VERITAS Volume Manager™ 3.5

Administrator's Guide

Solaris

August 2002
N08836F

VERITAS

Disclaimer

The information contained in this publication is subject to change without notice. VERITAS Software Corporation makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. VERITAS Software Corporation shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this manual.

Copyright

VERITAS Software Corporation
350 Ellis Street
Mountain View, CA 94043
Phone 650–527–8000
Fax 650-527-2908
www.veritas.com

Contents

Preface

Introduction

The *VERITAS Volume Manager™ Administrator's Guide* provides information on how to use VERITAS Volume Manager (VxVM) and all of its features.

Audience

This guide is intended for system administrators responsible for installing, configuring, and maintaining systems under the control of VxVM.

This guide assumes that the user has a:

- working knowledge of the UNIX operating system
- basic understanding of UNIX system administration
- basic understanding of volume management

Scope

The purpose of this guide is to provide the system administrator with a thorough knowledge of the procedures and concepts involved with volume management and system administration using VxVM. This guide includes guidelines on how to take advantage of various advanced VxVM features, and instructions on how to use VxVM commands to create and manipulate objects in VxVM.

Organization

This guide is organized as follows:

- Understanding VERITAS Volume Manager
- Administering Disks
- Creating and Administering Disk Groups
- Creating and Administering Subdisks
- Creating and Administering Plexes
- Creating Volumes
- Administering Volumes
- Administering Hot-Relocation
- Administering Dynamic Multipathing (DMP)
- Administering Cluster Functionality
- Configuring Off-Host Processing
- Performance Monitoring and Tuning

Using This Guide

This guide contains instructions for performing VxVM system administration tasks. VxVM administration functions can be performed through one or more of the following interfaces:

- a set of complex commands
- a single automated command (`vxassist`)
- a menu-driven interface (`vxdiskadm`)
- the VERITAS Enterprise Administrator™ (graphical user interface)

This guide describes how to use the various VxVM command line interfaces for administering VxVM. Details on how to use the VERITAS Enterprise Administrator can be found in the *VERITAS Volume Manager (UNIX) User's Guide — VEA*. Detailed descriptions of the VxVM commands and utilities, their options, and details on their use are located in the VxVM manual pages. Also see "Commands Summary" on page 315 for a listing of the commonly used commands in VxVM together with references to their descriptions.

Note Most VxVM commands require superuser or other appropriate privileges.

Related Documents

The following documents provide information related to VxVM:

- *VERITAS Volume Manager Installation Guide*
- *VERITAS Volume Manager Release Notes*
- *VERITAS Volume Manager Hardware Notes*
- *VERITAS Volume Manager Troubleshooting Guide*
- *VERITAS Volume Manager (UNIX) User's Guide — VEA*
- VERITAS Volume Manager manual pages

Conventions

The following table describes the typographic conventions used in this guide.

Typeface	Usage	Examples
`monospace`	Computer output, file contents, files, directories, software elements such as command options, function names, and parameters	Read tunables from the `/etc/vx/tunefstab` file. See the `ls`(1) manual page for more information.
italic	New terms, book titles, emphasis, variables to be replaced by a name or value	See the *User's Guide* for details. The variable *ncsize* determines the value of...
`monospace` (**`bold`**)	User input; the "#" symbol indicates a command prompt	# **`mount -F vxfs /h/filesys`**
`monospace` (***bold and italic***)	Variables to be replaced by a name or value in user input	# **`mount -F`** ***`fstype mount_point`***

Symbol	Usage	Examples
%	C shell prompt	
$	Bourne / Korn / Bash shell prompt	
#	Superuser prompt (all shells)	
\	Continued input on the following line	# **`mount -F vxfs`** \ **`/h/filesys`**
[]	In a command synopsis, brackets indicates an optional argument	`ls [ -a ]`
|	In a command synopsis, a vertical bar separates mutually exclusive arguments	`mount [suid | nosuid ]`

Getting Help

If you have any comments or problems with VERITAS products, contact VERITAS Technical Support:

- U.S. and Canadian Customers: 1-800-342-0652
- International Customers: +1 (650) 527-8555
- Email: `support@veritas.com`

For license information (U.S. and Canadian Customers):

- Phone: 1-925-931-2464
- Email: license@veritas.com
- Fax: 1-925-931-2487

For software updates:

- Email: swupdate@veritas.com

For information on purchasing VERITAS products:

- Phone: 1-800-258-UNIX (1-800-258-8649) or 1-650-527-8000
- Email: vx-sales@veritas.com

For additional technical support information, such as TechNotes, product alerts, and hardware compatibility lists, visit the VERITAS Technical Support Web site at:

- http://support.veritas.com

For additional information about VERITAS and VERITAS products, visit the Web site at:

- http://www.veritas.com

Using VRTSexplorer

The `VRTSexplorer` program can help VERITAS Technical Support engineers diagnose the cause of technical problems associated with VERITAS products. You can download this program from the VERITAS FTP site or install it from the VERITAS Installation CD. For more information, consult the *VERITAS Volume Manager Release Notes* and the README file in the `support` directory on the VERITAS Installation CD.

Understanding VERITAS Volume Manager 1

Introduction

VERITAS Volume Manager (VxVM) is a storage management subsystem that allows you to manage physical disks as logical devices called *volumes*. A volume is a logical device that appears to data management systems as a physical disk partition device.

VxVM provides easy-to-use online disk storage management for computing environments and Storage Area Network (SAN) environments. Through support of Redundant Array of Independent Disks (RAID), VxVM protects against disk and hardware failure. Additionally, VxVM provides features that enable fault tolerance and fast recovery from disk failure.

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

VxVM provides the tools to improve performance and ensure data availability and integrity. VxVM also dynamically configures disk storage while the system is active.

The following sections of this chapter explain fundamental concepts of VxVM:

- How VxVM Handles Storage Management
- Physical Objects—Physical Disks
- Virtual Objects
- Volume Layouts in VxVM

The following sections introduce you to advanced features of VxVM:

- Online Relayout
- Volume Resynchronization
- Dirty Region Logging (DRL)
- Volume Snapshots
- FastResync
- SmartSync Recovery Accelerator
- Hot-Relocation

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 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 multipathing (DMP) metadevice

This guide introduces you to the VxVM commands which are used to carry out the tasks associated with VxVM objects. These commands are described on the relevant manual pages and in the chapters of this guide when VxVM tasks are described.

