg` command to change the activation mode on a shared disk group.

See "Changing the activation mode on a shared disk group" on page 470.

It is also possible to configure a volume so that it can only be opened by a single node in a cluster.

See "Creating volumes with exclusive open access by a node" on page 471.

See "Setting exclusive open access to a volume by a node" on page 471.

Connectivity policy of shared disk groups

A shared disk group provides concurrent read and write access to the volumes that it contains for all nodes in a cluster. A shared disk group can be created on any node of the cluster. This has the following advantages and implications:

- All nodes in the cluster see exactly the same configuration.
- Commands to change the configuration are sent to the master node.
- Any changes on the master node are automatically coordinated and propagated to the slave nodes in the cluster.

- Any failures that require a configuration change must be sent to the master node so that they can be resolved correctly.
- As the master node resolves failures, all the slave nodes are correctly updated. This ensures that all nodes have the same view of the configuration.

The practical implication of this design is that I/O failure on any node results in the configuration of all nodes being changed. This is known as the global detach policy. However, in some cases, it is not desirable to have all nodes react in this way to I/O failure. To address this, an alternate way of responding to I/O failures, known as the local detach policy, was introduced.

The local detach policy is intended for use with shared mirrored volumes in a cluster. This policy prevents I/O failure on any of the nodes in the cluster from causing a plex to be detached. This would require the plex to be resynchronized when it is subsequently reattached.

The local detach policy is supported for disk groups that have a version number of 120 or greater.

For small mirrored volumes, non-mirrored volumes, volumes that use hardware mirrors, and volumes in private disk groups, there is no benefit in configuring the local detach policy. In most cases, it is recommended that you use the default global detach policy.

The choice between local and global detach polices is one of node availability versus plex availability when an individual node loses access to disks. Select the local detach policy for a diskgroup if you are using mirrored volumes within it, and would prefer a single node to lose write access to a volume rather than a plex of the volume being detached clusterwide. i.e. you consider the availability of your data (retaining mirrors) more important than any one node in the cluster. This will typically only apply in larger clusters, and where a parallel application is being used that can seamlessly provide the same service from the other nodes. For example, this option is not appropriate for fast failover configurations. Select the global detach policy in all other cases.

In the event of the master node losing access to all the disks containing log/config copies, the disk group failure policy is triggered. At this point no plexes can be detached, as this requires access to the log/config copies, no configuration changes to the disk group can be made, and any action requiring the kernel to write to the klog (first open, last close, mark dirty etc) will fail. If this happened in releases prior to 4.1, the master node always disabled the disk group. Release 4.1 introduces the disk group failure policy, which allows you to change this behavior for critical disk groups. This policy is only supported for disk groups that have a version number of 120 or greater.

Global detach policy

The global detach policy is the traditional and default policy for all nodes on the configuration. If there is a read or write I/O failure on a slave node, the master node performs the usual I/O recovery operations to repair the failure, and, if required, the plex is detached cluster-wide. All nodes remain in the cluster and continue to perform I/O, but the redundancy of the mirrors is reduced. When the problem that caused the I/O failure has been corrected, the disks should be re-attached and the mirrors that were detached must be recovered before the redundancy of the data can be restored.

Local detach policy

The local detach policy is designed to support failover applications in large clusters where the redundancy of the volume is more important than the number of nodes that can access the volume. If there is a write failure on any node, the usual I/O recovery operations are performed to repair the failure, and additionally all the nodes are contacted to see if the disk is still accessible to them. If the write failure is local, and only seen by a single node, I/O is stopped for the node that first saw the failure, and an error is returned to the application using the volume. The write failure is global if more than one node sees the failure. The volume is not disabled.

If required, configure the cluster management software to move the application to a different node, and/or remove the node that saw the failure from the cluster. The volume continues to return write errors, as long as one mirror of the volume has an error. The volume continues to satisfy read requests as long as one good plex is available.

If the reason for the I/O error is corrected and the node is still a member of the cluster, it can resume performing I/O from/to the volume without affecting the redundancy of the data.

The `vxdg` command can be used to set the disk detach policy on a shared disk group.

See "Setting the disk detach policy on a shared disk group" on page 470.

Table 13-3 summarizes the effect on a cluster of I/O failure to the disks in a mirrored volume.

Table 13-3 Cluster behavior under I/O failure to a mirrored volume for different disk detach policies

Type of I/O failure	Local (diskdetpolicy=local)	Global (diskdetpolicy=global)
Failure of path to one disk in a volume for a single node	Reads fail only if no plexes remain available to the affected node. Writes to the volume fail.	The plex is detached, and I/O from/to the volume continues. An I/O error is generated if no plexes remain.
Failure of paths to all disks in a volume for a single node	I/O fails for the affected node.	The plex is detached, and I/O from/to the volume continues. An I/O error is generated if no plexes remain.
Failure of one or more disks in a volume for all nodes.	The plex is detached, and I/O from/to the volume continues. An I/O error is generated if no plexes remain.	The plex is detached, and I/O from/to the volume continues. An I/O error is generated if no plexes remain.

Guidelines for choosing detach policies

In most cases it is recommended that you use the global detach policy, and particularly if any of the following conditions apply:

- When cluster-wide access to the shared data volumes is more critical than retaining data redundancy.
- If only non-mirrored, small mirrored, or hardware mirrored volumes are configured. In these cases, the global detach policy avoids the system overhead of the extra messaging that the local detach policy requires.

The local detach policy may be suitable in the following cases:

- When large mirrored volumes are configured–Resynchronizing a reattached plex can degrade system performance. The local detach policy can avoid the need to detach the plex at all. (Alternatively, the dirty region logging (DRL) feature can be used to reduce the amount of resynchronization that is required.)
- For clusters with more than four nodes–Keeping an application running on a particular node is less critical when there are many nodes in a cluster. It may be possible to configure the cluster management software to move an application to a node that has access to the volumes. In addition, load balancing may be able to move applications to a different volume from the one that experienced the I/O problem. This preserves data redundancy, and other nodes may still be able to perform I/O from/to the volumes on the disk.

Disk group failure policy

The local detach policy by itself is insufficient to determine the desired behavior if the master node loses access to all disks that contain copies of the configuration database and logs. In this case, the disk group is disabled. As a result, any action that would result in an update to log/config copy will also fail from the other nodes in the cluster. In release 4.1, the disk group failure policy is introduced to determine the behavior of the master node in such cases.

Table 13-4 shows how the behavior of the master node changes according to the setting of the failure policy.

Table 13-4 Behavior of master node for different failure policies

Type of I/O failure	Leave (dgfailpolicy= leave)	Disable (dgfailpolicy= dgdisable)	Request Leave (dgfailpolicy= requestleave)
Master node loses access to all copies of the logs.	The master node panics with the message “klog update failed” for a failed kernel-initiated transaction, or “cvm config update failed” for a failed user-initiated transaction.	The master node disables the disk group.	The master node leaves the cluster, after VCS handles all the applications dependent upon shared storage by either gracefully stopping them or failing them over to other nodes of the cluster.

The behavior of the master node under the disk group failure policy is independent of the setting of the disk detach policy. If the disk group failure policy is set to `leave`, all nodes panic in the unlikely case that none of them can access the log copies.

If the disk group failure policy is set to `requestleave`, the master node gracefully leaves the cluster if the master node loses access to all log/config copies of the disk group. If the master node loses access to the log/config copies of a shared disk group, Cluster Volume Manager (CVM) signals the CVM Cluster Veritas Cluster Server agent. Veritas Cluster Server (VCS) attempts to take offline the CVM group on the master node. When the CVM group is taken offline, the dependent services groups are also taken offline. If the dependent applications managed by VCS cannot be taken offline for some reason, the master node may not be able to leave the cluster gracefully.

The `vxdg` command can be used to set the failure policy on a shared disk group.

See "Setting the disk group failure policy on a shared disk group" on page 470.

Guidelines for failure policies

If you have a critical disk group that you do not want to become disabled in the case that the master node loses access to the copies of the logs, set the disk group failure policy to `leave` or `requestleave`. This prevents I/O failure on the master node disabling the disk group. However, critical applications running on the master node fail if they lose access to the other shared disk groups. In such a case, it may be preferable to set the policy to `dgdisable`, and to allow the disk group to be disabled.

Note: The `requestleave` disk group failure policy is supported only for disk groups containing volumes that have a single plex and that do not have a DCO log attached.

The default settings for the detach and failure policies are `global` and `dgdisable` respectively. You can use the `vxdg` command to change both the detach and failure policies on a shared disk group, as shown in this example:

```
# vxdg -g diskgroup set diskdetpolicy=local dgfailpolicy=leave
```

Effect of disk connectivity on cluster reconfiguration

The detach policy, previous I/O errors, or access to disks are not considered when a new master node is chosen. When the master node leaves a cluster, the node that takes over as master of the cluster may already have seen I/O failures for one or more disks. Under the local detach policy, if a node was affected before reconfiguration, and this node then becomes the master, the failure is treated differently from the global detach policy case.

Some failure scenarios do not result in a disk group failure policy being invoked, but can potentially impact the cluster. For example, if the local disk detach policy is in effect, and the new master node has a failed plex, this results in all nodes detaching the plex because the new master is unaffected by the policy.

The detach policy does not change the requirement that a node joining a cluster must have access to all the disks in all shared disk groups. Similarly, a node that is removed from the cluster because of an I/O failure cannot rejoin the cluster until this requirement is met.

Limitations of shared disk groups

Only raw device access may be performed via CVM. It does not support shared access to file systems in shared volumes unless the appropriate software, such as Veritas Cluster File System, is installed and configured.

Note: The boot disk group (usually aliased as `bootdg`) cannot be made cluster-shareable. It must be private.

The cluster functionality of VxVM does not support RAID-5 volumes, or task monitoring for cluster-shareable disk groups. These features can, however, be used in private disk groups that are attached to specific nodes of a cluster.

If you have RAID-5 volumes in a private disk group that you wish to make shareable, you must first relayout the volumes as a supported volume type such as `stripe-mirror` or `mirror-stripe`. Online relayout of shared volumes is supported provided that it does not involve RAID-5 volumes.

If a shared disk group contains RAID-5 volumes, deport it and then reimport the disk group as private on one of the cluster nodes. Reorganize the volumes into layouts that are supported for shared disk groups, and then deport and reimport the disk group as shared.

Multiple host failover configurations

Outside the context of CVM, VxVM disk groups can be imported (made available) on only one host at any given time. When a host imports a (private) disk group, the volumes and configuration of that disk group become accessible to the host. If the administrator or system software wants to privately use the same disk group from another host, the host that already has the disk group imported (importing host) must deport (give up access to) the disk group. Once deported, the disk group can be imported by another host.

If two hosts are allowed to access a disk group concurrently without proper synchronization, such as that provided by Oracle RAC, the configuration of the disk group, and possibly the contents of volumes, can be corrupted. Similar corruption can also occur if a file system or database on a raw disk partition is accessed concurrently by two hosts, so this problem in not limited to Veritas Volume Manager.

Import lock

When a host in a non-CVM environment imports a disk group, an import lock is written on all disks in that disk group. The import lock is cleared when the host

deports the disk group. The presence of the import lock prevents other hosts from importing the disk group until the importing host has deported the disk group.

Specifically, when a host imports a disk group, the import normally fails if any disks within the disk group appear to be locked by another host. This allows automatic re-importing of disk groups after a reboot (autoimporting) and prevents imports by another host, even while the first host is shut down. If the importing host is shut down without deporting the disk group, the disk group can only be imported by another host by clearing the host ID lock first (discussed later).

The import lock contains a host ID (the host name) reference to identify the importing host and enforce the lock. Problems can therefore arise if two hosts have the same host ID.

Since Veritas Volume Manager uses the host name as the host ID (by default), it is advisable to change the host name of one machine if another machine shares its host name. To change the host name, use the `vxdctl hostid new_hostname` command.

Failover

The import locking scheme works well in an environment where disk groups are not normally shifted from one system to another. However, consider a setup where two hosts, Node A and Node B, can access the drives of a disk group. The disk group is initially imported by Node A, but the administrator wants to access the disk group from Node B if Node A crashes. Such a failover scenario can be used to provide manual high availability to data, where the failure of one node does not prevent access to data. Failover can be combined with a “high availability” monitor to provide automatic high availability to data: when Node B detects that Node A has crashed or shut down, Node B imports (fails over) the disk group to provide access to the volumes.

Veritas Volume Manager can support failover, but it relies on the administrator or on an external high-availability monitor, such as VCS, to ensure that the first system is shut down or unavailable before the disk group is imported to another system.

See “Moving disk groups between systems” on page 238.

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

Corruption of disk group configuration

If `vxdg import` is used with `-C` (clears locks) and/or `-f` (forces import) to import a disk group that is still in use from another host, disk group configuration corruption is likely to occur. Volume content corruption is also likely if a file

system or database is started on the imported volumes before the other host crashes or shuts down.

If this kind of corruption occurs, your configuration must typically be rebuilt from scratch and all data be restored from a backup. There are typically numerous configuration copies for each disk group, but corruption nearly always affects all configuration copies, so redundancy does not help in this case.

As long as the configuration backup daemon, `vxconfigbackupd`, is running, VxVM will backup configurations whenever the configuration is changed. By default, backups are stored in `/etc/vx/cbr/bk`. You may also manually backup the configuration using the `vxconfigbackup` utility. The configuration can be rebuilt using the `vxrestore` utility.

See the `vxconfigbackup`, `vxconfigbackupd`, `vxconfigrestore` man pages.

Disk group configuration corruption usually shows up as missing or duplicate records in the configuration databases. This can result in a variety of `vxconfigd` error messages

```
VxVM vxconfigd ERROR
V-5-1-569 Disk group group,Disk disk:
Cannot auto-import group: reason
```

where the *reason* can describe errors such as:

```
Association not resolved
Association count is incorrect
Duplicate record in configuration
Configuration records are inconsistent
```

These errors are typically reported in association with specific disk group configuration copies, but usually apply to all copies. The following is usually displayed along with the error:

```
Disk group has no valid configuration copies
```

See the *Veritas Volume Manager Troubleshooting Guide*.

About the cluster functionality of VxVM

A cluster consists of a number of hosts or nodes that share a set of disks. The following are the main benefits of cluster configurations:

Availability	If one node fails, the other nodes can still access the shared disks. When configured with suitable software, mission-critical applications can continue running by transferring their execution to a standby node in the cluster. This ability to provide continuous uninterrupted service by switching to redundant hardware is commonly termed failover. Failover is transparent to users and high-level applications for database and file-sharing. You must configure cluster management software, such as Veritas Cluster Server (VCS), to monitor systems and services, and to restart applications on another node in the event of either hardware or software failure. VCS also allows you to perform general administration tasks such as making nodes join or leave a cluster. Note that a standby node need not remain idle. It could be used to serve other applications in parallel.
Off-host processing	Clusters can reduce contention for system resources by performing activities such as backup, decision support and report generation on the more lightly loaded nodes of the cluster. This allows businesses to derive enhanced value from their investment in cluster systems.

The Cluster Volume Manager (CVM) is capable of supporting clusters with up to 64 nodes.

The nodes can simultaneously access and manage a set of disks or LUNs under VxVM control. The same logical view of disk configuration and any changes to this view are available on all the nodes. When the CVM functionality is enabled, all cluster nodes can share VxVM objects such as shared disk groups. Private disk groups are supported in the same way as in a non-clustered environment. This chapter discusses the cluster functionality that is provided with VxVM.

Veritas Dynamic Multi-Pathing (DMP) can be used in a clustered environment.

See “DMP in a clustered environment” on page 165.

Campus cluster configurations (also known as stretch cluster or remote mirror configurations) can also be configured and administered.

See “About sites and remote mirrors” on page 477.

CVM initialization and configuration

Before any nodes can join a new cluster for the first time, you must supply certain configuration information during cluster monitor setup. This information is normally stored in some form of cluster monitor configuration database. The

precise content and format of this information depends on the characteristics of the cluster monitor. The information required by VxVM is as follows:

- Cluster ID
- Node IDs
- Network addresses of nodes
- Port addresses

When a node joins the cluster, this information is automatically loaded into VxVM on that node at node startup time.

Note: The CVM functionality of VxVM is supported only when used with a cluster monitor that has been configured correctly to work with VxVM.

Use a cluster monitor such as GAB (Group Membership and Atomic Broadcast) in Veritas Cluster Service (VCS). For a VCS environment, use the `vxcvmconfig` command on any node to configure the cluster to use the CVM functionality of VxVM. The `vxcvmconfig` command is not included with Veritas Volume Manager.

The cluster monitor startup procedure effects node initialization, and brings up the various cluster components (such as VxVM with cluster support, the cluster monitor, and a distributed lock manager) on the node. Once this is complete, applications may be started. The cluster monitor startup procedure must be invoked on each node to be joined to the cluster.

For VxVM in a cluster environment, initialization consists of loading the cluster configuration information and joining the nodes in the cluster. The first node to join becomes the master node, and later nodes (slaves) join to the master. If two nodes join simultaneously, VxVM chooses the master. After a given node joins, that node has access to the shared disk groups and volumes.

Cluster reconfiguration

Cluster reconfiguration occurs if a node leaves or joins a cluster. Each node's cluster monitor continuously watches the other cluster nodes. When the membership of the cluster changes, the cluster monitor informs VxVM for it to take appropriate action.

During cluster reconfiguration, VxVM suspends I/O to shared disks. I/O resumes when the reconfiguration completes. Applications may appear to freeze for a short time during reconfiguration.

If other operations, such as VxVM operations or recoveries, are in progress, cluster reconfiguration can be delayed until those operations complete. Volume

reconfigurations do not take place at the same time as cluster reconfigurations. Depending on the circumstances, an operation may be held up and restarted later. In most cases, cluster reconfiguration takes precedence. However, if the volume reconfiguration is in the commit stage, it completes first.

See "Volume reconfiguration" on page 456.