VxVM relies on the following constantly running daemons 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`–The VxVM I/O daemon provides extended I/O operations without blocking calling processes. Several `vxiod` daemons are usually started at boot time, and 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.

How Data is Stored

There are several methods 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.

Hard disks are formatted, and information stored, using two methods: physical-storage layout and logical-storage layout. VxVM uses the *logical-storage layout* method. The types of storage layout supported by VxVM are introduced in this chapter.

How VxVM Handles Storage Management

VxVM uses two types of *objects* to handle storage management: *physical objects* and *virtual objects*.

- Physical objects—Physical disks or other hardware with block and raw operating system device interfaces that are used to store data.
- 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.

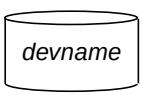
Physical Objects—Physical Disks

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 `c#t#d#s#`, where:

- `c#` specifies the controller
- `t#` specifies the target ID
- `d#` specifies the disk
- s# specifies the partition or slice

The figure, "Physical Disk Example", shows how a physical disk and device name (*devname*) are illustrated in this document. For example, device name `c0t0d0s2` is the entire hard disk connected to controller number `0` in the system, with a target ID of `0`, and physical disk number `0`.

Physical Disk Example

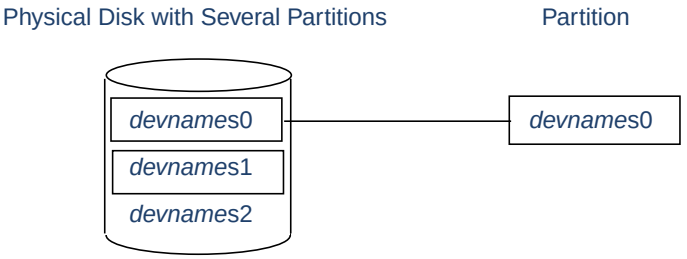


VxVM writes identification information on physical disks under VxVM control (VM disks). VxVM disks can be identified even after physical disk disconnection or system outages. VxVM can then re-form disk groups and logical objects to provide failure detection and to speed system recovery.

Partitions

A physical disk can be divided into one or more *partitions*, also known as *slices*. The *partition number* is added at the end of the devname, and is denoted by `s#`. Note that partition s2 refers to an entire physical disk. See the partition shown in "Partition Example."

Partition Example



Disk Arrays

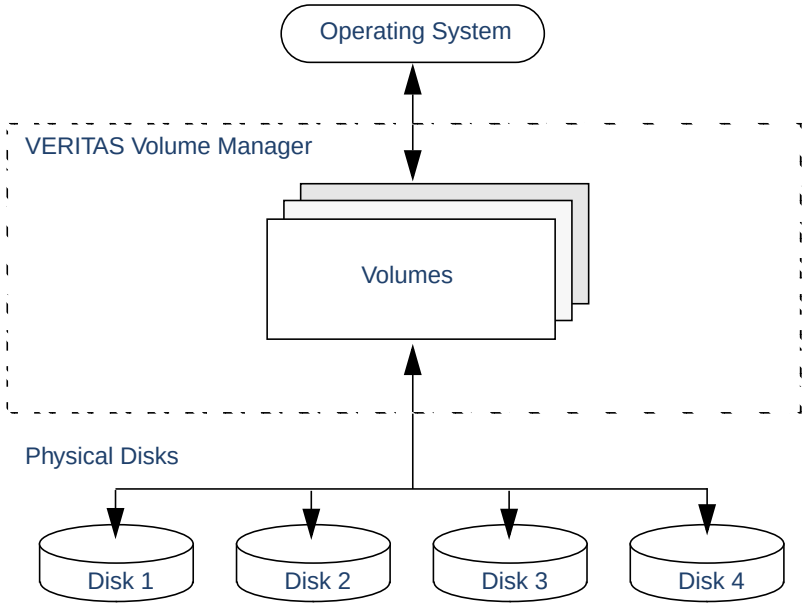
Performing I/O to disks is a relatively slow process because disks are physical devices that require time to move the heads to the correct position on the disk before reading or writing. If all of the read or write operations are done to individual disks, one at a time, the read-write time can become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

"How VxVM Presents the Disks in a Disk Array as Volumes to the Operating System" illustrates how VxVM represents the disks in a disk array as several volumes to the operating system.

Data can be spread across several disks within an array 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.

How VxVM Presents the Disks in a Disk Array as Volumes to the Operating System



Multipathed 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). For more detailed information, see “Administering Dynamic Multipathing (DMP)” on page 95.

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 services 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.

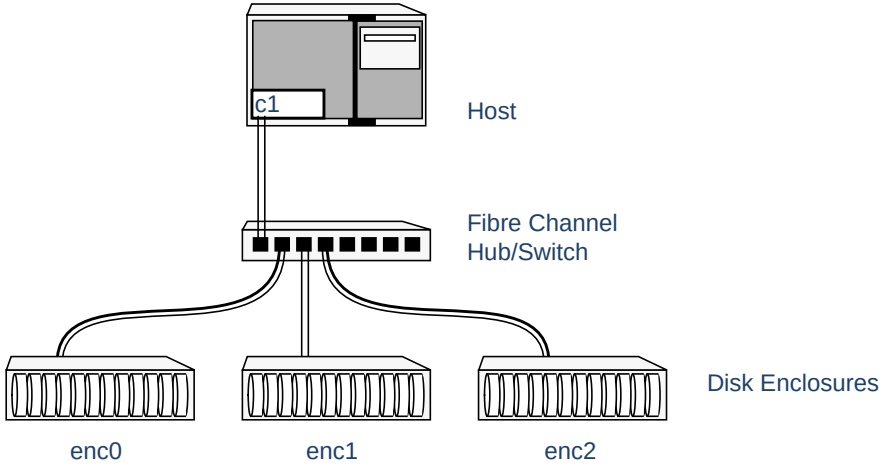
This means that you can dynamically add a new disk array to a host, and run a command which scans the operating system's device tree for all the attached disk devices, and reconfigures DMP with the new device database. For more information, see "Administering the Device Discovery Layer" on page 58.