See "vxclustadm utility" on page 454.

vxclustadm utility

The `vxclustadm` command provides an interface to the CVM functionality of VxVM when VCS is used as the cluster monitor. It is also called during cluster startup and shutdown. In the absence of a cluster monitor, `vxclustadm` can also be used to activate or deactivate the CVM functionality of VxVM on any node in a cluster.

The `startnode` keyword to `vxclustadm` starts CVM functionality on a cluster node by passing cluster configuration information to the VxVM kernel. In response to this command, the kernel and the VxVM configuration daemon, `vxconfigd`, perform initialization.

The `stopnode` keyword stops CVM functionality on a node. It waits for all outstanding I/O to complete and for all applications to close shared volumes.

The `setmaster` keyword migrates the CVM master to the specified node. The migration is an online operation. Symantec recommends that you switch the master when the cluster is not handling VxVM configuration changes or cluster reconfiguration operations.

The `reinit` keyword allows nodes to be added to or removed from a cluster without stopping the cluster. Before running this command, the cluster configuration file must have been updated with information about the supported nodes in the cluster.

The `nidmap` keyword prints a table showing the mapping between CVM node IDs in VxVM's cluster-support subsystem and node IDs in the cluster monitor. It also prints the state of the nodes in the cluster.

The `nodestate` keyword reports the state of a cluster node and also the reason for the last abort of the node as shown in this example:

```
# vxclustadm nodestate

state: out of cluster
reason: user initiated stop
```

Table 13-5 lists the various reasons that may be given for a node abort.

Table 13-5 Node abort messages

Reason	Description
`cannot find disk on slave node`	Missing disk or bad disk on the slave node.
`cannot obtain configuration data`	The node cannot read the configuration data due to an error such as disk failure.
`cluster device open failed`	Open of a cluster device failed.
`clustering license mismatch with master node`	Clustering license does not match that on the master node.
`clustering license not available`	Clustering license cannot be found.
`connection refused by master`	The join operation of a node refused by the master node.
`disk in use by another cluster`	A disk belongs to a cluster other than the one that a node is joining.
`join timed out during reconfiguration`	The join operation of a node has timed out due to reconfiguration taking place in the cluster.
`klog update failed`	Cannot update kernel log copies during the join operation of a node.
`master aborted during join`	Master node aborted while another node was joining the cluster.
`protocol version out of range`	Cluster protocol version mismatch or unsupported version.
`recovery in progress`	Volumes that were opened by the node are still recovering.
`transition to role failed`	Changing the role of a node to be the master failed.
`user initiated abort`	Node is out of cluster due to an abort initiated by the user or by the cluster monitor.
`user initiated stop`	Node is out of cluster due to a stop initiated by the user or by the cluster monitor.
`vxconfigd is not enabled`	The VxVM configuration daemon is not enabled.

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

Volume reconfiguration

Volume reconfiguration is the process of creating, changing, and removing VxVM objects such as disk groups, volumes and plexes. In a cluster, all nodes cooperate to perform such operations. The `vxconfigd` daemons play an active role in volume reconfiguration. For reconfiguration to succeed, a `vxconfigd` daemon must be running on each of the nodes.

See "vxconfigd daemon" on page 457.

A volume reconfiguration transaction is initiated by running a VxVM utility on the master node. The utility contacts the local `vxconfigd` daemon on the master node, which validates the requested change. For example, `vxconfigd` rejects an attempt to create a new disk group with the same name as an existing disk group. The `vxconfigd` daemon on the master node then sends details of the changes to the `vxconfigd` daemons on the slave nodes. The `vxconfigd` daemons on the slave nodes then perform their own checking. For example, each slave node checks that it does not have a private disk group with the same name as the one being created. If the operation involves a new disk, each node checks that it can access that disk. When the `vxconfigd` daemons on all the nodes agree that the proposed change is reasonable, each notifies its kernel. The kernels then cooperate to either commit or to abandon the transaction. Before the transaction can be committed, all of the kernels ensure that no I/O is underway, and block any I/O issued by applications until the reconfiguration is complete. The master node is responsible both for initiating the reconfiguration, and for coordinating the commitment of the transaction. The resulting configuration changes appear to occur simultaneously on all nodes.

If a `vxconfigd` daemon on any node goes away during reconfiguration, all nodes are notified and the operation fails. If any node leaves the cluster, the operation fails unless the master has already committed it. If the master node leaves the cluster, the new master node, which was previously a slave node, completes or fails the operation depending on whether or not it received notification of successful completion from the previous master node. This notification is performed in such a way that if the new master does not receive it, neither does any other slave.

If a node attempts to join a cluster while a volume reconfiguration is being performed, the result of the reconfiguration depends on how far it has progressed. If the kernel has not yet been invoked, the volume reconfiguration is suspended until the node has joined the cluster. If the kernel has been invoked, the node waits until the reconfiguration is complete before joining the cluster.

When an error occurs, such as when a check on a slave fails or a node leaves the cluster, the error is returned to the utility and a message is sent to the console on the master node to identify on which node the error occurred.

vxconfigd daemon

The VxVM configuration daemon, `vxconfigd`, maintains the configuration of VxVM objects. It receives cluster-related instructions from the kernel. A separate copy of `vxconfigd` runs on each node, and these copies communicate with each other over a network. When invoked, a VxVM utility communicates with the `vxconfigd` daemon running on the same node; it does not attempt to connect with `vxconfigd` daemons on other nodes. During cluster startup, the kernel prompts `vxconfigd` to begin cluster operation and indicates whether it is a master node or a slave node.

When a node is initialized for cluster operation, the `vxconfigd` daemon is notified that the node is about to join the cluster and is provided with the following information from the cluster monitor configuration database:

- cluster ID
- node IDs
- master node ID
- role of the node
- network address of the node

On the master node, the `vxconfigd` daemon sets up the shared configuration by importing shared disk groups, and informs the kernel when it is ready for the slave nodes to join the cluster.

On slave nodes, the `vxconfigd` daemon is notified when the slave node can join the cluster. When the slave node joins the cluster, the `vxconfigd` daemon and the VxVM kernel communicate with their counterparts on the master node to set up the shared configuration.

When a node leaves the cluster, the kernel notifies the `vxconfigd` daemon on all the other nodes. The master node then performs any necessary cleanup. If the master node leaves the cluster, the kernels select a new master node and the `vxconfigd` daemons on all nodes are notified of the choice.

The `vxconfigd` daemon also participates in volume reconfiguration.

See "Volume reconfiguration" on page 456.

vxconfigd daemon recovery

In a cluster, the `vxconfigd` daemons on the slave nodes are always connected to the `vxconfigd` daemon on the master node. If the `vxconfigd` daemon is stopped, volume reconfiguration cannot take place. However, a new node can join the cluster, as long as the `vxconfigd` daemon is running on the master node.

If the vxconfigd daemon stops, different actions are taken depending on which node this occurred:

- If the vxconfigd daemon is stopped on the master node, the vxconfigd daemons on the slave nodes periodically attempt to rejoin to the master node. Such attempts do not succeed until the vxconfigd daemon is restarted on the master. In this case, the vxconfigd daemons on the slave nodes have not lost information about the shared configuration, so that any displayed configuration information is correct.
- If the vxconfigd daemon is stopped on a slave node, the master node takes no action. When the vxconfigd daemon is restarted on the slave, the slave vxconfigd daemon attempts to reconnect to the master daemon and to re-acquire the information about the shared configuration. (Neither the kernel view of the shared configuration nor access to shared disks is affected.) Until the vxconfigd daemon on the slave node has successfully reconnected to the vxconfigd daemon on the master node, it has very little information about the shared configuration and any attempts to display or modify the shared configuration can fail. For example, shared disk groups listed using the vxdg list command are marked as disabled; when the rejoin completes successfully, they are marked as enabled.
- If the vxconfigd daemon is stopped on both the master and slave nodes, the slave nodes do not display accurate configuration information until vxconfigd is restarted on the master and slave nodes, and the daemons have reconnected.

If the CVM agent for VCS determines that the vxconfigd daemon has stopped on a node, vxconfigd is restarted automatically.

Warning: The -r reset option to vxconfigd restarts the vxconfigd daemon and recreates all states from scratch. This option cannot be used to restart vxconfigd while a node is joined to a cluster because it causes cluster information to be discarded.

To restart vxconfigd manually

1 Use the following command to disable failover on any service groups that contain VxVM objects:

```
# hagrp -freeze groupname
```

2 Enter the following command to stop and restart the VxVM configuration daemon on the affected node:

```
# vxconfigd -k
```

3 Use the following command to re-enable failover for the service groups that you froze in step 1:

```
# hagrp -unfreeze groupname
```

Node shutdown

Although it is possible to shut down the cluster on a node by invoking the shutdown procedure of the node's cluster monitor, this procedure is intended for terminating cluster components after stopping any applications on the node that have access to shared storage. VxVM supports clean node shutdown, which allows a node to leave the cluster gracefully when all access to shared volumes has ceased. The host is still operational, but cluster applications cannot be run on it.

The CVM functionality of VxVM maintains global state information for each volume. This enables VxVM to determine which volumes need to be recovered when a node crashes. When a node leaves the cluster due to a crash or by some other means that is not clean, VxVM determines which volumes may have writes that have not completed and the master node resynchronizes these volumes. It can use dirty region logging (DRL) or FastResync if these are active for any of the volumes.

Clean node shutdown must be used after, or in conjunction with, a procedure to halt all cluster applications. Depending on the characteristics of the clustered application and its shutdown procedure, a successful shutdown can require a lot of time (minutes to hours). For instance, many applications have the concept of draining, where they accept no new work, but complete any work in progress before exiting. This process can take a long time if, for example, a long-running transaction is active.

When the VxVM shutdown procedure is invoked, it checks all volumes in all shared disk groups on the node that is being shut down. The procedure then either continues with the shutdown, or fails for one of the following reasons:

- If all volumes in shared disk groups are closed, VxVM makes them unavailable to applications. Because all nodes are informed that these volumes are closed on the leaving node, no resynchronization is performed.
- If any volume in a shared disk group is open, the shutdown procedure fails. The shutdown procedure can be repeatedly retried until it succeeds. There is no timeout checking in this operation—it is intended as a service that verifies that the clustered applications are no longer active.

Once shutdown succeeds, the node has left the cluster. It is not possible to access the shared volumes until the node joins the cluster again.

Since shutdown can be a lengthy process, other reconfiguration can take place while shutdown is in progress. Normally, the shutdown attempt is suspended until the other reconfiguration completes. However, if it is already too far advanced, the shutdown may complete first.

Cluster shutdown

If all nodes leave a cluster, shared volumes must be recovered when the cluster is next started if the last node did not leave cleanly, or if resynchronization from previous nodes leaving uncleanly is incomplete. CVM automatically handles the recovery and resynchronization tasks when a node joins the cluster.

Dirty region logging in cluster environments

Dirty region logging (DRL) is an optional property of a volume that provides speedy recovery of mirrored volumes after a system failure. DRL is supported in cluster-shareable disk groups. This section provides a brief overview of how DRL behaves in a cluster environment.

In a cluster environment, the VxVM implementation of DRL differs slightly from the normal implementation.

A dirty region log on a system without cluster support has a recovery map and a single active map. A CVM DRL, however, has a single recovery map per cluster and one active map per cluster node.

The dirty region log size in clusters is typically larger than in non-clustered systems, as it must accommodate a recovery map plus active maps for each node in the cluster. The size of each map within the dirty region log is one or more whole blocks. The `vxassist` command automatically allocates a sufficiently large dirty region log for the size of the volume and the number of nodes.

It is possible to reimport a non-shared disk group (and its volumes) as a shared disk group in a cluster environment. However, the dirty region logs of the imported disk group may be considered invalid and a full recovery may result.

If a shared disk group is imported as a private disk group on a system without cluster support, VxVM considers the logs of the shared volumes to be invalid and conducts a full volume recovery. After the recovery completes, VxVM uses DRL.

The cluster functionality of VxVM can perform a DRL recovery on a non-shared volume. However, if such a volume is moved to a VxVM system with cluster support and imported as shared, the dirty region log is probably too small to accommodate maps for all the cluster nodes. VxVM then marks the log invalid and performs a full recovery anyway. Similarly, moving a DRL volume from a two-node cluster to a four-node cluster can result in too small a log size, which the cluster functionality of VxVM handles with a full volume recovery. In both cases, you must allocate a new log of sufficient size.

See "Dirty region logging" on page 56.

How DRL works in a cluster environment

When one or more nodes in a cluster crash, DRL must handle the recovery of all volumes that were in use by those nodes when the crashes occurred. On initial cluster startup, all active maps are incorporated into the recovery map during the volume start operation.

Nodes that crash (that is, leave the cluster as dirty) are not allowed to rejoin the cluster until their DRL active maps have been incorporated into the recovery maps on all affected volumes. The recovery utilities compare a crashed node's active maps with the recovery map and make any necessary updates. Only then can the node rejoin the cluster and resume I/O to the volume (which overwrites the active map). During this time, other nodes can continue to perform I/O.

VxVM tracks which nodes have crashed. If multiple node recoveries are underway in a cluster at a given time. VxVM tracks changes in the state of DRL recovery and prevents I/O collisions.

The master node performs volatile tracking of DRL recovery map updates for each volume, and prevents multiple utilities from changing the recovery map simultaneously.

Administering VxVM in cluster environments

The following sections describe the administration of VxVM's cluster functionality.

Requesting node status and discovering the master node

The `vxdctl` utility controls the operation of the `vxconfigd` volume configuration daemon. The `-c` option can be used to request cluster information and to find out which node is the master. To determine whether the `vxconfigd` daemon is enabled and/or running, use the following command:

```
vxdctl -c mode
```

Table 13-6 shows the various messages that may be output according to the current status of the cluster node.

Table 13-6 Cluster status messages

Status message	Description

`mode: enabled:`
`cluster active - MASTER`
`master: mozart`
	The node is the master.

`mode: enabled:`
`cluster active - SLAVE`
`master: mozart`
	The node is a slave.

`mode: enabled:`
`cluster active - role not set`
`master: mozart`
`state: joining`
`reconfig: master update`
	The node has not yet been assigned a role, and is in the process of joining the cluster.

`mode: enabled:`
`cluster active - SLAVE`
`master: mozart`
`state: joining`
	The node is configured as a slave, and is in the process of joining the cluster.

`mode: enabled:` `cluster inactive`	The cluster is not active on this node.
`mode: booted:` `master: ts4200-04`	Enable root disk encapsulation but not transactions.
`mode: disabled:`	Disable transactions.

If the vxconfigd daemon is disabled, no cluster information is displayed.

See the vxdctl(1M) manual page.

Changing the CVM master manually

You can change the CVM master manually from one node in the cluster to another node, while the cluster is online. CVM migrates the master node, and reconfigures the cluster.

Symantec recommends that you switch the master when the cluster is not handling VxVM configuration changes or cluster reconfiguration operations. In most cases, CVM aborts the operation to change the master, if CVM detects that any configuration changes are occurring in the VxVM or the cluster. After the master change operation starts reconfiguring the cluster, other commands that require configuration changes will fail.

See "Errors during CVM master switching" on page 464.

To change the master online, a cluster protocol version 100 or later is required.

To change the CVM master manually

1 To view the current master, use one of the following commands:

```
# vxclustadm nidmap
Name              CVM Nid     CM Nid     State
system01            0           0          Joined: Slave
system02            1           1          Joined: Master

# vxdctl -c mode
mode: enabled: cluster active - MASTER
master: system02
```

In this example, the CVM master is system02.

2 From any node on the cluster, run the following command to change the CVM master:

```
# vxclustadm setmaster nodename
```

where *nodename* specifies the name of the new CVM master.

The following example shows changing the master on a cluster from system02 to system01:

```
# vxclustadm setmaster system01
```

3 To monitor the master switching, use the following command:

```
# vxclustadm -v nodestate
 state: cluster member
        nodeId=0
        masterId=0
        neighborId=1
        members[0]=0xf
        joiners[0]=0x0
        leavers[0]=0x0
        members[1]=0x0
        joiners[1]=0x0
        leavers[1]=0x0
        reconfig_seqnum=0x9f9767
        vxfen=off
state: master switching in progress
reconfig: vxconfigd in join
```

In this example, the state indicates that master is being changed.

4 To verify whether the master has successfully changed, use one of the following commands:

```
# vxclustadm nidmap
Name              CVM Nid    CM Nid    State
system01            0          0         Joined: Master
system02            1          1         Joined: Slave
```

```
# vxdctl -c mode
mode: enabled: cluster active - MASTER
master: system01
```

Errors during CVM master switching

Symantec recommends that you switch the master when the cluster is not handling VxVM or cluster configuration changes.

In most cases, CVM aborts the operation to change the master, if CVM detects any configuration changes in progress. CVM logs the reason for the failure into the system logs. In some cases, the failure is displayed in the `vxclustadm setmaster` output as follows:

```
# vxclustadm setmaster system01
VxVM vxclustadm ERROR V-5-1-0 Master switching, a reconfiguration or
```

```
a transaction is in progress.
Try again
```

In some cases, if the master switching operation is interrupted by another reconfiguration operation, the master change fails. In this case, the existing master remains the master of the cluster. After the reconfiguration is complete, reissue the `vxclustadm setmaster` command to change the master.

If the master switching operation has started the reconfiguration, any command that initiates a configuration change fails with the following error:

```
Node processing a master-switch request. Retry operation.
```

If you see this message, retry the command after the master switching has completed.

Determining if a LUN is in a shareable disk group

The `vxdisk` utility manages VxVM disks. To use the `vxdisk` utility to determine whether a LUN is part of a cluster-shareable disk group, use the following command:

```
# vxdisk list accessname
```

where *accessname* is the disk access name (or device name).

For example, a portion of the output from this command (for the device `sde`) is shown here:

```
Device:     sde
devicetag:  sde
type:       auto
clusterid:  cvm2
disk:       name=shdg01 id=963616090.1034.cvm2
timeout:    30
group:      name=shdg id=963616065.1032.cvm2
flags:      online ready autoconfig shared imported
...
```

Note that the `clusterid` field is set to `cvm2` (the name of the cluster), and the `flags` field includes an entry for `shared`. The `imported` flag is only set if a node is a part of the cluster and the disk group is imported.

Listing shared disk groups

`vxdg` can be used to list information about shared disk groups. To display information for all disk groups, use the following command:

```
# vxdg list
```

Example output from this command is displayed here:

```
NAME        STATE              ID
group2      enabled,shared     774575420.1170.teal
group1      enabled,shared     774222028.1090.teal
```

Shared disk groups are designated with the flag `shared`.

To display information for shared disk groups only, use the following command:

```
# vxdg -s list
```

Example output from this command is as follows:

```
NAME      STATE             ID
group2    enabled,shared    774575420.1170.teal
group1    enabled,shared    774222028.1090.teal
```

To display information about one specific disk group, use the following command:

```
# vxdg list diskgroup
```

The following is example output for the command `vxdg list group1` on the master:

```
Group:     group1
dgid:      774222028.1090.teal
import-id: 32768.1749
flags:     shared
version:   140
alignment: 8192 (bytes)
ssb:       on
local-activation: exclusive-write
cluster-actv-modes: node0=ew node1=off
detach-policy: local
dg-fail-policy: leave
copies:    nconfig=2 nlog=2
config:    seqno=0.1976 permlen=1456 free=1448 templen=6
loglen=220
config disk sdk copy 1 len=1456 state=clean online
```

```
config disk sdk copy 1 len=1456 state=clean online
log disk sdk copy 1 len=220
log disk sdk copy 1 len=220
```

Note that the `flags` field is set to `shared`. The output for the same command when run on a slave is slightly different. The `local-activation` and `cluster-actv-modes` fields display the activation mode for this node and for each node in the cluster respectively. The `detach-policy` and `dg-fail-policy` fields indicate how the cluster behaves in the event of loss of connectivity to the disks, and to the configuration and log copies on the disks.

Creating a shared disk group

You can run the command to create a shared disk group on a master node or a slave node. If you create the disk group on a slave node, the command is shipped to the master and executed on the master.

If the cluster software has been run to set up the cluster, a shared disk group can be created using the following command:

```
# vxdg -s init diskgroup [diskname=]devicenames
```

where *diskgroup* is the disk group name, *diskname* is the administrative name chosen for a VM disk, and *devicename* is the device name (or disk access name).

Warning: The operating system cannot tell if a disk is shared. To protect data integrity when dealing with disks that can be accessed by multiple systems, use the correct designation when adding a disk to a disk group. VxVM allows you to add a disk that is not physically shared to a shared disk group if the node where the disk is accessible is the only node in the cluster. However, this means that other nodes cannot join the cluster. Furthermore, if you attempt to add the same disk to different disk groups (private or shared) on two nodes at the same time, the results are undefined. Perform all configuration on one node only, and preferably on the master node.

Importing disk groups as shared

You can import shared disk groups on a master node or a slave node. If you run the command to import the shared disk group on a slave node, the command is shipped to the master and executed on the master.

Disk groups can be imported as shared using the `vxdg -s import` command. If the disk groups are set up before the cluster software is run, the disk groups can be imported into the cluster arrangement using the following command:

```
# vxdg -s import diskgroup
```

where *diskgroup* is the disk group name or ID. On subsequent cluster restarts, the disk group is automatically imported as shared. Note that it can be necessary to deport the disk group (using the `vxdg deport diskgroup` command) before invoking the `vxdg` utility.

Forcibly importing a disk group

You can use the `-f` option to the `vxdg` command to import a disk group forcibly.

Warning: The force option(`-f`) must be used with caution and only if you are fully aware of the consequences such as possible data corruption.

When a cluster is restarted, VxVM can refuse to auto-import a disk group for one of the following reasons:

- A disk in the disk group is no longer accessible because of hardware errors on the disk. In this case, use the following command to forcibly reimport the disk group:

  ```
  # vxdg -s -f import diskgroup
  ```

  Note: After a forced import, the data on the volumes may not be available and some of the volumes may be in the disabled state.

- Some of the disks in the shared disk group are not accessible, so the disk group cannot access all of its disks. In this case, a forced import is unsafe and must not be attempted because it can result in inconsistent mirrors.

Handling cloned disks in a shared disk group

If a disk is cloned or copied in such a way to created a duplicate disk ID, you must perform special actions to import the disk into VxVM. The procedures are the same for shared disk groups as for private disk groups. When you are ready to import the disk, specify the `-s` option to the `vxdg import` command:

```
# vxdg import -s diskgroup
```

See "Handling cloned disks with duplicated identifiers" on page 244.

Converting a disk group from shared to private

You can convert shared disk groups on a master node or a slave node. If you run the command to convert the shared disk group on a slave node, the command is shipped to the master and executed on the master.

To convert a shared disk group to a private disk group, first deport it on the master node using this command:

```
# vxdg deport diskgroup
```

Then reimport the disk group on any cluster node using this command:

```
# vxdg import diskgroup
```

Moving objects between shared disk groups

You can move objects between shared disk groups on a master node or a slave node. If you run the command to move objects between shared disk groups on a slave node, the command is shipped to the master and executed on the master.

You can use the `vxdg move` command to move a self-contained set of VxVM objects such as disks and top-level volumes between disk groups. In a cluster, you can move such objects between private disk groups on any cluster node where those disk groups are imported.

See "Moving objects between disk groups" on page 269.

Splitting shared disk groups

You can use the `vxdg split` command to remove a self-contained set of VxVM objects from an imported disk group, and move them to a newly created disk group.

See "Splitting disk groups" on page 272.

Splitting a private disk group creates a private disk group, and splitting a shared disk group creates a shared disk group. You can split a private disk group on any cluster node where that disk group is imported.

You can split a shared disk group or create a shared target disk group on a master node or a slave node. If you run the command to split a shared disk group or to create a shared target disk group on a slave node, the command is shipped to the master and executed on the master.

Joining shared disk groups

You cannot join a private disk group and a shared disk group.

You can use the `vxdg join` command to merge the contents of two imported disk groups. In a cluster, you can join two private disk groups on any cluster node where those disk groups are imported.

If the source disk group and the target disk group are both shared, you can perform the join from a master node or a slave node. If you run the command to perform the join on a slave node, the command is shipped to the master and executed on the master.

See "Joining disk groups" on page 273.

Changing the activation mode on a shared disk group

The activation mode for access by a cluster node to a shared disk group is set directly on that node.

The activation mode of a shared disk group can be changed using the following command:

```
# vxdg -g diskgroup set activation=mode
```

The activation mode is one of `exclusivewrite` or `ew`, `readonly` or `ro`, `sharedread` or `sr`, `sharedwrite` or `sw`, or `off`.

If you use this command to change the activation mode of a shared disk group, you must first change the activation mode to `off` before setting it to any other value, as shown here:

```
# vxdg -g myshdg set activation=off
# vxdg -g myshdg set activation=readonly
```

See "Activation modes of shared disk groups" on page 441.

Setting the disk detach policy on a shared disk group

The `vxdg` command may be used to set either the `global` or `local` disk detach policy for a shared disk group:

```
# vxdg -g diskgroup set diskdetpolicy=global|local
```

The default disk detach policy is `global`.

Setting the disk group failure policy on a shared disk group

The `vxdg` command may be used to set either the `dgdisable` or `leave` failure policy for a shared disk group:

```
# vxdg -g diskgroup set dgfailpolicy=dgdisable|leave|requestleave
```

The default failure policy is `dgdisable`.

Creating volumes with exclusive open access by a node

When using the `vxassist` command to create a volume, you can use the `exclusive=on` attribute to specify that the volume may only be opened by one node in the cluster at a time. For example, to create the mirrored volume `volmir` in the disk group `dskgrp`, and configure it for exclusive open, use the following command:

```
# vxassist -g dskgrp make volmir 5g layout=mirror exclusive=on
```

Multiple opens by the same node are also supported. Any attempts by other nodes to open the volume fail until the final close of the volume by the node that opened it.

Specifying `exclusive=off` instead means that more than one node in a cluster can open a volume simultaneously. This is the default behavior.

Setting exclusive open access to a volume by a node

Exclusive open access on a volume can be set from the any node in the cluster. Ensure that none of the nodes in the cluster have the volume open when setting this attribute.

You can set the `exclusive=on` attribute with the vxvol command to specify that an existing volume may only be opened by one node in the cluster at a time.

For example, to set exclusive open on the volume `volmir` in the disk group `dskgrp`, use the following command:

```
# vxvol -g dskgrp set exclusive=on volmir
```

Multiple opens by the same node are also supported. Any attempts by other nodes to open the volume fail until the final close of the volume by the node that opened it.

Specifying `exclusive=off` instead means that more than one node in a cluster can open a volume simultaneously. This is the default behavior.

Displaying the cluster protocol version

The following command displays the cluster protocol version running on a node:

```
# vxdctl list
```

This command produces output similar to the following:

```
Volboot file
version: 3/1
seqno: 0.19
cluster protocol version: 100
hostid: giga
entries:
```

You can also check the existing cluster protocol version using the following command:

```
# vxdctl protocolversion
```

This produces output similar to the following:

```
Cluster running at protocol 100
```

Displaying the supported cluster protocol version range

The following command displays the maximum and minimum protocol version supported by the node and the current protocol version:

```
# vxdctl support
```

This command produces output similar to the following:

```
Support information:
vxconfigd_vrsn:     31
dg_minimum:         20
dg_maximum:         160
kernel:             31
protocol_minimum:   90
protocol_maximum:  100
protocol_current:  100
```

You can also use the following command to display the maximum and minimum cluster protocol version supported by the current Veritas Volume Manager release:

```
# vxdctl protocolrange
```

This produces output similar to the following:

```
minprotoversion: 90, maxprotoversion: 100
```

Recovering volumes in shared disk groups

The `vxrecover` utility is used to recover plexes and volumes after disk replacement. When a node leaves a cluster, it can leave some mirrors in an inconsistent state. The `vxrecover` utility can be used to recover such volumes. The `-c` option to `vxrecover` causes it to recover all volumes in shared disk groups. The `vxconfigd` daemon automatically calls the `vxrecover` utility with the `-c` option when necessary.

Warning: While the `vxrecover` utility is active, there can be some degradation in system performance.

Obtaining cluster performance statistics

The `vxstat` utility returns statistics for specified objects. In a cluster environment, `vxstat` gathers statistics from all of the nodes in the cluster. The statistics give the total usage, by all nodes, for the requested objects. If a local object is specified, its local usage is returned.

You can optionally specify a subset of nodes using the following form of the command:

```
# vxstat -g diskgroup -n node[,node...]
```

where *node* is the CVM node ID number. You can find out the CVM node ID by using the following command:

```
# vxclustadm nidmap
```

If a comma-separated list of nodes is supplied, the `vxstat` utility displays the sum of the statistics for the nodes in the list.

For example, to obtain statistics for node 2, volume `vol1`,use the following command:

```
# vxstat -g diskgroup -n 2 vol1
```

This command produces output similar to the following:

```
            OPERATIONS     BLOCKS           AVG TIME(ms)
TYP   NAME   READ    WRITE   READ      WRITE   READ    WRITE
vol   vol1   2421    0       600000    0       99.0    0.0
```

To obtain and display statistics for the entire cluster, use the following command:

```
# vxstat -b
```

The statistics for all nodes are summed. For example, if node 1 performed 100 I/O operations and node 2 performed 200 I/O operations, `vxstat -b` displays a total of 300 I/O operations.

Administering CVM from the slave node

CVM requires that the master node of the cluster executes configuration commands, which change the object configuration of a CVM shared disk group. Examples of configuration changes include creating shared disk groups, importing shared disk groups, deporting shared disk groups, and creating volumes or snapshots in a shared disk group.

Starting in this release, you can issue most configuration commands that operate on the shared disk group from any node in the cluster. If you issue the command on the slave node, CVM ships the commands from the slave node to the master node. CVM then executes the command on the master node. In normal conditions, we recommend that you issue configuration-changing commands for a shared disk group on the master node. If the circumstances require, you can issue these commands from the slave node.

Commands that operate on private disk groups are not shipped to the master node. Similarly, CVM does not ship commands that operate locally on the slave node, such as `vxprint` and `vxdisk list`.

When you issue a command on the slave that is executed on the master, the command output (on the slave node) displays the object names corresponding to the master node. For example, the command displays the disk access name (`daname`) from the master node.

When run from a slave node, a query command such as `vxtask` or `vxstat` displays the status of the commands on the slave node. The command does not show the status of commands that originated from the slave node and that are executing on the master node.

Note the following error handling for commands that you originate from the slave node, which CVM executes on the master:

- If the `vxconfigd` daemon on either the slave node or on the master node fails, the command exits. The instance of the command on the master also exits. To determine if the command executed successfully, use the `vxprint` command to check the status of the VxVM objects.
- If the slave node that shipped the command or the master node leaves the cluster while the master is executing the command, the command exits on the master node as well as on the slave node. To determine if the command executed successfully, use the`vxprint` command to check the status of the VxVM objects.

Note the following limitations for issuing CVM commands from the slave node:

- The CVM protocol version 100 or later is required on all nodes in the cluster.
  See "Displaying the cluster protocol version" on page 471.
- CVM uses the values in the defaults file on the master node when CVM executes the command. To avoid any ambiguity, we recommend that you use the same values in the defaults file for each of the nodes in the cluster.
- CVM does not support executing all commands on the slave node. You must issue the following commands only on the master node:
  - Commands that specify a controller name. For example:

    ```
    # vxassist -g shareddg make sharedvol 20M ctlr:fscsi0
    ```

  - Commands that specify both a shared disk group and a private disk group. For example:

    ```
    # vxdg destroy privatedg shareddg
    ```

  - Commands that include the defaults file as an argument. For example:

    ```
    # vxassist -d defaults_file
    ```

  - Veritas Volume Replicator (VVR) commands including `vxibc`, `vxrlink`, `vxrsync`, `vxrvg`, `vrport`, `vrstat`, and `vradmin`.
  - The `vxdisk` command options that act on shared disk groups.

  See "CVM commands supported for executing on the slave node" on page 548.

Chapter 14

Administering sites and remote mirrors

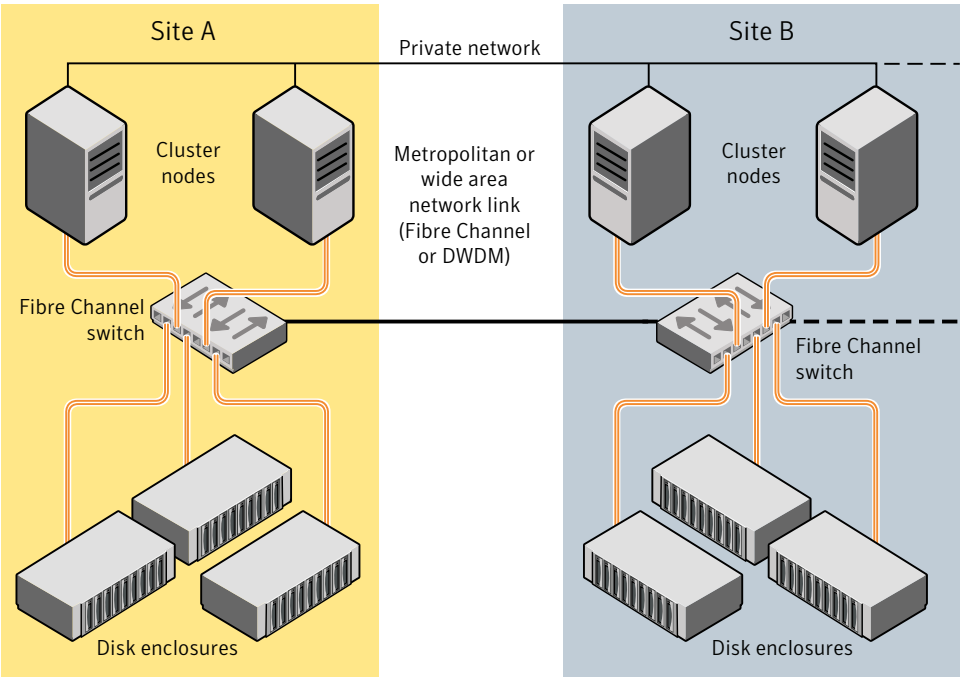
This chapter includes the following topics:

About sites and remote mirrors

In a Remote Mirror configuration (also known as a campus cluster or stretch cluster) the hosts and storage of a cluster that would usually be located in one place, are instead divided between two or more sites. These sites are typically connected via a redundant high-capacity network that provides access to storage and private link communication between the cluster nodes.

Figure 14-1 shows a typical two-site remote mirror configuration.

Figure 14-1 Example of a two-site remote mirror configuration



If a disk group is configured across the storage at the sites, and inter-site communication is disrupted, there is a possibility of a serial split brain condition arising if each site continues to update the local disk group configuration copies.

See "Handling conflicting configuration copies" on page 255.

VxVM provides mechanisms for dealing with the serial split brain condition, monitoring the health of a remote mirror, and testing the robustness of the cluster against various types of failure (also known as fire drill).

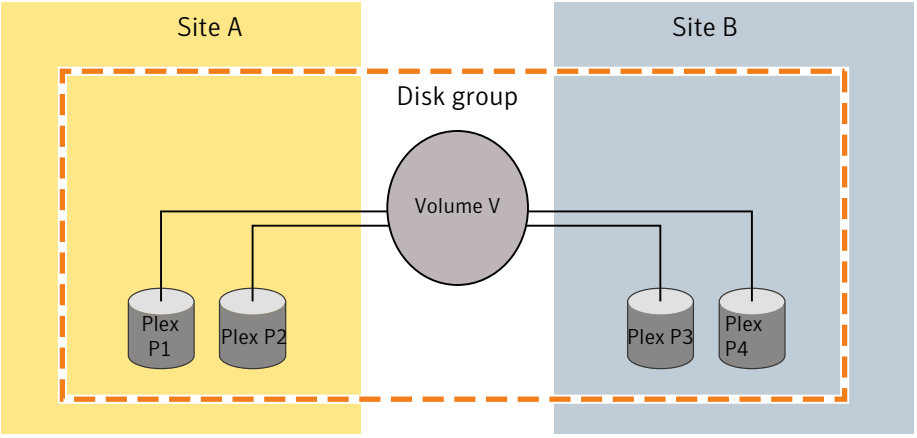
For applications and services to function correctly at a site when other sites have become inaccessible, at least one complete plex of each volume must be configured at each site (site-based allocation), and the consistency of the data in the plexes at each site must be ensured (site consistency).