Enclosure-Based Naming

Enclosure-based naming provides an alternative to the disk device naming described in "Physical Objects—Physical Disks" on page 3. This allows disk devices to be named for enclosures rather than for the controllers through which they are accessed. In a Storage Area Network (SAN) that uses Fibre Channel hubs or fabric switches, information about disk location provided by the operating system may not correctly indicate the physical location of the disks. For example, c#t#d#s# naming assigns controller-based device names to disks in separate enclosures that are connected to the same host controller. 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.

In a typical SAN environment, host controllers are connected to multiple enclosures in a daisy chain or through a Fibre Channel hub or fabric switch as illustrated in"Example Configuration for Disk Enclosures Connected via a Fibre Channel Hub/Switch."

Example Configuration for Disk Enclosures Connected via a Fibre Channel Hub/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.

Note In many advanced disk arrays, you can use hardware-based storage management to represent several physical disks as one logical disk device 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 or a logical device.

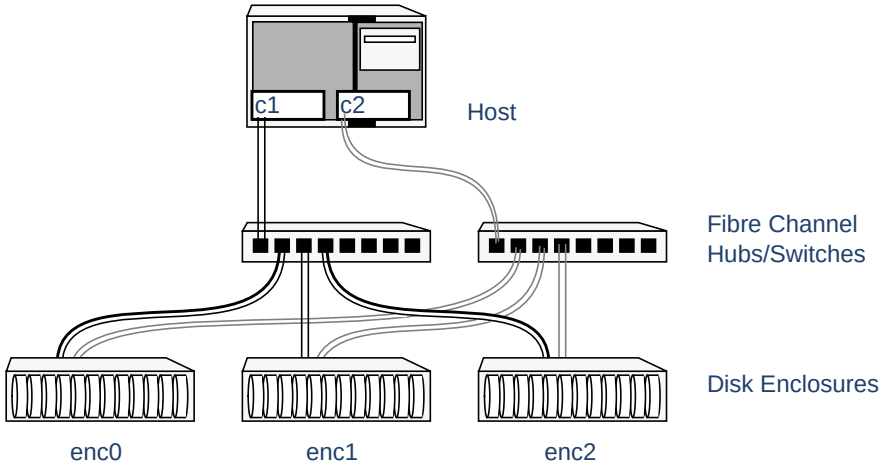
Another important benefit of enclosure-based naming is that it enables VxVM to avoid placing redundant copies of data in the same enclosure. This is a good thing to avoid as each enclosure can be considered to be a separate fault domain. For example, if a mirrored volume were configured only on the disks in enclosure `enc1`, the failure of the cable between the hub 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 115 for details.

In High Availability (HA) configurations, redundant-loop access to storage can be implemented by connecting independent controllers on the host to separate hubs with independent paths to the enclosures as shown in "Example HA Configuration Using Multiple Hubs/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 hubs. 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 `c1t99d0` and `c2t99d0`.

To take account of fault domains when configuring data redundancy, you can control how mirrored volumes are laid out across enclosures as described in "Mirroring across Targets, Controllers or Enclosures" on page 188.

Example HA Configuration Using Multiple Hubs/Switches to Provide Redundant-Loop Access



See “Disk Device Naming in VxVM” on page 54 and “Changing the Disk-Naming Scheme” on page 61 for details of the standard and the enclosure-based naming schemes, and how to switch between them.

Virtual Objects

Virtual objects in VxVM include the following:

- VM Disks
- Disk Groups
- Subdisks
- Plexes
- Volumes

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.

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

Note The `vxprint` command displays detailed information on existing VxVM objects. For additional information on the `vxprint` command, see "Displaying Volume Information" on page 197 and the `vxprint`(1M) manual page.

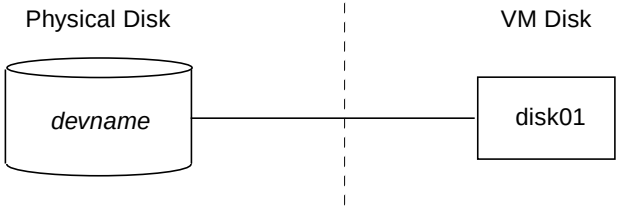
VM Disks

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 *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 typically takes the form `disk##`. "VM Disk Example" shows a VM disk with a media name of `disk01` that is assigned to the physical disk *devname*.

VM Disk Example



Disk Groups

A *disk group* is a collection of VM disks that share a common configuration. A disk group configuration is a set of records with detailed information about related VxVM objects, their attributes, and their connections. The default disk group is `rootdg` (or *root disk group*). A disk group name can be up to 31 characters long.

Note Even though `rootdg` is the default disk group, it does not contain the root disk unless the root disk is encapsulated.

You can create additional disk groups as necessary. 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. The ability to move whole volumes and disks between disk groups, to split whole volumes and disks between disk groups, and to join disk groups is described in "Reorganizing the Contents of Disk Groups" on page 133.

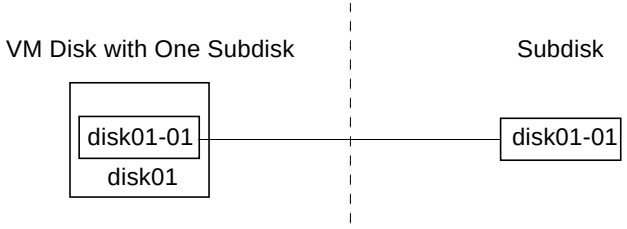
Volumes are created within a disk group. A given volume must be configured from disks in the same disk group.

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.