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.

As shown in the examples, the network connectivity can be Fibre Channel (FC) or Dense Wavelength Division Multiplex (DWDM). The storage network and the heartbeat network can be the same network.

Figure 14-2 shows an example of a site-consistent volume with two plexes configured at each of two sites.

Figure 14-2 Site-consistent volume with two plexes at each of two sites



The storage for plexes P1 and P2 is allocated storage that is tagged as belonging to site A, and the storage for plexes P3 and P4 is allocated storage that is tagged as belonging to site B.

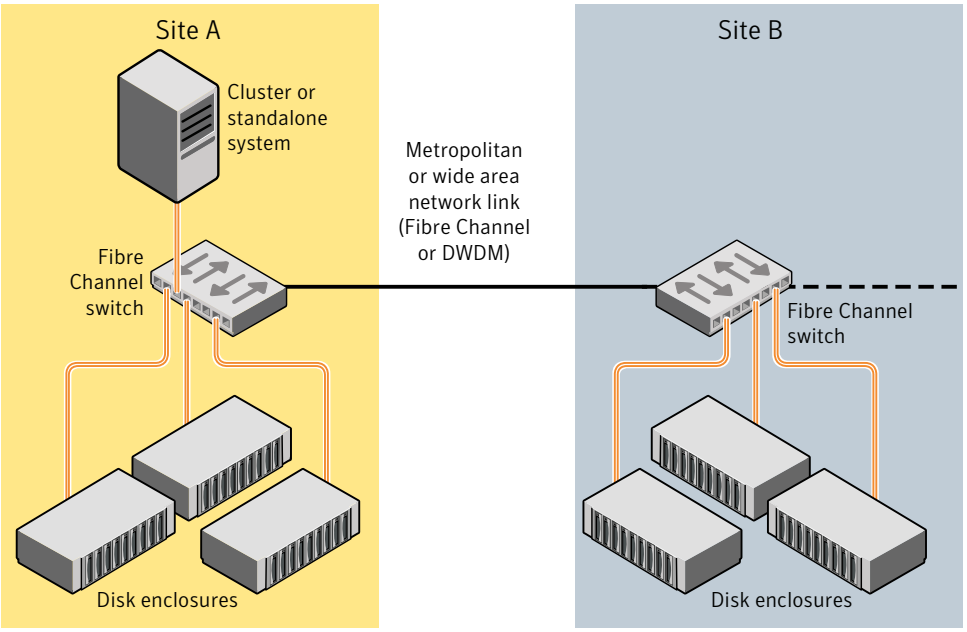
Although not shown in this figure, DCO log volumes are also mirrored across the sites, and disk group configuration copies are distributed across the sites.

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 14-3 shows a configuration with remote storage only that is also supported.

Figure 14-3 Example of a two-site configuration with remote storage only



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 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`.

Note: By default, volumes created will be mirrored when sites are configured in a disk group. Initial synchronization occurs between mirrors. Depending on the size of the volume, synchronization may take a long time. If you do not need to perform an initial synchronization across mirrors, use `init=active` with the `vxassist` command.

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 site consistency 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 489.

Before setting site consistency on a disk group, be sure to meet the following requirements:

- A license enabling the Site Awareness feature must be purchased for each host 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 483.
- 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 the site to which 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.

See "Configuring automatic site tagging for a disk group" on page 488.

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. 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 available 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 382.

Making an existing disk group site consistent

The site consistency feature requires that a license enabling the site awareness feature has been purchased for 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.

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

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 `DCOv20` if it is not attached already. `DCOv20` is required to ensure that site detach and reattach are instantaneous.

See "Preparing a volume for DRL and instant snapshots" on page 371.

```
# 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 purchased for 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 483.

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 in a production environment.

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 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 487.

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
   ```

3. Verify the new site name:

   ```
   # vxdctl list
   ```

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 -o tag=site 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 20 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 14-1 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 14-1 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 be equal to or more than 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.

Table 14-1 Examples of storage allocation by specifying sites *(continued)*

Command	Description
`# 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`.
`# 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 -o tag=site listtag diskgroup
```

Failure and recovery scenarios

Table 14-2 lists the possible failure scenarios and recovery procedures for the Remote Mirror feature.

Table 14-2 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 493.
Failure of hosts at a site.	See "Recovering from host failure" on page 494.
Failure of storage at a site.	See "Recovering from storage failure" on page 494.
Failure of both hosts and storage at a site.	See "Recovering from site failure" on page 495.

Recovering from a loss of site connectivity

Warning: To avoid a potential loss of data, it is recommended that you configure Veritas Cluster Server to handle network split-brain.

If the network links between the sites are disrupted, the application environments may continue to run in parallel, and this may lead to inconsistencies between the disk group configuration copies at the sites. If the parallel instances of an application issue writes to volumes, an unrecoverable data loss may occur and manual intervention is needed. To avoid data loss, it is recommended that you configure the VCS fencing mechanism to handle network split-brain situations. When connectivity between the sites is restored, a serial split-brain condition will be detected between the sites. One site must be chosen as having the preferred version of the data and the disk group configuration copies. The data from the chosen site is resynchronized to the other site. If new writes are issued to volumes after the network split, they are overwritten with the data from the chosen site. The configuration copies at the other sites are updated from the copies at the chosen site.

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 255.

Recovering from host failure

If one or more cluster nodes fail at a site, but the storage remains online, this is handled either by VCS failover in the case of the Storage Foundation HA product, or by node takeover in the case that the node was the master for a shared disk group as supported by the Storage Foundation Cluster File System software.

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 495.

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 Volume Manager 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 [-o overridessb] reattachsite sitename
# vxrecover -g diskgroup
```

The `-o overridessb` option is only required if a serial split-brain condition is indicated. A serial split-brain condition may happen if the site was brought back up while the private network link was inoperative. This option updates the configuration database on the reattached site with the consistent copies at the other sites.

See "Handling conflicting configuration copies" on page 255.

For more information about recovering a disk group, refer to the *Veritas Volume Manager Troubleshooting Guide*.

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 `/etc/init.d/vxvm-recover` startup script, and reboot the system.

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 `/etc/init.d/vxvm-recover` startup script.

Chapter 15

Performance monitoring and tuning

This chapter includes the following topics:

Performance guidelines

Veritas Volume Manager (VxVM) can improve system performance by optimizing the layout of data storage on the available hardware. VxVM lets you optimize data storage performance using the following strategies:

- Balance the I/O load among the available disk drives.
- Use striping and mirroring to increase I/O bandwidth to the most frequently accessed data.

VxVM also provides data redundancy through mirroring and RAID-5, which allows continuous access to data in the event of disk failure.

Data assignment

When you decide where to locate file systems, you typically try to balance I/O load among the available disk drives. The effectiveness of this approach is limited. It is difficult to predict future usage patterns, and you cannot split file systems across the drives. For example, if a single file system receives the most disk accesses, moving the file system to another drive also moves the bottleneck.

VxVM can split volumes across multiple drives. This approach gives you a finer level of granularity when you locate data. After you measure access patterns, you can adjust your decisions on where to place file systems. You can reconfigure volumes online without adversely impacting their availability.

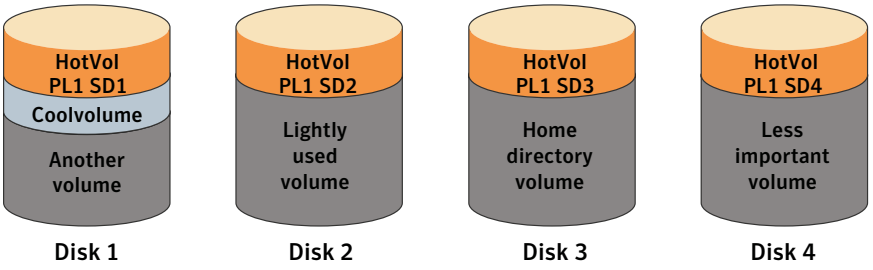
Striping

Striping improves access performance by cutting data into slices and storing it on multiple devices that can be accessed in parallel. Striped plexes improve access performance for both read and write operations.

After you identify the most heavily-accessed volumes (containing file systems or databases), you can increase access bandwidth to this data by striping it across portions of multiple disks.

Figure 15-1 shows an example of a single volume (`HotVol`) that has been identified as a data-access bottleneck.

Figure 15-1 Use of striping for optimal data access



This volume is striped across four disks. The remaining space on these disks is free for use by less-heavily used volumes.

Mirroring

Mirroring stores multiple copies of data on a system. When you apply mirroring properly, data is continuously available. Mirroring also protects against data loss due to physical media failure. If the system crashes or a disk or other hardware fails, mirroring improves the chance of data recovery.

In some cases, you can also use mirroring to improve I/O performance. Unlike striping, the performance gain depends on the ratio of reads to writes in the disk accesses. If the system workload is primarily write-intensive (for example, greater than 30 percent writes), mirroring can reduce performance.

Combining mirroring and striping

When you have multiple I/O streams, you can use mirroring and striping together to significantly improve performance.

Because parallel I/O streams can operate concurrently on separate devices, striping provides better throughput. When I/O fits exactly across all stripe units in one stripe, serial access is optimized.

Because mirroring is generally used to protect against loss of data due to disk failures, it is often applied to write-intensive workloads. This approach degrades throughput. In those cases, you can combine mirroring with striping to deliver high availability and increased throughput.

You can create a mirrored-stripe volume. Stripe half of the available disks to form one striped data plex, and stripe the remaining disks to form the other striped data plex in the mirror. This approach is often the best way to configure a set of disks for optimal performance with reasonable reliability. However, if a disk in one of the plexes fails, the entire plex is unavailable.

You can also arrange equal numbers of disks into separate mirror volumes. Afterwards, create a striped plex across these mirror volumes to form a striped-mirror volume.

See "Mirroring plus striping (striped-mirror, RAID-1+0 or RAID-10)" on page 42.

If a disk in a mirror fails, it does not take the disks in the other mirrors out of use. For large volumes or large numbers of disks, a striped-mirror layout is preferred over a mirrored-stripe layout.

RAID-5

RAID-5 offers many of the advantages of combined mirroring and striping, but it requires more disk space. RAID-5 read performance is similar to that of striping, and RAID-5 parity offers redundancy similar to mirroring. The disadvantages of RAID-5 include relatively slow write performance.

RAID-5 is not usually seen as a way to improve throughput performance. The exception is when the access patterns of applications show a high ratio of reads to writes. .

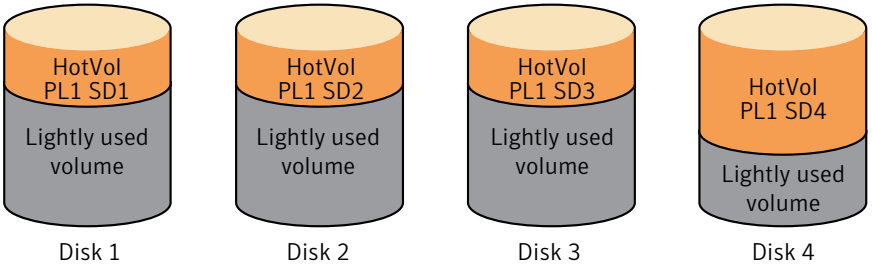
Volume read policies

To help optimize performance for different types of volumes, VxVM lets you set one of several read policies on data plexes.

See "Changing the read policy for mirrored volumes" on page 382.

Figure 15-2 shows an example in which the read policy of the mirrored-stripe volume labeled `HotVol` is set to `prefer` for the striped plex `PL1`.

Figure 15-2 Use of mirroring and striping for improved performance



The `prefer` policy distributes the load when reading across the otherwise lightly-used disks in `PL1`, as opposed to the single disk in plex `PL2`. (`HotVol` is an example of a mirrored-stripe volume in which one data plex is striped and the other data plex is concatenated.)

To improve performance for read-intensive workloads, you can attach up to 32 data plexes to the same volume. However, this approach is usually an ineffective use of disk space for the gain in read performance.

Performance monitoring

As a system administrator, you have two sets of priorities for setting priorities for performance. One set is physical, concerned with hardware such as disks and controllers. The other set is logical, concerned with managing software and its operation.

Setting performance priorities

The important physical performance characteristics of disk hardware are the relative amounts of I/O on each drive, and the concentration of the I/O within a drive to minimize seek time. Based on monitored results, you can then move the location of subdisks to balance I/O activity across the disks.

The logical priorities involve software operations and how they are managed. Based on monitoring, you may choose to change the layout of certain volumes to improve their performance. You might even choose to reduce overall throughput to improve the performance of certain critical volumes. Only you can decide what is important on your system and what trade-offs you need to make.

Best performance is usually achieved by striping and mirroring all volumes across a reasonable number of disks and mirroring between controllers, when possible. This procedure tends to even out the load between all disks, but it can make VxVM more difficult to administer. For large numbers of disks (hundreds or thousands), set up disk groups containing 10 disks, where each group is used to create a striped-mirror volume. This technique provides good performance while easing the task of administration.

Obtaining performance data

VxVM provides two types of performance information: I/O statistics and I/O traces. Each of these can help in performance monitoring. You can obtain I/O statistics using the `vxstat` command, and I/O traces using the `vxtrace` command. A brief discussion of each of these utilities may be found in the following sections.

Tracing volume operations

Use the `vxtrace` command to trace operations on specified volumes, kernel I/O object types or devices. The `vxtrace` command either prints kernel I/O errors or I/O trace records to the standard output or writes the records to a file in binary format. For I/O related to reclaim operations, the I/O trace records indicate that it is a reclaim I/O. Binary trace records written to a file can also be read back and formatted by `vxtrace`.

If you do not specify any operands, `vxtrace` reports either all error trace data or all I/O trace data on all virtual disk devices. With error trace data, you can select all accumulated error trace data, wait for new error trace data, or both of these (this is the default action). Selection can be limited to a specific disk group, to specific VxVM kernel I/O object types, or to particular named objects or devices.

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

Printing volume statistics

Use the `vxstat` command to access information about activity on volumes, plexes, subdisks, and disks under VxVM control, and to print summary statistics to the standard output. These statistics represent VxVM activity from the time the system initially booted or from the last time the counters were reset to zero. If no VxVM object name is specified, statistics from all volumes in the configuration database are reported.

VxVM records the following I/O statistics:

- count of operations
- number of blocks transferred (one operation can involve more than one block)

- average operation time (which reflects the total time through the VxVM interface and is not suitable for comparison against other statistics programs)

These statistics are recorded for logical I/O including reads, writes, atomic copies, verified reads, verified writes, plex reads, and plex writes for each volume. As a result, one write to a two-plex volume results in at least five operations: one for each plex, one for each subdisk, and one for the volume. Also, one read that spans two subdisks shows at least four reads—one read for each subdisk, one for the plex, and one for the volume.

VxVM also maintains other statistical data. For each plex, it records read and write failures. For volumes, it records corrected read and write failures in addition to read and write failures.

To reset the statistics information to zero, use the `-r` option. This can be done for all objects or for only those objects that are specified. Resetting just prior to an operation makes it possible to measure the impact of that particular operation.

The following is an example of output produced using the `vxstat` command:

```
                OPERATIONS          BLOCKS             AVG TIME(ms)
TYP  NAME         READ    WRITE      READ     WRITE     READ      WRITE
vol  blop            0        0         0         0      0.0        0.0
vol  foobarvol       0        0         0         0      0.0        0.0
vol  rootvol     73017   181735    718528   1114227     26.8       27.9
vol  swapvol     13197    20252    105569    162009     25.8      397.0
vol  testvol         0        0         0         0      0.0        0.0
```

Additional volume statistics are available for RAID-5 configurations.

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

Using performance data

When you have gathered performance data, you can use it to determine how to configure your system to use resources most effectively. The following sections provide an overview of how you can use this data.

Using I/O statistics

Examination of the I/O statistics can suggest how to reconfigure your system. You should examine two primary statistics: volume I/O activity and disk I/O activity.

Before obtaining statistics, reset the counters for all existing statistics using the `vxstat -r` command. This eliminates any differences between volumes or disks

due to volumes being created, and also removes statistics from boot time (which are not usually of interest).

After resetting the counters, allow the system to run during typical system activity. Run the application or workload of interest on the system to measure its effect. When monitoring a system that is used for multiple purposes, try not to exercise any one application more than usual. When monitoring a time-sharing system with many users, let statistics accumulate for several hours during the normal working day.

To display volume statistics, enter the `vxstat` command with no arguments. The following is a typical display of volume statistics:

```
                  OPERATIONS          BLOCKS         AVG TIME(ms)
TYP  NAME      READ    WRITE      READ     WRITE     READ    WRITE
vol  archive    865      807      5722      3809     0.32     0.24
vol  home      2980     5287      6504     10550     0.37     2.21
vol  local    49477    49230    507892    204975     0.28     0.33
vol  rootvol 102906   342664   1085520   1962946     0.28     0.25
vol  src       79174    23603    425472    139302     0.22     0.30
vol  swapvol   22751    32364    182001    258905     0.25     3.23
```

Such output helps to identify volumes with an unusually large number of operations or excessive read or write times.