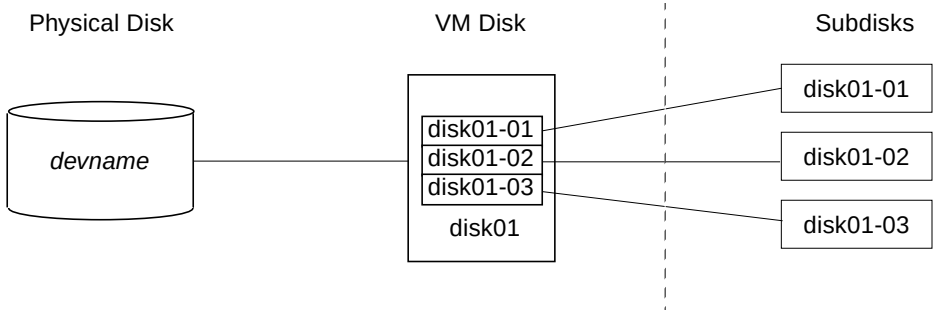
The default name for a VM disk is `disk##` (such as `disk01`) and the default name for a subdisk is `disk##-##`. In the figure, "`Subdisk Example`", `disk01-01` is the name of the first subdisk on the VM disk named `disk01`.

Subdisk Example



A VM disk can contain multiple subdisks, but subdisks cannot overlap or share the same portions of a VM disk. "Example of Three Subdisks Assigned to One VM Disk" shows a VM disk with three subdisks. The VM disk is assigned to one physical disk.

Example of Three Subdisks Assigned to One VM Disk



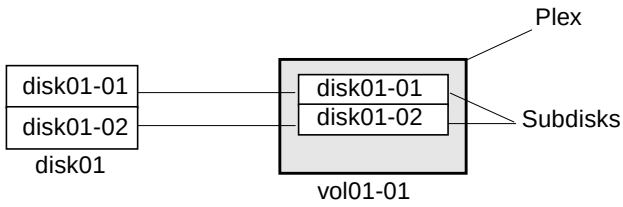
Any VM disk space that is not part of a subdisk is free space. You can use free space to create new subdisks.

VxVM release 3.0 or higher supports the concept of layered volumes in which subdisks can contain volumes. For more information, see "Layered Volumes" on page 29.

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. For example, see the plex `vol01-01` shown in "Example of a Plex with Two Subdisks"

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 described in "Volume Layouts in VxVM" on page 14.

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.

Note 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.

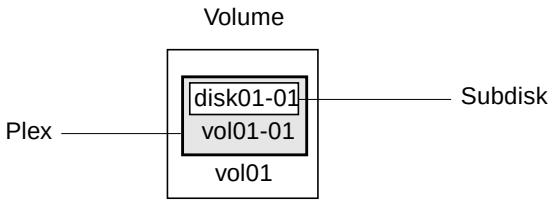
A volume may be created under the following constraints:

- 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.

See “Example of a Volume with One Plex”.

Example of a Volume with One Plex

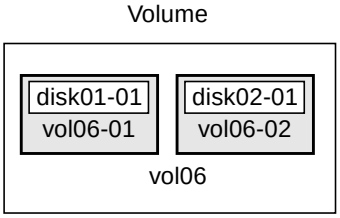


Volume `vol01` has the following characteristics:

- 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`.

A volume with two or more data plexes is “mirrored” and contains mirror images of the data. See “Example of a Volume with Two Plexes”

Example of a Volume with Two Plexes



Each plex contains an identical copy of the volume data. For more information, see “Mirroring (RAID-1)” on page 21.

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`)

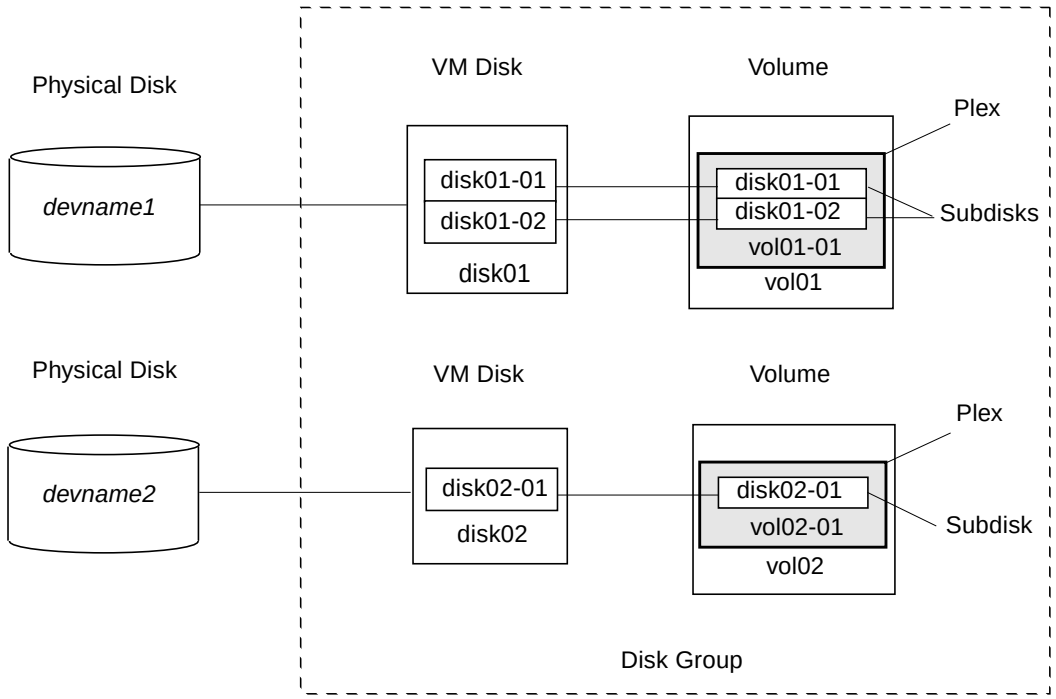
Combining Virtual Objects in VxVM

VxVM virtual objects are combined to build volumes. The virtual objects contained in volumes are VM disks, disk groups, subdisks, and plexes. VERITAS Volume Manager objects are organized as follows:

- 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

The figure, "Connection Between Objects in VxVM", shows the connections between VERITAS Volume Manager virtual objects and how they relate to physical disks. The disk group consists of two VM disks: `disk01` has a volume with one plex and two subdisks, and `disk02` has a volume with one plex and a single subdisk

Connection Between Objects in VxVM



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 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 each provide different levels of storage service. A volume layout can be configured and reconfigured to match particular levels of desired storage service.

Implementation of Non-Layered Volumes

In a *non-layered* volume, a subdisk is restricted to mapping 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.

Implementation of 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. 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 29 for more information.