To display disk statistics, use the `vxstat -d` command. The following is a typical display of disk statistics:

```
                  OPERATIONS          BLOCKS         AVG TIME(ms)
TYP  NAME      READ    WRITE      READ     WRITE     READ    WRITE
dm   mydg01   40473   174045    455898    951379     0.29     0.35
dm   mydg02   32668    16873    470337    351351     0.35     1.02
dm   mydg03   55249    60043    780779    731979     0.35     0.61
dm   mydg04   11909    13745    114508    128605     0.25     0.30
```

If you need to move the volume named `archive` onto another disk, use the following command to identify on which disks it lies:

```
# vxprint -g mydg -tvh archive
```

The following is an extract from typical output:

```
V   NAME        RVG/VSET/CO  KSTATE     STATE     LENGTH      READPOL      REFPLEX      UTYPE
PL  NAME        VOLUME       KSTATE     STATE     LENGTH      LAYOUT       NCOL/WDTH    MODE
SD  NAME        PLEX         DISK       DISKOFFS  LENGTH      [COL/]OFF    DEVICE       MODE
```

```
v   archive    -            ENABLED   ACTIVE    20480   SELECT    -        fsgen
pl  archive-01 archive      ENABLED   ACTIVE    20480   CONCAT    -        RW
sd  mydg03-03  archive-01   mydg03    0         40960   0         sdc      ENA
```

The subdisks line (beginning `sd`) indicates that the volume `archive` is on disk `mydg03`. To move the volume off `mydg03`, use the following command.

Note: The ! character is a special character in some shells. This example shows how to escape it in a bash shell.

```
# vxassist -g mydg move archive \!mydg03 dest_disk
```

Here *dest_disk* is the destination disk to which you want to move the volume. It is not necessary to specify a destination disk. If you do not specify a destination disk, the volume is moved to an available disk with enough space to contain the volume.

For example, to move a volume from disk `mydg03` to disk `mydg04`, in the disk group, `mydg`, use the following command:

```
# vxassist -g mydg move archive \!mydg03 mydg04
```

This command indicates that the volume is to be reorganized so that no part of it remains on `mydg03`.

If two volumes (other than the root volume) on the same disk are busy, move them so that each is on a different disk.

If one volume is particularly busy (especially if it has unusually large average read or write times), stripe the volume (or split the volume into multiple pieces, with each piece on a different disk). If done online, converting a volume to use striping requires sufficient free space to store an extra copy of the volume. If sufficient free space is not available, a backup copy can be made instead. To convert a volume, create a striped plex as a mirror of the volume and then remove the old plex. For example, the following commands stripe the volume `archive` across disks `mydg02`, `mydg03`, and `mydg04` in the disk group, `mydg`, and then remove the original plex `archive-01`:

```
# vxassist -g mydg mirror archive layout=stripe mydg02 mydg03 \
  mydg04
# vxplex -g mydg -o rm dis archive-01
```

After reorganizing any particularly busy volumes, check the disk statistics. If some volumes have been reorganized, clear statistics first and then accumulate statistics for a reasonable period of time.

If some disks appear to be excessively busy (or have particularly long read or write times), you may want to reconfigure some volumes. If there are two relatively busy volumes on a disk, move them closer together to reduce seek times on the disk. If there are too many relatively busy volumes on one disk, move them to a disk that is less busy.

Use I/O tracing (or subdisk statistics) to determine whether volumes have excessive activity in particular regions of the volume. If the active regions can be identified, split the subdisks in the volume and move those regions to a less busy disk.

Warning: Striping a volume, or splitting a volume across multiple disks, increases the chance that a disk failure results in failure of that volume. For example, if five volumes are striped across the same five disks, then failure of any one of the five disks requires that all five volumes be restored from a backup. If each volume were on a separate disk, only one volume would need to be restored. Use mirroring or RAID-5 to reduce the chance that a single disk failure results in failure of a large number of volumes.

Note that file systems and databases typically shift their use of allocated space over time, so this position-specific information on a volume is often not useful. Databases are reasonable candidates for moving to non-busy disks if the space used by a particularly busy index or table can be identified.

Examining the ratio of reads to writes helps to identify volumes that can be mirrored to improve their performance. If the read-to-write ratio is high, mirroring can increase performance as well as reliability. The ratio of reads to writes where mirroring can improve performance depends greatly on the disks, the disk controller, whether multiple controllers can be used, and the speed of the system bus. If a particularly busy volume has a high ratio of reads to writes, it is likely that mirroring can significantly improve performance of that volume.

Using I/O tracing

I/O statistics provide the data for basic performance analysis; I/O traces serve for more detailed analysis. With an I/O trace, focus is narrowed to obtain an event trace for a specific workload. This helps to explicitly identify the location and size of a hot spot, as well as which application is causing it.

Using data from I/O traces, real work loads on disks can be simulated and the results traced. By using these statistics, you can anticipate system limitations and plan for additional resources.

See “Gathering and displaying I/O statistics” on page 185.

See “Specifying the I/O policy” on page 194.

Tuning VxVM

This section describes how to adjust the tunable parameters that control the system resources that are used by VxVM. Depending on the system resources that are available, adjustments may be required to the values of some tunable parameters to optimize performance.

General tuning guidelines

VxVM is optimally tuned for most configurations ranging from small systems to larger servers. When you can use tuning to increase performance on larger systems at the expense of a valuable resource (such as memory), VxVM is generally tuned to run on the smallest supported configuration. You must perform any tuning changes with care. Changes may adversely affect overall system performance or may even leave VxVM unusable.

You can tune the VxVM and DMP parameters using the `vxvoltune` and `vxdmptune` utilities.

See "Changing the values of VxVM tunables" on page 507.

Tuning guidelines for large systems

On smaller systems (with fewer than a hundred disk drives), tuning is unnecessary. VxVM can adopt reasonable defaults for all configuration parameters. On larger systems, configurations can require additional control over the tuning of these parameters, both for capacity and performance reasons.

Generally, only a few significant decisions must be made when setting up VxVM on a large system. One is to decide on the size of the disk groups and the number of configuration copies to maintain for each disk group. Another is to choose the size of the private region for all the disks in a disk group.

Larger disk groups have the advantage of providing a larger free-space pool for the `vxassist` command to select from. They also allow for the creation of larger volumes. Smaller disk groups do not require as large a configuration database and so can exist with smaller private regions. Very large disk groups can eventually exhaust the private region size in the disk group. The result is that no more configuration objects can be added to that disk group. At that point, the configuration either has to be split into multiple disk groups, or the private regions have to be enlarged. Each disk in the disk group must be re-initialized. This can involve reconfiguring everything and restoring from backup.

Number of configuration copies for a disk group

Selection of the number of configuration copies for a disk group is based on a trade-off between redundancy and performance. As a general rule, reducing the number of configuration copies in a disk group speeds up initial access of the disk group, initial startup of the `vxconfigd` daemon, and transactions that are performed within the disk group. However, reducing the number of configuration copies also increases the risk of complete loss of the configuration database, which results in the loss of all objects in the database and of all data in the disk group.

The default policy for configuration copies in the disk group is to allocate a configuration copy for each controller identified in the disk group, or for each target that contains multiple addressable disks. This provides a sufficient degree of redundancy, but can lead to a large number of configuration copies under some circumstances. If this is the case, we recommended that you limit the number of configuration copies to a maximum of 4. Distribute the copies across separate controllers or targets to enhance the effectiveness of this redundancy.

To set the number of configuration copies for a new disk group, use the `nconfig` operand with the `vxdg init` command.

See the `vxdg`(1M) manual page for details.

You can also change the number of copies for an existing group by using the `vxedit set` command. For example, to configure five configuration copies for the disk group, `bigdg`, use the following command:

```
# vxedit set nconfig=5 bigdg
```

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

Changing the values of VxVM tunables

To display the current value of a tunable that is used by VxVM, use the following commands with the `/proc` file system:

```
# cat < /proc/sys/vxvm/vxio/vol_maxio
2048
```

To change the value of a VxVM tunable, specify the new value using the `vxvoltune` command:

```
# vxvoltune vxvm_tunable value
```

You must then shut down and reboot the system for the change to take effect. The new value persists across system reboots until it is next changed.

For example, the following command sets the value of `vol_maxio` to 8192:

```
# vxvoltune vol_maxio 8192
```

Warning: The `vxvoltune` utility modifies the tunable values stored in the `/etc/vx/vxvm_tunables` file. We recommend that you use the `vxvoltune` command to change the values stored in this file. Do not edit this file directly.

You can use the `vxvoltune` command to display the current values of a tunable parameter that is set in the `vxvm_tunables` file:

```
# vxvoltune vxvm_tunable
```

DMP tunables are set online (without requiring a reboot) by using the `vxdmpadm` command as shown here:

```
# vxdmpadm settune dmp_tunable=value
```

The values of these tunables can be displayed by using this command:

```
# vxdmpadm gettune [dmp_tunable]
```

Tunable parameters for VxVM

Table 15-1 lists the kernel tunable parameters for VxVM.

Table 15-1 Kernel tunable parameters for 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.

Table 15-1 Kernel tunable parameters for VxVM *(continued)*

Parameter	Description
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.

Table 15-1 Kernel tunable parameters for VxVM *(continued)*

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 4KB. The maximum and minimum permitted values are 1KB and 8KB. **Note:** The value of this tunable does not have any effect on Persistent FastResync.

Table 15-1 Kernel tunable parameters for VxVM *(continued)*

Parameter	Description
vol_max_vol	The maximum number of volumes that can be created on the system. The minimum and maximum permitted values are 1 and the maximum number of minor numbers representable on the system. The default value is 65534.
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 (1MB). 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.
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 ioctl call. The default value is 256. This value should not be changed.
vol_subdisk_num	The maximum number of subdisks that can be attached to a single plex. There is no theoretical limit to this number, but it has been limited to a default value of 4096. This default can be changed, if required.

Table 15-1 Kernel tunable parameters for VxVM *(continued)*

Parameter	Description
`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 3.
`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 1024 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`.
`voliomem_chunk_size`	The granularity of memory chunks used by VxVM when allocating or releasing system memory. A larger granularity reduces CPU overhead due to memory allocation by allowing VxVM to retain hold of a larger amount of memory. The default value is 32KB.

Table 15-1 Kernel tunable parameters for VxVM *(continued)*

Parameter	Description
`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`.
`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 (8KB). If trace data is often being lost due to this buffer size being too small, then this value can be tuned to a more generous amount.

Table 15-1 Kernel tunable parameters for VxVM *(continued)*

Parameter	Description
`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 (128KB).
`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 (64KB). Increasing this buffer can provide for larger traces to be taken without loss for very heavily used volumes. Care should be taken not to increase this value above the value for the `voliot_iobuf_limit` tunable value.
`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.

Table 15-1 Kernel tunable parameters for VxVM *(continued)*

Parameter	Description
`volpagemod_max_memsz`	The amount of memory, measured in kilobytes, that is allocated for caching FastResync and cache object metadata. This default value is 6144KB (6MB). The valid range for this tunable is from 0 to 50% of physical memory. The memory allocated for this cache is exclusively dedicated to it. It is not available for other processes or applications. Setting the value below 512KB fails if cache objects or volumes that have been prepared for instant snapshot operations are present on the system. If you do not use the FastResync or DRL features that are implemented using a version 20 DCO volume, the value can be set to 0. However, if you subsequently decide to enable these features, you can use the `vxtune` command to change the value to a more appropriate one: `# vxtune volpagemod_max_memsz value` The value that should be used for *value* is determined by the region size and the number of volumes for which space-optimized instant snapshots are taken: *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. If there were 10 such volumes, 60MB of paging memory would be required.
`volraid_minpool_size`	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).

Table 15-1 Kernel tunable parameters for VxVM *(continued)*

Parameter	Description
`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.

DMP tunable parameters

Table 15-2 shows the DMP parameters that can be tuned by using the `vxdmpadm settune` command.

Table 15-2 DMP parameters that are tunable

Parameter	Description
`dmp_cache_open`	If this parameter is set to `on`, the first open of a device that is performed by an array support library (ASL) is cached. This caching enhances the performance of device discovery by minimizing the overhead that is caused by subsequent opens by ASLs. If this parameter is set to `off`, caching is not performed. The default value is `on`.
`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.

Table 15-2 DMP parameters that are tunable *(continued)*

Parameter	Description
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_enable_restore	If this parameter is set to on, it enables the path restoration thread to be started. See "Configuring DMP path restoration policies" on page 207. If this parameter is set to off, it disables the path restoration thread. If the path restoration thread is currently running, use the vxdmpadm stop restore command to stop the thread. The default is on. See "Stopping the DMP path restoration thread" on page 208.
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, provided that 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.
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.

Table 15-2 DMP parameters that are tunable *(continued)*

Parameter	Description
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 existed in releases before 5.0. 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. 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.
dmp_lun_retry_timeout	Retry period for handling transient errors. The value is specified in seconds. When all paths to a disk fail, there may be certain paths that have a temporary failure and are likely to be restored soon. The I/Os may be failed to the application layer even though the failures are transient, unless the I/Os are retried. The dmp_lun_retry_timeout tunable provides a mechanism to retry such transient errors. If the tunable is set to a non-zero value, I/Os to a disk with all failed paths are retried until dmp_lun_retry_timeout interval or until the I/O succeeds on one of the path, whichever happens first. The default value of tunable is 0, which means that the paths are probed only once.

Table 15-2 DMP parameters that are tunable *(continued)*

Parameter	Description
dmp_monitor_fabric	Determines whether the Event Source daemon (vxesd) uses the Storage Networking Industry Association (SNIA) HBA API. This API allows DDL to improve the performance of failover by collecting information about the SAN topology and by monitoring fabric events. If this parameter is set to on, DDL uses the SNIA HBA API. (Note that the HBA vendor specific HBA-API library should be available to use this feature.) If this parameter is set to off, the SNIA HBA API is not used. 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_osevent	Determines whether the Event Source daemon (vxesd) monitors operating system events such as reconfiguration operations. If this parameter is set to on, vxesd monitors operations such as attaching operating system devices. If this parameter is set to off, vxesd does not monitor operating system operations. When DMP co-exists with EMC PowerPath, Symantec recommends setting this parameter to off to avoid any issues. The default setting is on, unless EMC PowerPath is installed. If you install DMP on a system that already has PowerPath installed, DMP sets the dmp_monitor_osevent to off.
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 a Storage Foundation product is installed, the default value is off. When Veritas Dynamic Multi-Pathing is installed, the default value is on.

Table 15-2 DMP parameters that are tunable *(continued)*

Parameter	Description
`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 of this parameter is set to 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 194.
`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`.

Table 15-2 DMP parameters that are tunable *(continued)*

Parameter	Description
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_queue_depth	The maximum number of queued I/O requests on a path during I/O throttling. The default value is 32. A value can also be set for paths to individual arrays by using the vxdmpadm command. See "Configuring the I/O throttling mechanism" on page 204.
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. The value of this tunable can also be set using the vxdmpadm start restore command. See "Configuring DMP path restoration policies" on page 207.
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 207.

Table 15-2 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 207.
dmp_retry_count	If an inquiry succeeds on a path, but there is an I/O error, the number of retries to attempt on the path. The default value is 5. A value can also be set for paths to individual arrays by using the vxdmpadm command. See "Configuring the response to I/O failures" on page 202.
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.
dmp_stat_interval	The time interval between gathering DMP statistics. The default and minimum value are 1 second.

Disabling I/O statistics collection

By default, Veritas Volume Manager collects I/O statistics on all objects in the configuration. This helps you tune different parameters that depend upon the environment and workload.

See "Tunable parameters for VxVM" on page 508.

See "DMP tunable parameters" on page 516.

After the tuning is done, you may choose to disable I/O statistics collection because it improves I/O throughput.

To display whether I/O statistics are enabled

- Enter the following command:

```
# vxtune vol_stats_enable
```

If the system displays `1`, I/O statistics collection is enabled. If it displays `0`, I/O statistics collection is disabled.

To disable I/O statistics collection until the next system reboot

- Enter the following command:

```
# vxtune vol_stats_enable 0
```

If you are concerned about high I/O throughput, you may also choose to disable DMP I/O statistics collection.

To disable DMP I/O statistics collection

- Enter the following command:

```
# vxdmpadm iostat stop
```

Enabling I/O statistics collection

By default, Veritas Volume Manager collects I/O statistics on all objects in the configuration. You only need to perform the tasks in this section if you disabled I/O statistics collection.

To display whether I/O statistics are enabled

- Enter the following command:

```
# vxtune vol_stats_enable
```

If the system displays `1`, I/O statistics collection is enabled. If it displays `0`, I/O statistics collection is disabled.

To enable I/O statistics collection

- Enter the following command:

```
# vxtune vol_stats_enable 1
```

To enable DMP I/O statistics collection

- Enter the following command:

```
# vxdmpadm iostat start
```

See the vxdmpadm(1M) manual page.

Appendix A

Using Veritas Volume Manager commands

This appendix includes the following topics:

About Veritas Volume Manager commands

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 "Online manual pages" on page 555.

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 A-1 lists commands for obtaining information about objects in VxVM.
- Table A-2 lists commands for administering disks.
- Table A-3 lists commands for creating and administering disk groups.
- Table A-4 lists commands for creating and administering subdisks.
- Table A-5 lists commands for creating and administering plexes.
- Table A-6 lists commands for creating volumes.
- Table A-7 lists commands for administering volumes.
- Table A-8 lists commands for monitoring and controlling tasks in VxVM.

Table A-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 142. Example: `# vxdisk -g mydg list`

Table A-1 Obtaining information about objects in VxVM *(continued)*

Command	Description
`vxdg list [diskgroup]`	Lists information about disk groups. See "Displaying disk group information" on page 228. Example: `# vxdg list mydg`
`vxdg -s list`	Lists information about shared disk groups. See "Listing shared disk groups" on page 466. 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.