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 and Spanning
- Striping (RAID-0)
- Mirroring (RAID-1)
- Striping Plus Mirroring (Mirrored-Stripe or RAID-0+1)
- Mirroring Plus Striping (Striped-Mirror, RAID-1+0 or RAID-10)
- RAID-5 (Striping with Parity)

The following sections describe each layout method.

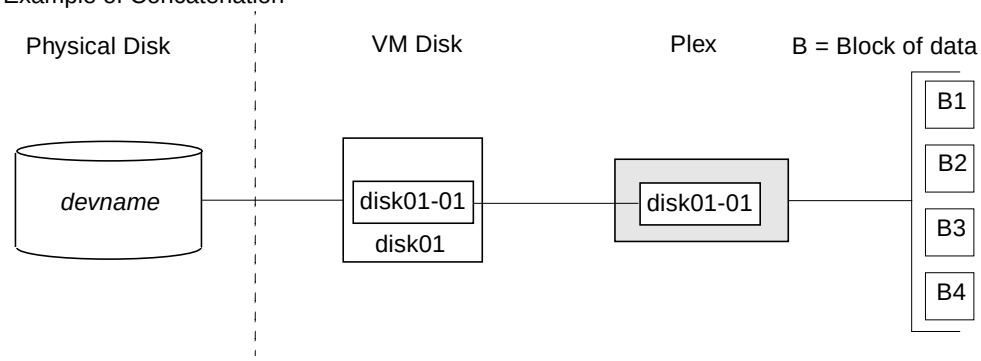
Concatenation and Spanning

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 beginning to end. Data is then accessed in the remaining subdisks sequentially from beginning to end, 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*.

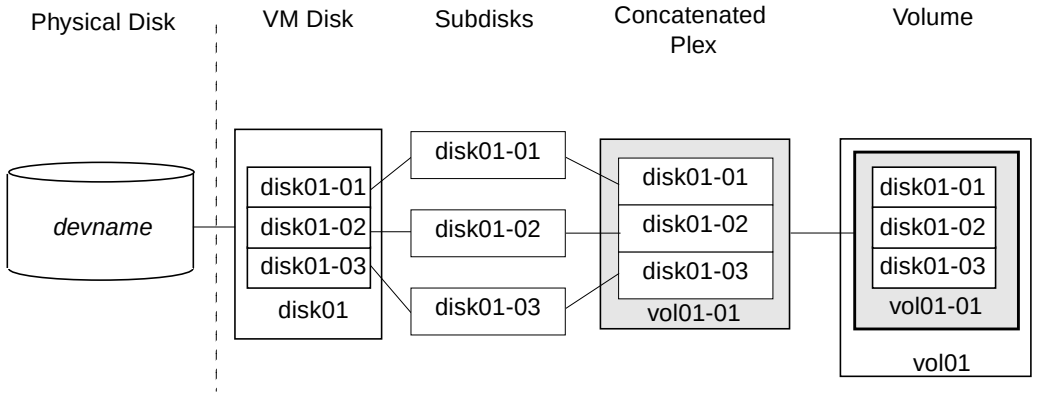
The figure, "Example of Concatenation", shows concatenation with one subdisk.

Example of Concatenation



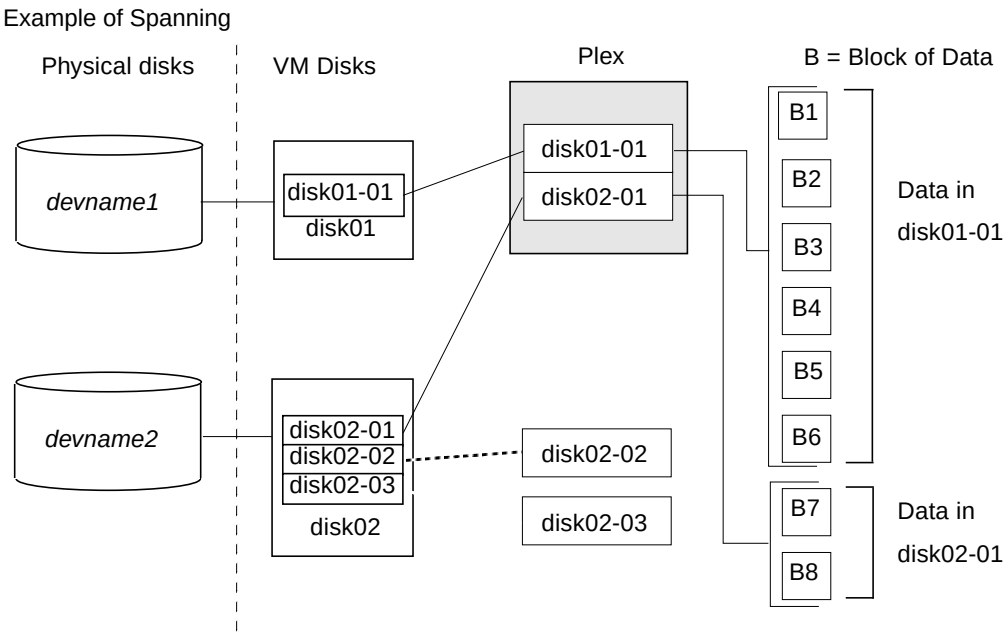
You can use concatenation with multiple subdisks when there is insufficient contiguous space for the plex on any one disk. This form of concatenation can be used for load balancing between disks, and for head movement optimization on a particular disk. See the figure, "Example of a Volume in a Concatenated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

Example of a Volume in a Concatenated Configuration



The figure, "Example of Spanning" on page 17 shows data spread over two subdisks in a spanned plex. In the figure, "Example of Spanning," the first six blocks of data (`B1` through `B6`) use most of the space on the disk to which VM disk `disk01` is assigned. This requires space only on subdisk `disk01-01` on `disk01`. However, the last two blocks of data, `B7` and `B8`, use only a portion of the space on the disk to which VM disk `disk02` is assigned.

The remaining free space on VM disk `disk02` can be put to other uses. In this example, subdisks `disk02-02` and `disk02-03` are available for other disk management tasks.



Caution 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 (both described later) to reduce the risk that a single disk failure results in a volume failure.

See "Creating a Volume on Any Disk" on page 177 for information on how to create a concatenated volume that may span several disks.