`vxinfo [-g diskgroup] [volume ...]`	Displays information about the accessibility and usability of volumes.
	See the *Veritas Volume Manager Troubleshooting Guide*.
	Example:
	`# vxinfo -g mydg myvol1 \`
	`  myvol2`

`vxprint -hrt [-g diskgroup] [object ...]`	Prints single-line information about objects in VxVM.
	See "Displaying volume information" on page 348.
	Example:
	`# vxprint -g mydg myvol1 \`
	`  myvol2`

Table A-1 Obtaining information about objects in VxVM *(continued)*

Command	Description
`vxprint -st [-g diskgroup] [subdisk ...]`	Displays information about subdisks. See "Displaying subdisk information" on page 285. Example: `# vxprint -st -g mydg`
`vxprint -pt [-g diskgroup] [plex ...]`	Displays information about plexes. See "Displaying plex information" on page 293. Example: `# vxprint -pt -g mydg`

Table A-2 Administering disks

Command	Description
`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 115. Example: `# vxdiskadd sde`
`vxedit [-g diskgroup] rename \ olddisk newdisk`	Renames a disk under control of VxVM. See "Renaming a disk" on page 153. Example: `# vxedit -g mydg rename \ mydg03 mydg02`

Table A-2 Administering disks *(continued)*

Command	Description

`vxedit [-g diskgroup] set \ reserve=on|off diskname`	Sets aside/does not set aside a disk from use in a disk group.
	See "Reserving disks" on page 154.
	Examples:
	`# vxedit -g mydg set \ reserve=on mydg02`
	`# vxedit -g mydg set \ reserve=off mydg02`

`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 427.
	See "Making a disk available for hot-relocation use" on page 428.
	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 425.
	See "Removing a disk from use as a hot-relocation spare" on page 426.
	Examples:
	`# vxedit -g mydg set \ spare=on mydg04`
	`# vxedit -g mydg set \ spare=off mydg04`

Table A-2 Administering disks *(continued)*

Command	Description
`vxdisk offline devicename`	Takes a disk offline. See "Taking a disk offline" on page 153. Example: `# vxdisk offline sde`
`vxdisk -g diskgroup [-o full] reclaim disk|enclosure|diskgroup`	Performs thin reclamation on a disk, enclosure, or disk group. See "Thin Reclamation of a disk, a disk group, or an enclosure" on page 357. Example: `# vxdisk reclaim disk1`
`vxdg -g diskgroup rmdisk diskname`	Removes a disk from its disk group. See "Removing a disk from a disk group" on page 232. Example: `# vxdg -g mydg rmdisk mydg02`
`vxdiskunsetup devicename`	Removes a disk from control of VxVM. See "Removing a disk from a disk group" on page 232. Example: `# vxdiskunsetup sdg`

Table A-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 230. See "Creating a shared disk group" on page 467. Example: `# vxdg init mydg \ mydg01=sde`

Table A-3 Creating and administering disk groups *(continued)*

Command	Description
`vxdg -g diskgroup listssbinfo`	Reports conflicting configuration information. See "Handling conflicting configuration copies" on page 255. 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 234. 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 235. 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. See "Importing disk groups as shared" on page 467. Example: `# vxdg -n newsdg -s import \ mysdg`

Table A-3 Creating and administering disk groups *(continued)*

Command	Description
`vxdg [-o expand] listmove sourcedg \ targetdg object ...`	Lists the objects potentially affected by moving a disk group. See "Listing objects potentially affected by a move" on page 267. Example: `# vxdg -o expand listmove \ mydg newdg myvol1`
`vxdg [-o expand] move sourcedg \ targetdg object ...`	Moves objects between disk groups. See "Moving objects between disk groups" on page 269. Example: `# vxdg -o expand move mydg \ newdg myvol1`
`vxdg [-o expand] split sourcedg \ targetdg object ...`	Splits a disk group and moves the specified objects into the target disk group. See "Splitting disk groups" on page 272. Example: `# vxdg -o expand split mydg \ newdg myvol2 myvol3`
`vxdg join sourcedg targetdg`	Joins two disk groups. See "Joining disk groups" on page 273. Example: `# vxdg join newdg mydg`

Table A-3 Creating and administering disk groups *(continued)*

Command	Description
`vxdg -g diskgroup set \ activation=ew|ro|sr|sw|off`	Sets the activation mode of a shared disk group in a cluster. See "Changing the activation mode on a shared disk group" on page 470. Example: `# vxdg -g mysdg set \ activation=sw`
`vxrecover -g diskgroup -sb`	Starts all volumes in an imported disk group. See "Moving disk groups between systems" on page 238. Example: `# vxrecover -g mydg -sb`
`vxdg destroy diskgroup`	Destroys a disk group and releases its disks. See "Destroying a disk group" on page 276. Example: `# vxdg destroy mydg`

Table A-4 Creating and administering subdisks

Command	Description
`vxmake [-g diskgroup] sd subdisk \ diskname,offset,length`	Creates a subdisk. See "Creating subdisks" on page 284. Example: `# vxmake -g mydg sd \ mydg02-01 mydg02,0,8000`

Table A-4 Creating and administering subdisks *(continued)*

Command	Description
`vxsd [-g diskgroup] assoc plex \ subdisk...`	Associates subdisks with an existing plex. See "Associating subdisks with plexes" on page 287. Example: `# vxsd -g mydg assoc home-1 \ mydg02-01 mydg02-00 \ mydg02-01`
`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. See "Associating subdisks with plexes" on page 287. 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. See "Moving subdisks" on page 286. Example: `# vxsd -g mydg mv mydg01-01 \ mydg02-01`
`vxsd [-g diskgroup] -s size split \ subdisk sd1 sd2`	Splits a subdisk in two. See "Splitting subdisks" on page 286. Example: `# vxsd -g mydg -s 1000m \ split mydg03-02 mydg03-02 \ mydg03-03`

Table A-4 Creating and administering subdisks *(continued)*

Command	Description
`vxsd [-g diskgroup] join \ sd1 sd2 ... subdisk`	Joins two or more subdisks. See "Joining subdisks" on page 287. 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. See "Moving relocated subdisks using vxassist" on page 431. 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.
`vxunreloc [-g diskgroup] original_disk`	Relocates subdisks to their original disks. See "Moving relocated subdisks using vxunreloc" on page 431. Example: `# vxunreloc -g mydg mydg01`
`vxsd [-g diskgroup] dis subdisk`	Dissociates a subdisk from a plex. See "Dissociating subdisks from plexes" on page 290. Example: `# vxsd -g mydg dis mydg02-01`

Table A-4 Creating and administering subdisks *(continued)*

Command	Description
`vxedit [-g diskgroup] rm subdisk`	Removes a subdisk. See "Removing subdisks" on page 291. Example: `# vxedit -g mydg rm mydg02-01`

`vxsd [-g diskgroup] -o rm dis subdisk`	Dissociates and removes a subdisk from a plex.
	See "Dissociating subdisks from plexes" on page 290.
	Example:
	`# vxsd -g mydg -o rm dis \`
	`  mydg02-01`

Table A-5 Creating and administering plexes

Command	Description

`vxmake [-g diskgroup] plex plex \`
`sd=subdisk1[,subdisk2,...]`
	Creates a concatenated plex.
	See "Creating plexes" on page 293.
	Example:
	`# vxmake -g mydg plex \`
	`  vol01-02 \`
	`  sd=mydg02-01,mydg02-02`

`vxmake [-g diskgroup] plex plex \`
`layout=stripe|raid5 stwidth=W \`
`ncolumn=N \`
`sd=subdisk1[,subdisk2,...]`
	Creates a striped or RAID-5 plex.
	See "Creating a striped plex" on page 293.
	Example:
	`# vxmake -g mydg plex pl-01 \`
	`  layout=stripe stwidth=32 \`
	`  ncolumn=2 \`
	`  sd=mydg01-01,mydg02-01`

Table A-5 Creating and administering plexes *(continued)*

Command	Description

`vxplex [-g diskgroup] att volume plex`	Attaches a plex to an existing volume.
	See "Attaching and associating plexes" on page 298.
	See "Reattaching plexes" on page 300.
	Example:
	`# vxplex -g mydg att vol01 \`
	`vol01-02`

`vxplex [-g diskgroup] det plex`	Detaches a plex. See "Detaching plexes" on page 300. Example: `# vxplex -g mydg det vol01-02`
`vxmend [-g diskgroup] off plex`	Takes a plex offline for maintenance. See "Taking plexes offline" on page 299. Example: `# vxmend -g mydg off vol02-02`
`vxmend [-g diskgroup] on plex`	Re-enables a plex for use. See "Reattaching plexes" on page 300. Example: `# vxmend -g mydg on vol02-02`

`vxplex [-g diskgroup] mv oldplex \`
`newplex`
	Replaces a plex.
	See "Moving plexes" on page 302.
	Example:
	`# vxplex -g mydg mv \`
	`vol02-02 vol02-03`

Table A-5 Creating and administering plexes *(continued)*

Command	Description
`vxplex [-g diskgroup] cp volume \ newplex`	Copies a volume onto a plex. See "Copying volumes to plexes" on page 303. 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 plexes" on page 300. Example: `# vxmend -g mydg fix clean \ vol02-02`
`vxplex [-g diskgroup] -o rm dis plex`	Dissociates and removes a plex from a volume. See "Dissociating and removing plexes" on page 303. Example: `# vxplex -g mydg -o rm dis \ vol03-01`

Table A-6 Creating volumes

Command	Description
`vxassist [-g diskgroup] maxsize \ layout=layout [attributes]`	Displays the maximum size of volume that can be created. See "Discovering the maximum size of a volume" on page 315. Example: `# vxassist -g mydg maxsize \ layout=raid5 nlog=2`

Table A-6 Creating volumes *(continued)*

Command	Description
`vxassist -b [-g diskgroup] make \ volume length [layout=layout] \ [attributes]`	Creates a volume. See "Creating a volume on any disk" on page 316. See "Creating a volume on specific disks" on page 317. 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 323. 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. See "Creating volumes with exclusive open access by a node" on page 471. Example: `# vxassist -b -g mysdg make \ mysmvol 20g layout=mirror \ exclusive=on`

Table A-6 Creating volumes *(continued)*

Command	Description
`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 329. See "Creating a RAID-5 volume" on page 333. Example: `# vxassist -b -g mydg make \ mysvol 20g layout=stripe \ stripeunit=32 ncol=4`
`vxassist -b [-g diskgroup] make \ volume length layout=mirror \ mirror=ctlr [attributes]`	Creates a volume with mirrored data plexes on separate controllers. See "Mirroring across targets, controllers or enclosures" on page 331. Example: `# vxassist -b -g mydg make \ mymcvol 20g layout=mirror \ mirror=ctlr`
`vxmake -b [-g diskgroup] \ -Uusage_type vol volume \ [len=length] plex=plex,...`	Creates a volume from existing plexes. See "Creating a volume using vxmake" on page 336. Example: `# vxmake -g mydg -Uraid5 \ vol r5vol \ plex=raidplex,raidlog1,\ raidlog2`
`vxvol [-g diskgroup] start volume`	Initializes and starts a volume for use. See "Initializing and starting a volume" on page 338. See "Starting a volume" on page 362. Example: `# vxvol -g mydg start r5vol`

Table A-6 Creating volumes *(continued)*

Command	Description
`vxvol [-g diskgroup] init zero \ volume`	Initializes and zeros out a volume for use. See "Initializing and starting a volume" on page 338. Example: `# vxvol -g mydg init zero \ myvol`

Table A-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 367. 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 369. 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.

Table A-7 Administering volumes *(continued)*

Command	Description
`vxassist [-g diskgroup] \ {growto|growby} volume length`	Grows a volume to a specified size or by a specified amount. See "Resizing volumes with vxassist" on page 365. 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. See "Resizing volumes with vxassist" on page 365. Example: `# vxassist -g mydg shrinkto \ myvol 20g`
`vxresize -b -F vxfs [-g diskgroup] \ volume length diskname ...`	Resizes a volume and the underlying Veritas File System. See "Resizing volumes with vxresize" on page 364. 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 "Preparing a volume for DRL and instant snapshots" on page 371. Example: `# vxsnap -g mydg prepare \ myvol drl=on`

Table A-7 Administering volumes *(continued)*

Command	Description

`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.
	For information about creating snapshots, see the *Veritas Storage Foundation Advanced Features Administrator's Guide*.
	Example:
	`# vxsnap -g mydg make \`
	`source=myvol/\`
	`newvol=mysnpvol/\`
	`nmirror=2`

`vxsnap [-g diskgroup] make \`
`source=volume/snapvol=snapvol`
	Takes a full-sized instant snapshot of a volume using a prepared empty volume.
	For information about snapshots, see the *Veritas Storage Foundation Advanced Features Administrator's Guide*.
	Example:
	`# vxsnap -g mydg make \`
	`source=myvol/snapvol=snpvol`

Table A-7 Administering volumes *(continued)*

Command	Description

`vxmake [-g diskgroup] cache \ cache_object cachevolname=volume \ [regionsize=size]`	Creates a cache object for use by space-optimized instant snapshots.
	For information about snapshots, see the *Veritas Storage Foundation Advanced Features Administrator's Guide*.
	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. For information about creating snapshots, see the *Veritas Storage Foundation Advanced Features Administrator's Guide*. Example: `# vxsnap -g mydg make \ source=myvol/\ newvol=mysosvol/\ cache=cobj`

Table A-7 Administering volumes *(continued)*

Command	Description
`vxsnap [-g diskgroup] refresh snapshot`	Refreshes a snapshot from its original volume. For information about snapshots, see the *Veritas Storage Foundation Advanced Features Administrator's Guide*. Example: `# vxsnap -g mydg refresh \ mysnpvol`
`vxsnap [-g diskgroup] dis snapshot`	Turns a snapshot into an independent volume. For information about snapshots, see the *Veritas Storage Foundation Advanced Features Administrator's Guide*. Example: `# vxsnap -g mydg dis mysnpvol`
`vxsnap [-g diskgroup] unprepare \ volume`	Removes support for instant snapshots and DRL logging from a volume. For information about snapshots, see the *Veritas Storage Foundation Advanced Features Administrator's Guide*. See "Removing support for DRL and instant snapshots from a volume" on page 376. Example: `# vxsnap -g mydg unprepare \ myvol`

Table A-7 Administering volumes *(continued)*

Command	Description
`vxassist [-g diskgroup] relayout \ volume [layout=layout] \ [relayout_options]`	Performs online relayout of a volume. See "Performing online relayout" on page 387. Example: `# vxassist -g mydg relayout \ vol2 layout=stripe`
`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 387. 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 70. 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. See "Converting between layered and non-layered volumes" on page 393. Example: `# vxassist -g mydg convert \ vol3 layout=stripe-mirror`

Table A-7 Administering volumes *(continued)*

Command	Description
`vxassist [-g diskgroup] remove \ volume volume`	Removes a volume. See "Removing a volume" on page 383. Example: `# vxassist -g mydg remove \ myvol`

Table A-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 352. 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 354. Example: `# vxtask -h -g mydg list`
`vxtask monitor task`	Monitors the progress of a task. See "Using the vxtask command" on page 354. Example: `# vxtask monitor mytask`

Table A-8 Monitoring and controlling tasks *(continued)*

Command	Description
`vxtask pause task`	Suspends operation of a task. See "Using the vxtask command" on page 354. Example: `# vxtask pause mytask`
`vxtask -p [-g diskgroup] list`	Lists all paused tasks. See "Using the vxtask command" on page 354. Example: `# vxtask -p -g mydg list`
`vxtask resume task`	Resumes a paused task. See "Using the vxtask command" on page 354. Example: `# vxtask resume mytask`
`vxtask abort task`	Cancels a task and attempts to reverse its effects. See "Using the vxtask command" on page 354. Example: `# vxtask abort mytask`

CVM commands supported for executing on the slave node

Table A-9 shows the complete list of commands that are supported for executing on the slave node.

Table A-9 List of CVM commands supported for executing on the slave node

Command	Supported operations
vxdg	

Table A-9 List of CVM commands supported for executing on the slave node *(continued)*

Command	Supported operations
	vxdg -s init <shared_dg> [cds=on|off]
	vxdg -T < different_versions> -s init <shared_dg> [minor=base-minor] [cds=on|off]
	vxdg [-n newname] [-h new-host-id] deport <shared_dg>
	vxdg [-Cfst] [-n newname] [-o clearreserve] [-o useclonedev={on|off}] [-o updateid] [-o noreonline] [-o selectcp=diskid] [-o dgtype=shared] import <shared_dg>
	vxdg destroy <shared_dg>
	vxdg -g <shared_dg> [-o overridessb] [-f] adddisk [disk=]device
	vxdg-g <shared_dg>addsite site
	vxdg -g <shared_dg> reattachsite site
	vxdg -g <shared_dg> detachsite site
	vxdg -g <shared_dg> rmsite site
	vxdg -g <shared_dg> renamesite oldname newname
	vxdg flush <shared_dg>
	vxdg [-qa] -g <shared_dg> free [medianame...]
	vxdg join sourcedg targetdg (both dgs should be shared)
	vxdg split sourcedg targetdg
	vxdg [-q] [-s] [-o listreserve] list [diskgroup...]
	vxdg [-o expand] move sourcedg targetdg object (both dgs should be shared)
	vxdg -g shared_dg recover
	vxdg -g <shared_dg> [-f] reminor <shared_dg> new-minor-number
	vxdg -g <shared_dg> rmdisk medianame...
	vxdg -g <shared_dg>[-q] spare [medianame...]
	vxdg -g <shared_dg> [-f] [-o retain|replace] settag [encl:<enclosure>] name[=value name[=value]
	vxdg [-q] listtag <shared_dg>
	vxdg -g <shared_dg> rmtag [encl:<enclosure>] name=value
	vxdg -g <shared_dg> set siteconsistent=on
	vxdg upgrade <shared_dg>

Table A-9 List of CVM commands supported for executing on the slave node *(continued)*

Command	Supported operations
	vxdg -g <shared_dg>set attr=value ...
vxassist	vxassist -g <shared_dg> [-b] convert volume layout=<type> vxassist -g <shared_dg> [-b] addlog volume vxassist -g <shared_dg> [-b] mirror volume vxassist [-b]-g <shared_dg>make volume length [layout=layout] diskname ... vxassist -g <shared_dg> [-b] growby volume lengthchange [attribute ...] vxassist [-b] -g <shared_dg> growto volume newlength vxassist -g <shared_dg> shrinkby volume lengthchange vxassist -g <shared_dg> shrinkto volume newlength vxassist -g <shared_dg> settag volume|vset tagname[=tagvalue] vxassist -g <shared_dg>replacetag volume|vset oldtag newtag vxassist -g <shared_dg> removetag volume|vset tagname vxassist -g <shared_dg> move volume-name storage-spec vxassist -g <shared_dg> relayout {volume-name} layout=<type> vxassist -g <shared_dg> remove {volume|mirror|log} volume-name vxassist -g <shared_dg> snapshot volume-name [snapshot-name] [comment=<comment>] vxassist -g <shared_dg> snapstart volume vxassist -g <shared_dg> maxsize layout=<> nmirror=<> / nlog=<> vxassist -g <shared_dg> maxgrow volume vxassist -g <shared_dg> snapback snapvol vxassist -g <shared_dg> snapclear snapvol1

Table A-9 List of CVM commands supported for executing on the slave node *(continued)*

Command	Supported operations
vxcache	vxcache -g <shared_dg> start cacheobject
	vxcache -g <shared_dg> stop cacheobject
	vxcache -g <shared_dg> att volume cacheobject
	vxcache -g <shared_dg> dis cachevol
	vxcache -g <shared_dg> shrinkcacheto cacheobject newlength
	vxcache -g <shared_dg> shrinkcacheby cacheobject lengthchange
	vxcache -g <shared_dg> growcacheto cacheobject newlength
	vxcache -g <shared_dg> growcacheby cacheobject lengthchange
vxdco	vxdco -g <shared_dg> dis dco
	vxdco -g <shared_dg> att volume dco
	vxdco -g <shared_dg>[-o force] enable dco
vxedit	vxedit -g <shared_dg> set comment="plex comment" plex1
	vxedit -g <shared_dg> -rf rm volume
	vxedit -g <shared_dg>rename oldname newname
	vxedit -g <shared_dg> set what=value
	vxedit-g <shared_dg> set user=value mode=value medianame
	vxedit -g <shared_dg> set failing=off <disk name>
	vxedit -g <shared_dg> set fstype volumename
	vxedit -g <shared_dg> set len subdisk
	vxedit -g <shared_dg> set orig_dmname subdisk
	vxedit -g <shared_dg> set orig_dmoffset subdisk
	vxedit -g <shared_dg> set diskdetpolicy diskgroup
vxmake	vxmake -g <shared_dg> sd name [attr...]
	vxmake-g <shared_dg>plex plex sd=subdisk1[,subdisk2,...]
	vxmake -g <shared_dg> -U fsgen vol homevol1 plex=plex-1
	vxmake -g <shared_dg> -U fsgen vol volume1 plex=plex1,plex2
	vxmake -g <shared_dg> cache name regionsize=<size>
	vxmake -g <shared_dg> dco volume log=dco

Table A-9 List of CVM commands supported for executing on the slave node *(continued)*

Command	Supported operations
vxmend	vxmend -g <shared_dg> on plex vxmend -g <shared_dg> off plex
vxmirror	vxmirror -g <shared_dg>medianame vxmirror -g <shared_dg> -d [yes|no]
vxplex	vxplex -g <shared_dg> att volume plex vxplex -g <shared_dg> cp volume new_plex vxplex -g <shared_dg> dis plex1 vxplex -g <shared_dg>mv original_plex new_plex vxplex -g <shared_dg> snapstart vol snapplex vxplex -g <shared_dg> snaphot snapplex vxplex -g <shared_dg> snapback vol snapplex vxplex -g <shared_dg> plex
vxrelayout	vxrelayout -g <shared_dg> status volume vxrelayout -g <shared_dg> start volume vxrelayout -g <shared_dg> reverse volname
vxsd	vxsd -g <shared_dg> assoc plex subdisk1 [subdisk2 subdisk3 ...] vxsd -g <shared_dg> [-o force] dis subdisk vxsd -g <shared_dg> mv old_subdisk new_subdisk [new_subdisk ...] vxsd -g <shared_dg> aslog plex2 sdisk3 vxsd -g <shared_dg> join subdisk1 subdisk2 ... new_subdisk vxsd -g <shared_dg> [-o force] dis subdisk vxsd -g <shared_dg> split subdisk newsd [newsd2]...

Table A-9 List of CVM commands supported for executing on the slave node *(continued)*

Command	Supported operations
vxsnap	vxsnap -g <shared_dg> addmir volume [nmirror=N]
	vxsnap -g <shared_dg> prepare volume
	vxsnap -g <shared_dg> rmmir volume
	vxsnap -g <shared_dg>unprepare volume
	vxsnap -g <shared_dg> make snapshot_tuple [snapshot_tuple]... [alloc=storage_attributes]
	vxsnap [-f] -g <shared_dg> dis volume
	vxsnap -g <shared_dg> addmap volumename count
	vxsnap -g <shared_dg> print volumename
	vxsnap -g <shared_dg> list volumename
	vxsnap -g <shared_dg> syncwait snapvol
	vxsnap -g <shared_dg> snapwait
	vxsnap -g <shared_dg> refresh snapvol source=volume
	vxsnap -g <shared_dg> restore target source=volname
	vxsnap -g <shared_dg> split volumename
	vxsnap -g <shared_dg> reattach volname source=volname
vxsnptadm	vxsnptadm -g <shared_dg> create vol [snptname=snpt] [snapvolname=snapvol] [data={yes|no}]
	vxsnptadm -g <shared_dg> info vol [snptname=snpt]
	vxsnptadm -g <shared_dg> remove vol snptname=snpt
	vxsnptadm -g <shared_dg> removeall vol [cookie=cookie]
	vxsnptadm -g <shared_dg> rename vol snptname=snpt newname=snpt2

Table A-9 List of CVM commands supported for executing on the slave node *(continued)*

Command	Supported operations
vxvol	vxvol -g <shared_dg> set logtype=drl | drlseq volume vxvol -g <shared_dg> start volume vxvol -g <shared_dg> stop volume vxvol -g <shared_dg> {startall|stopall} volume vxvol -g <shared_dg> init enable volume vxvol -g <shared_dg> init active volume vxvol -g <shared_dg> maint volumename vxvol -g <shared_dg> set len volumename vxvol -g <shared_dg> set logtype volumename vxvol -g <shared_dg> set loglen volumename
vxvset	vxvset -g <shared_dg> make volume-set-name volume-name vxvset -g <shared_dg> addvol volume-set-name volume-name vxvset -g <shared_dg> list volume-set-name vxvset -g <shared_dg> rmvol volume-set-name volume-name vxvset -g <shared_dg> stop volume-set-name vxvset -g <shared_dg> start volume-set-name
vxevac	vxevac -g <shared_dg> medianame
vxresize	vxresize [-Vsb] [-F fstype] -g <shared_dg> volume length
vxrecover	vxrecover -g <shared_dg> vxrecover -g <shared_dg> volume
vxckdiskrm	vxckdiskrm -g <shared_dg> medianame

Online manual pages

Manual pages are organized into the following sections:

1M	Administrative commands.
4	File formats.

Section 1M — administrative commands

Table A-10 lists the manual pages in section 1M for commands that are used to administer Veritas Volume Manager.

Table A-10 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.
vxcached	Resize 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.

Table A-10 Section 1M manual pages *(continued)*

Name	Description
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.
vxdiskunsetup	Deconfigure a disk from use with Veritas Volume Manager.
vxdmpadm	DMP subsystem administration.
vxdmptune	Display and change values of DMP tunable parameters.
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.

Table A-10 Section 1M manual pages *(continued)*

Name	Description
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.
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.

Table A-10 Section 1M manual pages *(continued)*

Name	Description
`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.
`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.
`vxvset`	Create and administer volume sets.

Section 4 — file formats

Table A-11 lists the manual pages in section 4 that describe the format of files that are used by Veritas Volume Manager.

Table A-11 Section 4 manual pages

Name	Description
`vol_pattern`	Disk group search specifications.
`vxmake`	`vxmake` description file.

Appendix B

Configuring Veritas Volume Manager

This appendix includes the following topics:

Setup tasks after installation

A number of setup tasks can be performed after installing the Veritas Volume Manager (VxVM) software.

The following tasks are to perform initial setup:

- Create disk groups by placing disks under Veritas Volume Manager control.
- Create volumes in the disk groups.
- Configure file systems on the volumes.

The following setup tasks are optional:

- Encapsulate the `root` disk, and mirror it to create an alternate boot disk.
- Designate hot-relocation spare disks in each disk group.

- Add mirrors to volumes.
- Configure DRL and FastResync on volumes.

The following tasks are to perform ongoing maintenance:

- Resize volumes and file systems.
- Add more disks, create new disk groups, and create new volumes.
- Create and maintain snapshots.

Unsupported disk arrays

After installation, add any disk arrays that are unsupported by Symantec to the DISKS (JBOD) category.

See "How to administer the Device Discovery Layer" on page 86.

Foreign devices

The device discovery feature of VxVM can discover some devices that are controlled by third-party drivers, such as for EMC PowerPath. For these devices it may be preferable to use the multipathing capability that is provided by the third-party drivers rather than using the Dynamic Multi-Pathing (DMP) feature. Provided that a suitable array support library is available, DMP can co-exist with such drivers. Other foreign devices, for which a compatible ASL does not exist, can be made available to Veritas Volume Manager as simple disks by using the `vxddladm addforeign` command. This also has the effect of bypassing DMP.

See "How to administer the Device Discovery Layer" on page 86.

Initialization of disks and creation of disk groups

To place disks in disk groups, use the `vxdiskadm` program after completing the installation.

See "Adding a disk to VxVM" on page 106.

Guidelines for configuring storage

A disk failure can cause loss of data on the failed disk and loss of access to your system. Loss of access is due to the failure of a key disk used for system operations. Veritas Volume Manager can protect your system from these problems.

To maintain system availability, data important to running and booting your system must be mirrored. The data must be preserved so it can be used in case of failure.

The following are suggestions for protecting your system and data:

- Perform regular backups to protect your data. Backups are necessary if all copies of a volume are lost or corrupted. Power surges can damage several (or all) disks on your system. Also, typing a command in error can remove critical files or damage a file system directly. Performing regular backups ensures that lost or corrupted data is available to be retrieved.
- Place the disk containing the `root` file system (the root or boot disk) under Veritas Volume Manager control through encapsulation. Encapsulation converts the `root` and `swap` devices to volumes (`rootvol` and `swapvol`). Mirror the root disk so that an alternate root disk exists for booting purposes. By mirroring disks critical to booting, you ensure that no single disk failure leaves your system unbootable and unusable.
  See "Rootability" on page 122.
- Use mirroring to protect data against loss from a disk failure.
  See "Mirroring guidelines" on page 563.
- Use the DRL feature to speed up recovery of mirrored volumes after a system crash.
  See "Dirty region logging guidelines" on page 564.
- Use striping to improve the I/O performance of volumes.
  See "Striping guidelines" on page 565.
- Make sure enough disks are available for a combined striped and mirrored configuration. At least two disks are required for the striped plex, and one or more additional disks are needed for the mirror.
- When combining striping and mirroring, never place subdisks from one plex on the same physical disk as subdisks from the other plex.
- Use logging to prevent corruption of recovery data in RAID-5 volumes. Make sure that each RAID-5 volume has at least one log plex.
  See "RAID-5 guidelines" on page 566.
- Leave the Veritas Volume Manager hot-relocation feature enabled.
  See "Hot-relocation guidelines" on page 566.

Mirroring guidelines

Refer to the following guidelines when using mirroring.

- Do not place subdisks from different plexes of a mirrored volume on the same physical disk. This action compromises the availability benefits of mirroring and degrades performance. Using the `vxassist` or `vxdiskadm` commands precludes this from happening.
- To provide optimum performance improvements through the use of mirroring, at least 70 percent of physical I/O operations should be read operations. A higher percentage of read operations results in even better performance. Mirroring may not provide a performance increase or may even result in a performance decrease in a write-intensive workload environment.
- The operating system implements a file system cache. Read requests can frequently be satisfied from the cache. This can cause the read/write ratio for physical I/O operations through the file system to be biased toward writing (when compared to the read/write ratio at the application level).
- Where possible, use disks attached to different controllers when mirroring or striping. Most disk controllers support overlapped seeks. This allows seeks to begin on two disks at once. Do not configure two plexes of the same volume on disks that are attached to a controller that does not support overlapped seeks. This is important for older controllers or SCSI disks that do not cache on the drive. It is less important for modern SCSI disks and controllers. Mirroring across controllers allows the system to survive a failure of one of the controllers. Another controller can continue to provide data from a mirror.
- A plex exhibits superior performance when striped or concatenated across multiple disks, or when located on a much faster device. Set the read policy to prefer the faster plex. By default, a volume with one striped plex is configured to prefer reading from the striped plex.

See "Mirroring (RAID-1)" on page 40.

Dirty region logging guidelines

Dirty region logging (DRL) can speed up recovery of mirrored volumes following a system crash. When DRL is enabled, Veritas Volume Manager keeps track of the regions within a volume that have changed as a result of writes to a plex.

Warning: Using Dirty Region Logging can adversely impact system performance in a write-intensive environment.

See "Dirty region logging" on page 56.

Striping guidelines

Refer to the following guidelines when using striping.

- Do not place more than one column of a striped plex on the same physical disk.
- Calculate stripe-unit sizes carefully. In general, a moderate stripe-unit size (for example, 64 kilobytes, which is also the default used by `vxassist`) is recommended.
- If it is not feasible to set the stripe-unit size to the track size, and you do not know the application I/O pattern, use the default stripe-unit size.
- Many modern disk drives have variable geometry. This means that the track size differs between cylinders, so that outer disk tracks have more sectors than inner tracks. It is therefore not always appropriate to use the track size as the stripe-unit size. For these drives, use a moderate stripe-unit size (such as 64 kilobytes), unless you know the I/O pattern of the application.
- Volumes with small stripe-unit sizes can exhibit poor sequential I/O latency if the disks do not have synchronized spindles. Generally, striping over disks without synchronized spindles yields better performance when used with larger stripe-unit sizes and multi-threaded, or largely asynchronous, random I/O streams.
- Typically, the greater the number of physical disks in the stripe, the greater the improvement in I/O performance; however, this reduces the effective mean time between failures of the volume. If this is an issue, combine striping with mirroring to combine high-performance with improved reliability.
- If only one plex of a mirrored volume is striped, set the policy of the volume to `prefer` for the striped plex. (The default read policy, `select`, does this automatically.)
- If more than one plex of a mirrored volume is striped, configure the same stripe-unit size for each striped plex.
- Where possible, distribute the subdisks of a striped volume across drives connected to different controllers and buses.
- Avoid the use of controllers that do not support overlapped seeks. (Such controllers are rare.)

The `vxassist` command automatically applies and enforces many of these rules when it allocates space for striped plexes in a volume.

See "Striping (RAID-0)" on page 37.

RAID-5 guidelines

Refer to the following guidelines when using RAID-5.

In general, the guidelines for mirroring and striping together also apply to RAID-5. The following guidelines should also be observed with RAID-5:

- Only one RAID-5 plex can exist per RAID-5 volume (but there can be multiple log plexes).
- The RAID-5 plex must be derived from at least three subdisks on three or more physical disks. If any log plexes exist, they must belong to disks other than those used for the RAID-5 plex.
- RAID-5 logs can be mirrored and striped.
- If the volume length is not explicitly specified, it is set to the length of any RAID-5 plex associated with the volume; otherwise, it is set to zero. If you specify the volume length, it must be a multiple of the stripe-unit size of the associated RAID-5 plex, if any.
- If the log length is not explicitly specified, it is set to the length of the smallest RAID-5 log plex that is associated, if any. If no RAID-5 log plexes are associated, it is set to zero.
- Sparse RAID-5 log plexes are not valid.
- RAID-5 volumes are not supported for sharing in a cluster.

See "RAID-5 (striping with parity)" on page 43.

Hot-relocation guidelines

Hot-relocation automatically restores redundancy and access to mirrored and RAID-5 volumes when a disk fails. This is done by relocating the affected subdisks to disks designated as spares and/or free space in the same disk group.

The hot-relocation feature is enabled by default. The associated daemon, `vxrelocd`, is automatically started during system startup.

Refer to the following guidelines when using hot-relocation.

- The hot-relocation feature is enabled by default. Although it is possible to disable hot-relocation, it is advisable to leave it enabled. It will notify you of the nature of the failure, attempt to relocate any affected subdisks that are redundant, and initiate recovery procedures.
- Although hot-relocation does not require you to designate disks as spares, designate at least one disk as a spare within each disk group. This gives you some control over which disks are used for relocation. If no spares exist, Veritas Volume Manager uses any available free space within the disk group. When

free space is used for relocation purposes, it is possible to have performance degradation after the relocation.

- After hot-relocation occurs, designate one or more additional disks as spares to augment the spare space. Some of the original spare space may be occupied by relocated subdisks.
- If a given disk group spans multiple controllers and has more than one spare disk, set up the spare disks on different controllers (in case one of the controllers fails).
- For a mirrored volume, configure the disk group so that there is at least one disk that does not already contain a mirror of the volume. This disk should either be a spare disk with some available space or a regular disk with some free space and the disk is not excluded from hot-relocation use.
- For a mirrored and striped volume, configure the disk group so that at least one disk does not already contain one of the mirrors of the volume or another subdisk in the striped plex. This disk should either be a spare disk with some available space or a regular disk with some free space and the disk is not excluded from hot-relocation use.
- For a RAID-5 volume, configure the disk group so that at least one disk does not already contain the RAID-5 plex (or one of its log plexes) of the volume. This disk should either be a spare disk with some available space or a regular disk with some free space and the disk is not excluded from hot-relocation use.
- If a mirrored volume has a DRL log subdisk as part of its data plex, you cannot relocate the data plex. Instead, place log subdisks in log plexes that contain no data.
- Hot-relocation does not guarantee to preserve the original performance characteristics or data layout. Examine the locations of newly-relocated subdisks to determine whether they should be relocated to more suitable disks to regain the original performance benefits.
- Although it is possible to build Veritas Volume Manager objects on spare disks, it is recommended that you use spare disks for hot-relocation only.