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.

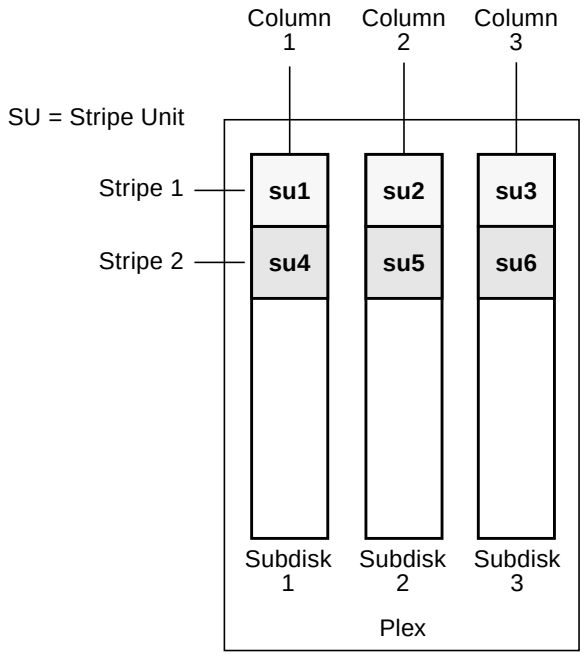
Caution Striping a volume, or splitting a volume across multiple disks, increases the chance that a disk failure will result in failure of that volume.

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 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 units (*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 (or *width*) is 64 kilobytes.

For example, if there are three columns in a striped plex and six stripe units, data is striped over the three columns, as illustrated in "Striping Across Three Columns."

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 1
- stripe unit 2 in column 2

- stripe unit 3 in column 3

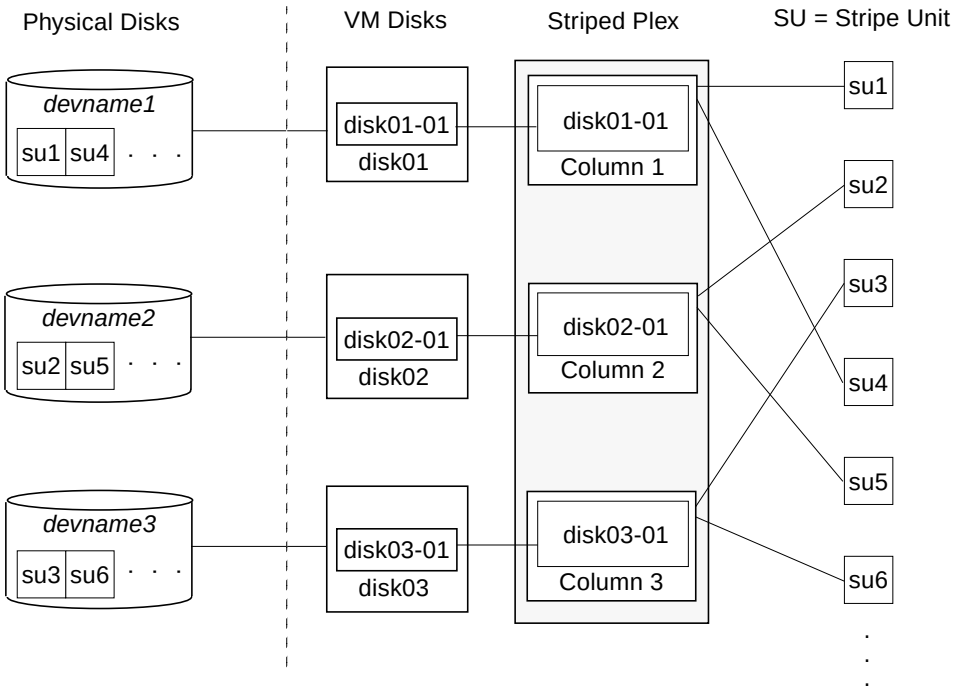
The second stripe consists of:

- stripe unit 4 in column 1
- stripe unit 5 in column 2
- stripe unit 6 in column 3

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.

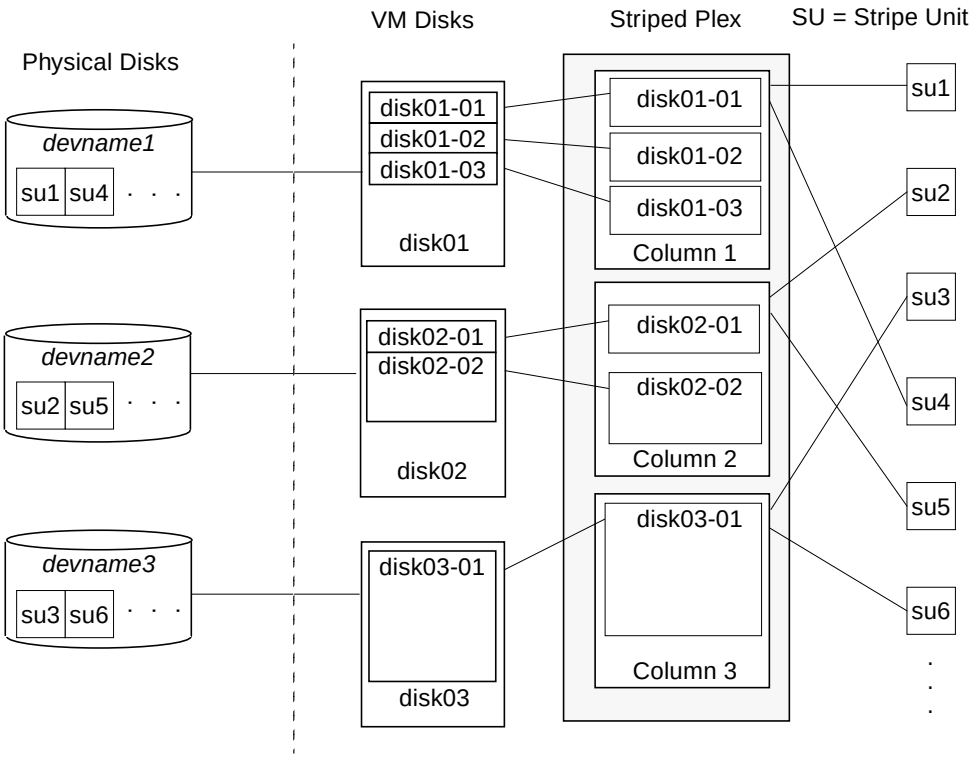
"Example of a Striped Plex with One Subdisk per Column" shows a striped plex with three equal sized, single-subdisk columns. There is one column per physical disk. This example shows three subdisks that occupy all of the space on the VM disks. It is also possible for each subdisk in a striped plex to occupy only a portion of the VM disk, which leaves free space for other disk management tasks.

Example of a Striped Plex with One Subdisk per Column



"Example of a Striped Plex with Concatenated Subdisks per Column" illustrates a striped plex with three columns containing subdisks of different sizes. 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.

Example of a Striped Plex with Concatenated Subdisks per Column



See "Creating a Striped Volume" on page 186 for information on how to create a striped volume.

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.

Note 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 183 for information on how to create a mirrored volume.

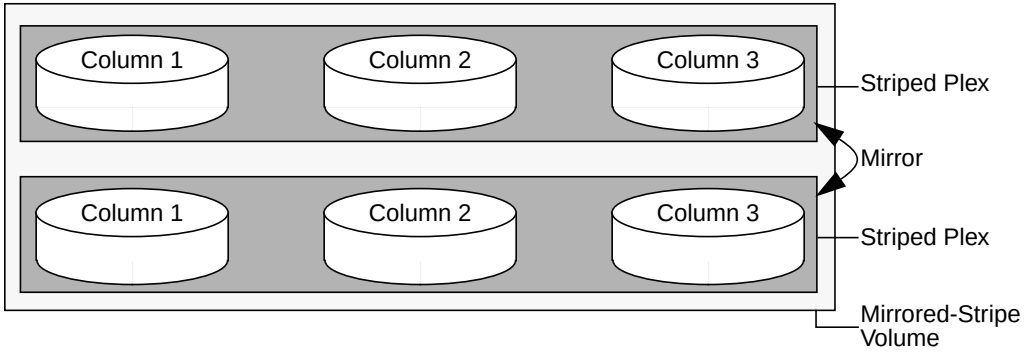
Striping Plus Mirroring (Mirrored-Stripe or RAID-0+1)

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.

The figure, “Mirrored-Stripe Volume Laid out on Six Disks” 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.

Mirrored-Stripe Volume Laid out on Six Disks



See "Creating a Mirrored-Stripe Volume" on page 187 for information on how to create a mirrored-stripe volume.

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)

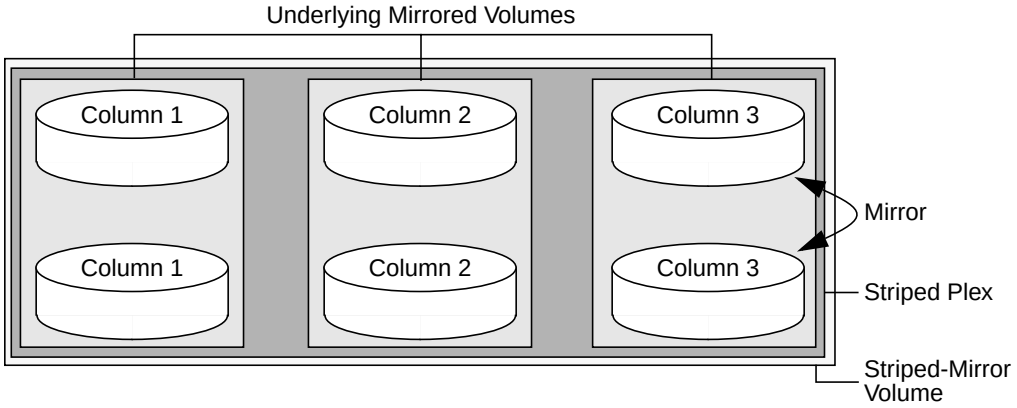
VxVM supports the combination of striping above mirroring. This combined layout is called a *striped-mirror* layout. Putting mirroring below striping mirrors each column of the stripe. If there are multiple subdisks per column, each subdisk can be mirrored individually instead of each column.

Note A striped-mirror volume is an example of a layered volume. See "Layered Volumes" on page 29 for more information.

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.

"Striped-Mirror Volume Laid out on Six Disks" 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.

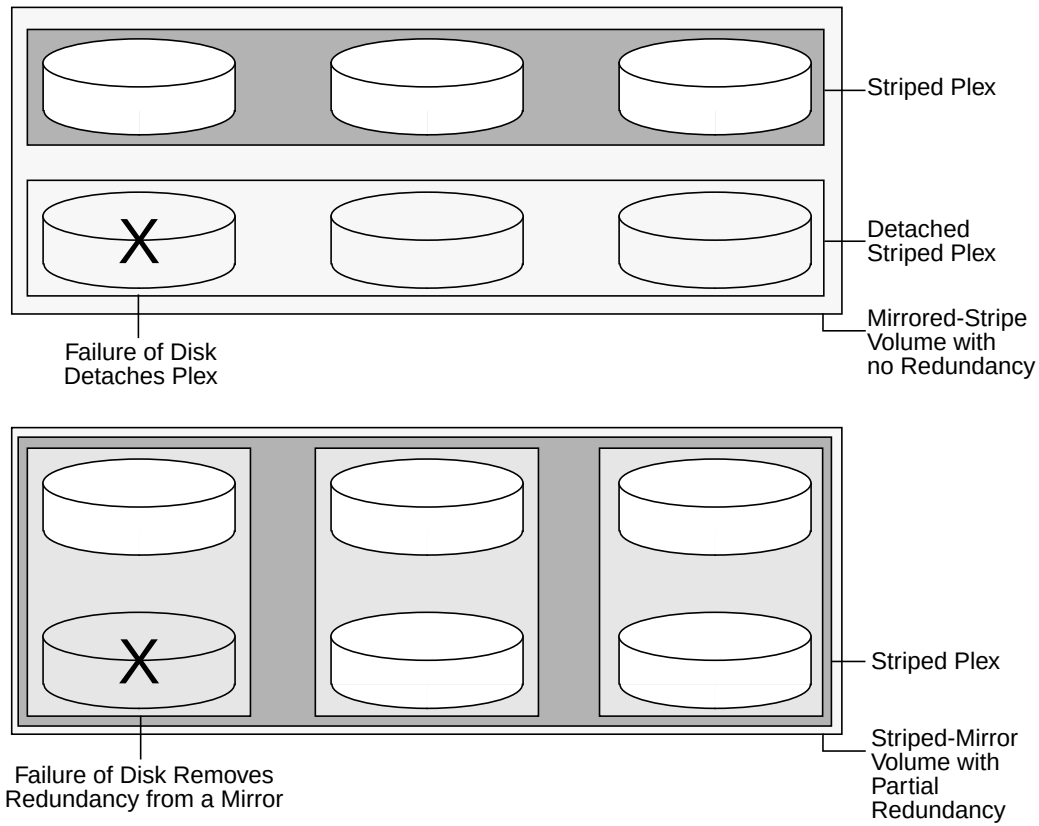
Striped-Mirror Volume Laid out on Six Disks