See "How hot-relocation works" on page 418.

Accessing volume devices

As soon as a volume has been created and initialized, it is available for use as a virtual disk partition by the operating system for the creation of a file system, or by application programs such as relational databases and other data management software.

Creating a volume in a disk group sets up block and character (raw) device files that can be used to access the volume:

`/dev/vx/dsk/dg/vol`	block device file for volume *vol* in disk group *dg*
`/dev/vx/rdsk/dg/vol`	character device file for volume *vol* in disk group *dg*

The pathnames include a directory named for the disk group. Use the appropriate device node to create, mount and repair file systems, and to lay out databases that require raw partitions.

VxVM's view of multipathed devices

You can use the `vxdiskadm` command to control how a device is treated by the Veritas Dynamic Multi-Pathing (DMP).

See "Disabling multi-pathing and making devices invisible to VxVM" on page 166.

Cluster support

The Veritas Volume Manager software includes a licensable feature that enables it to be used in a cluster environment. The cluster functionality in Veritas Volume Manager allows multiple hosts to simultaneously access and manage a set of disks under Veritas Volume Manager control. A cluster is a set of hosts sharing a set of disks; each host is referred to as a node in the cluster.

See the *Veritas Storage Foundation Getting Started Guide*.

Configuring shared disk groups

If you are installing Veritas Volume Manager for the first time or adding disks to an existing cluster, you need to configure new shared disks.

Note: RAID-5 volumes are not supported for sharing in a cluster.

If you are setting up Veritas Volume Manager for the first time, configure the shared disks using the following steps in the specified order:

- Start the cluster on one node only to prevent access by other nodes.

- On one node, run the `vxdiskadm` program and choose option 1 to initialize new disks. When asked to add these disks to a disk group, choose `none` to leave the disks for future use.
- On other nodes in the cluster, run `vxdctl enable` to see the newly initialized disks.
- Create disk groups on the shared disks.
- Use the `vxdg` command or the Veritas Operations Manager (VOM) to create disk groups. If you use the `vxdg` command, specify the `-s` option to create shared disk groups.
- Use `vxassist` or VOM to create volumes in the disk groups.
- If the cluster is only running with one node, bring up the other cluster nodes. Enter the `vxdg list` command on each node to display the shared disk groups.

Converting existing VxVM disk groups to shared disk groups

To convert existing disk groups to shared disk groups

1. Start the cluster on one node only to prevent access by other nodes.
2. Configure the disk groups using the following procedure.

   To list all disk groups, use the following command:

   ```
   # vxdg list
   ```

   To deport the disk groups that are to be shared, use the following command:

   ```
   # vxdg deport diskgroup
   ```

   To import disk groups to be shared, use the following command:

   ```
   # vxdg -s import diskgroup
   ```

   This procedure marks the disks in the shared disk groups as shared and stamps them with the ID of the cluster, enabling other nodes to recognize the shared disks.

   If dirty region logs exist, ensure they are active. If not, replace them with larger ones.

   To display the shared flag for all the shared disk groups, use the following command:

   ```
   # vxdg list
   ```

   The disk groups are now ready to be shared.
3. Bring up the other cluster nodes. Enter the `vxdg list` command on each node to display the shared disk groups. This command displays the same list of shared disk groups displayed earlier.

   See the *Veritas Storage Foundation Cluster File System Installation Guide*.

Glossary

Active/Active disk arrays	This type of multipathed disk array allows you to access a disk in the disk array through all the paths to the disk simultaneously, without any performance degradation.
Active/Passive disk arrays	This type of multipathed disk array allows one path to a disk to be designated as primary and used to access the disk at any time. Using a path other than the designated active path results in severe performance degradation in some disk arrays.
associate	The process of establishing a relationship between VxVM objects; for example, a subdisk that has been created and defined as having a starting point within a plex is referred to as being associated with that plex.
associated plex	A plex associated with a volume.
associated subdisk	A subdisk associated with a plex.
atomic operation	An operation that either succeeds completely or fails and leaves everything as it was before the operation was started. If the operation succeeds, all aspects of the operation take effect at once and the intermediate states of change are invisible. If any aspect of the operation fails, then the operation aborts without leaving partial changes. In a cluster, an atomic operation takes place either on all nodes or not at all.
attached	A state in which a VxVM object is both associated with another object and enabled for use.
block	The minimum unit of data transfer to or from a disk or array.
boot disk	A disk that is used for the purpose of booting a system.
boot disk group	A private disk group that contains the disks from which the system may be booted.
bootdg	A reserved disk group name that is an alias for the name of the boot disk group.
clean node shutdown	The ability of a node to leave a cluster gracefully when all access to shared volumes has ceased.
cluster	A set of hosts (each termed a node) that share a set of disks.
cluster manager	An externally-provided daemon that runs on each node in a cluster. The cluster managers on each node communicate with each other and inform VxVM of changes in cluster membership.

cluster-shareable disk group A disk group in which access to the disks is shared by multiple hosts (also referred to as a shared disk group).

column A set of one or more subdisks within a striped plex. Striping is achieved by allocating data alternately and evenly across the columns within a plex.

concatenation A layout style characterized by subdisks that are arranged sequentially and contiguously.

configuration copy A single copy of a configuration database.

configuration database A set of records containing detailed information on existing VxVM objects (such as disk and volume attributes).

DCO (data change object) A VxVM object that is used to manage information about the FastResync maps in the DCO volume. Both a DCO object and a DCO volume must be associated with a volume to implement Persistent FastResync on that volume.

data stripe This represents the usable data portion of a stripe and is equal to the stripe minus the parity region.

DCO volume A special volume that is used to hold Persistent FastResync change maps and dirty region logs. See also see dirty region logging.

detached A state in which a VxVM object is associated with another object, but not enabled for use.

device name The device name or address used to access a physical disk, such as `sda` or `sda3`, where `sda` indicates the whole device, and `sda3` refers to the third partition on `sda`.

In a SAN environment, it is more convenient to use enclosure-based naming, which forms the device name by concatenating the name of the enclosure (such as `enc0`) with the disk's number within the enclosure, separated by an underscore (for example, `enc0_2`). The term disk access name can also be used to refer to a device name.

dirty region logging The method by which the VxVM monitors and logs modifications to a plex as a bitmap of changed regions. For a volumes with a new-style DCO volume, the dirty region log (DRL) is maintained in the DCO volume. Otherwise, the DRL is allocated to an associated subdisk called a log subdisk.

disabled path A path to a disk that is not available for I/O. A path can be disabled due to real hardware failures or if the user has used the `vxdmpadm disable` command on that controller.

disk A collection of read/write data blocks that are indexed and can be accessed fairly quickly. Each disk has a universally unique identifier.

disk access name An alternative term for a device name.

disk access records Configuration records used to specify the access path to particular disks. Each disk access record contains a name, a type, and possibly some type-specific information, which is used by VxVM in deciding how to access and manipulate the disk that is defined by the disk access record.

disk array A collection of disks logically arranged into an object. Arrays tend to provide benefits such as redundancy or improved performance.

disk array serial number This is the serial number of the disk array. It is usually printed on the disk array cabinet or can be obtained by issuing a vendor- specific SCSI command to the disks on the disk array. This number is used by the DMP subsystem to uniquely identify a disk array.

disk controller In the multipathing subsystem of VxVM, the controller (host bus adapter or HBA) or disk array connected to the host, which the operating system represents as the parent node of a disk.

disk enclosure An intelligent disk array that usually has a backplane with a built-in Fibre Channel loop, and which permits hot-swapping of disks.

disk group A collection of disks that share a common configuration. A disk group configuration is a set of records containing detailed information on existing VxVM objects (such as disk and volume attributes) and their relationships. Each disk group has an administrator-assigned name and an internally defined unique ID. The disk group names `bootdg` (an alias for the boot disk group), `defaultdg` (an alias for the default disk group) and `nodg` (represents no disk group) are reserved.

disk group ID A unique identifier used to identify a disk group.

disk ID A universally unique identifier that is given to each disk and can be used to identify the disk, even if it is moved.

disk media name An alternative term for a disk name.

disk media record A configuration record that identifies a particular disk, by disk ID, and gives that disk a logical (or administrative) name.

disk name A logical or administrative name chosen for a disk that is under the control of VxVM, such as `disk03`. The term disk media name is also used to refer to a disk name.

dissociate The process by which any link that exists between two VxVM objects is removed. For example, dissociating a subdisk from a plex removes the subdisk from the plex and adds the subdisk to the free space pool.

dissociated plex A plex dissociated from a volume.

dissociated subdisk A subdisk dissociated from a plex.

distributed lock manager A lock manager that runs on different systems in a cluster, and ensures consistent access to distributed resources.

enabled path A path to a disk that is available for I/O.

encapsulation A process that converts existing partitions on a specified disk to volumes. If any partitions contain file systems, `/etc/fstab` entries are modified so that the file systems are mounted on volumes instead.

enclosure See disk enclosure.

enclosure-based naming See device name.

fabric mode disk A disk device that is accessible on a Storage Area Network (SAN) via a Fibre Channel switch.

FastResync A fast resynchronization feature that is used to perform quick and efficient resynchronization of stale mirrors, and to increase the efficiency of the snapshot mechanism.

Fibre Channel A collective name for the fiber optic technology that is commonly used to set up a Storage Area Network (SAN).

file system A collection of files organized together into a structure. The UNIX file system is a hierarchical structure consisting of directories and files.

free space An area of a disk under VxVM control that is not allocated to any subdisk or reserved for use by any other VxVM object.

free subdisk A subdisk that is not associated with any plex and has an empty `putil[0]` field.

hostid A string that identifies a host to VxVM. The host ID for a host is stored in its `volboot` file, and is used in defining ownership of disks and disk groups.

hot-relocation A technique of automatically restoring redundancy and access to mirrored and RAID-5 volumes when a disk fails. This is done by relocating the affected subdisks to disks designated as spares and/or free space in the same disk group.

hot-swap Refers to devices that can be removed from, or inserted into, a system without first turning off the power supply to the system.

initiating node The node on which the system administrator is running a utility that requests a change to VxVM objects. This node initiates a volume reconfiguration.

JBOD (just a bunch of disks) The common name for an unintelligent disk array which may, or may not, support the hot-swapping of disks.

log plex A plex used to store a RAID-5 log. The term log plex may also be used to refer to a Dirty Region Logging plex.

log subdisk A subdisk that is used to store a dirty region log.

master node A node that is designated by the software to coordinate certain VxVM operations in a cluster. Any node is capable of being the master node.

mastering node The node to which a disk is attached. This is also known as a disk owner.

mirror A duplicate copy of a volume and the data therein (in the form of an ordered collection of subdisks). Each mirror consists of one plex of the volume with which the mirror is associated.

mirroring A layout technique that mirrors the contents of a volume onto multiple plexes. Each plex duplicates the data stored on the volume, but the plexes themselves may have different layouts.

multipathing Where there are multiple physical access paths to a disk connected to a system, the disk is called multipathed. Any software residing on the host, (for example, the DMP driver) that hides this fact from the user is said to provide multipathing functionality.

node One of the hosts in a cluster.

node abort A situation where a node leaves a cluster (on an emergency basis) without attempting to stop ongoing operations.

node join The process through which a node joins a cluster and gains access to shared disks.

Non-Persistent FastResync A form of FastResync that cannot preserve its maps across reboots of the system because it stores its change map in memory.

object An entity that is defined to and recognized internally by VxVM. The VxVM objects are: volume, plex, subdisk, disk, and disk group. There are actually two types of disk objects—one for the physical aspect of the disk and the other for the logical aspect.

parity A calculated value that can be used to reconstruct data after a failure. While data is being written to a RAID-5 volume, parity is also calculated by performing an exclusive OR (XOR) procedure on data. The resulting parity is then written to the volume. 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 the parity.

parity stripe unit A RAID-5 volume storage region that contains parity information. The data contained in the parity stripe unit can be used to help reconstruct regions of a RAID-5 volume that are missing because of I/O or disk failures.

partition The standard division of a physical disk device, as supported directly by the operating system and disk drives.

path When a disk is connected to a host, the path to the disk consists of the HBA (Host Bus Adapter) on the host, the SCSI or fibre cable connector and the controller on the disk or disk array. These components constitute a path to a disk. A failure on any of these results in DMP trying to shift all I/O for that disk onto the remaining (alternate) paths.

pathgroup In the case of disks which are not multipathed by `vxdmp`, VxVM will see each path as a disk. In such cases, all paths to the disk can be grouped. This way only one of the paths from the group is made visible to VxVM.

Persistent FastResync	A form of FastResync that can preserve its maps across reboots of the system by storing its change map in a DCO volume on disk).
persistent state logging	A logging type that ensures that only active mirrors are used for recovery purposes and prevents failed mirrors from being selected for recovery. This is also known as kernel logging.
physical disk	The underlying storage device, which may or may not be under VxVM control.
plex	A plex is a logical grouping of subdisks that creates an area of disk space independent of physical disk size or other restrictions. Mirroring 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. Plexes may also be created to represent concatenated, striped and RAID-5 volume layouts, and to store volume logs.
primary path	In Active/Passive disk arrays, a disk can be bound to one particular controller on the disk array or owned by a controller. The disk can then be accessed using the path through this particular controller.
private disk group	A disk group in which the disks are accessed by only one specific host in a cluster.
private region	A region of a physical disk used to store private, structured VxVM information. The private region contains a disk header, a table of contents, and a configuration database. The table of contents maps the contents of the disk. The disk header contains a disk ID. All data in the private region is duplicated for extra reliability.
public region	A region of a physical disk managed by VxVM that contains available space and is used for allocating subdisks.
RAID (redundant array of independent disks)	A disk array set up with part of the combined storage capacity used for storing duplicate information about the data stored in that array. This makes it possible to regenerate the data if a disk failure occurs.
read-writeback mode	A recovery mode in which each read operation recovers plex consistency for the region covered by the read. Plex consistency is recovered by reading data from blocks of one plex and writing the data to all other writable plexes.
root configuration	The configuration database for the root disk group. This is special in that it always contains records for other disk groups, which are used for backup purposes only. It also contains disk records that define all disk devices on the system.
root disk	The disk containing the root file system. This disk may be under VxVM control.
root file system	The initial file system mounted as part of the UNIX kernel startup sequence.
root partition	The disk region on which the root file system resides.
root volume	The VxVM volume that contains the root file system, if such a volume is designated by the system configuration.

rootability The ability to place the `root` file system and the `swap` device under VxVM control. The resulting volumes can then be mirrored to provide redundancy and allow recovery in the event of disk failure.

secondary path In Active/Passive disk arrays, the paths to a disk other than the primary path are called secondary paths. A disk is supposed to be accessed only through the primary path until it fails, after which ownership of the disk is transferred to one of the secondary paths.

sector A unit of size, which can vary between systems. Sector size is set per device (hard drive, CD-ROM, and so on). Although all devices within a system are usually configured to the same sector size for interoperability, this is not always the case.

A sector is commonly 512 bytes.

shared disk group A disk group in which access to the disks is shared by multiple hosts (also referred to as a cluster-shareable disk group).

shared volume A volume that belongs to a shared disk group and is open on more than one node of a cluster at the same time.

shared VM disk A VM disk that belongs to a shared disk group in a cluster.

slave node A node that is not designated as the master node of a cluster.

slice The standard division of a logical disk device. The terms partition and slice are sometimes used synonymously.

snapshot A point-in-time copy of a volume (volume snapshot) or a file system (file system snapshot).

spanning A layout technique that permits a volume (and its file system or database) that is too large to fit on a single disk to be configured across multiple physical disks.

sparse plex A plex that is not as long as the volume or that has holes (regions of the plex that do not have a backing subdisk).

SAN (storage area network) A networking paradigm that provides easily reconfigurable connectivity between any subset of computers, disk storage and interconnecting hardware such as switches, hubs and bridges.

stripe A set of stripe units that occupy the same positions across a series of columns.

stripe size The sum of the stripe unit sizes comprising a single stripe across all columns being striped.

stripe unit Equally-sized areas that are allocated alternately on the subdisks (within columns) of each striped plex. In an array, this is a set of logically contiguous blocks that exist on each disk before allocations are made from the next disk in the array. A stripe unit may also be referred to as a stripe element.

stripe unit size The size of each stripe unit. The default stripe unit size is 64KB. The stripe unit size is sometimes also referred to as the stripe width.

striping A layout technique that spreads data across several physical disks using stripes. The data is allocated alternately to the stripes within the subdisks of each plex.

subdisk A consecutive set of contiguous disk blocks that form a logical disk segment. Subdisks can be associated with plexes to form volumes.

swap area A disk region used to hold copies of memory pages swapped out by the system pager process.

swap volume A VxVM volume that is configured for use as a swap area.

transaction A set of configuration changes that succeed or fail as a group, rather than individually. Transactions are used internally to maintain consistent configurations.

VM disk A disk that is both under VxVM control and assigned to a disk group. VM disks are sometimes referred to as VxVM disks.

volboot file A small file that is used to locate copies of the boot disk group configuration. The file may list disks that contain configuration copies in standard locations, and can also contain direct pointers to configuration copy locations. The `volboot` file is stored in a system-dependent location.

volume A virtual disk, representing an addressable range of disk blocks used by applications such as file systems or databases. A volume is a collection of from one to 32 plexes.

volume configuration device The volume configuration device (`/dev/vx/config`) is the interface through which all configuration changes to the volume device driver are performed.

volume device driver The driver that forms the virtual disk drive between the application and the physical device driver level. The volume device driver is accessed through a virtual disk device node whose character device nodes appear in `/dev/vx/rdsk`, and whose block device nodes appear in `/dev/vx/dsk`.

vxconfigd The VxVM configuration daemon, which is responsible for making changes to the VxVM configuration. This daemon must be running before VxVM operations can be performed.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z