See "Creating a Striped-Mirror Volume" on page 188 for information on how to create a striped-mirrored volume.

As shown in the figure, "How the Failure of a Single Disk Affects Mirrored-Stripe and Striped-Mirror Volumes," the failure of a disk in a mirrored- stripe layout detaches an entire data plex, thereby losing redundancy on the entire volume. 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.

How the Failure of a Single Disk Affects Mirrored-Stripe and Striped-Mirror Volumes



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.

Note The VERITAS Enterprise Administrator (VEA) terms a striped-mirror as `Striped-Pro`, and a concatenated-mirror as `Concatenated-Pro`.

RAID-5 (Striping with Parity)

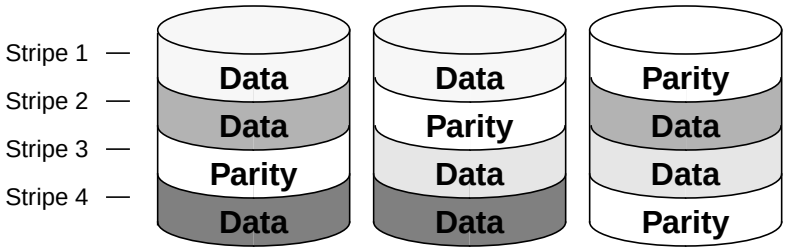
Note VxVM supports RAID-5 for private disk groups, but not for shareable disk groups in a cluster environment.

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.

The figure, "Parity Locations in a RAID-5 Model", shows parity locations in a RAID-5 array configuration. 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.

Parity Locations in a RAID-5 Model



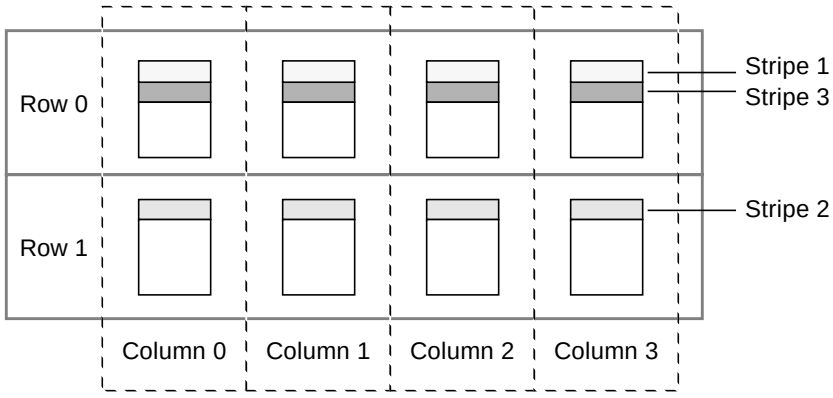
RAID-5 and how it is implemented by the VxVM is described in "*VERITAS Volume Manager RAID-5 Arrays*" on page 25.

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.

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. The figure, "Traditional RAID-5 Array", shows the row and column arrangement of a traditional RAID-5 array.

Traditional RAID-5 Array



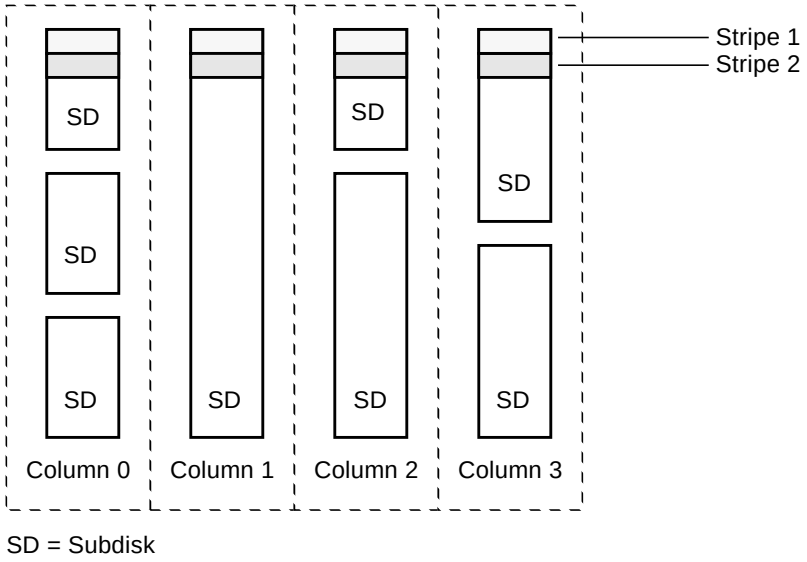
This traditional array structure supports growth by adding more rows per column. Striping is accomplished by applying the first stripe across the disks in Row 0, then the second stripe across the disks in Row 1, then the third stripe across the Row 0 disks, and so on. This type of array requires all 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. Instead, VxVM uses columns consisting of variable length subdisks (as shown in "VERITAS Volume ManagerRAID-5 Array" on page 26). Each subdisk represents a specific area of a disk.

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 17 for further information about stripe units.

VERITAS Volume ManagerRAID-5 Array



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

See “Creating a RAID-5 Volume” on page 189 for information on how to create a RAID-5 volume.

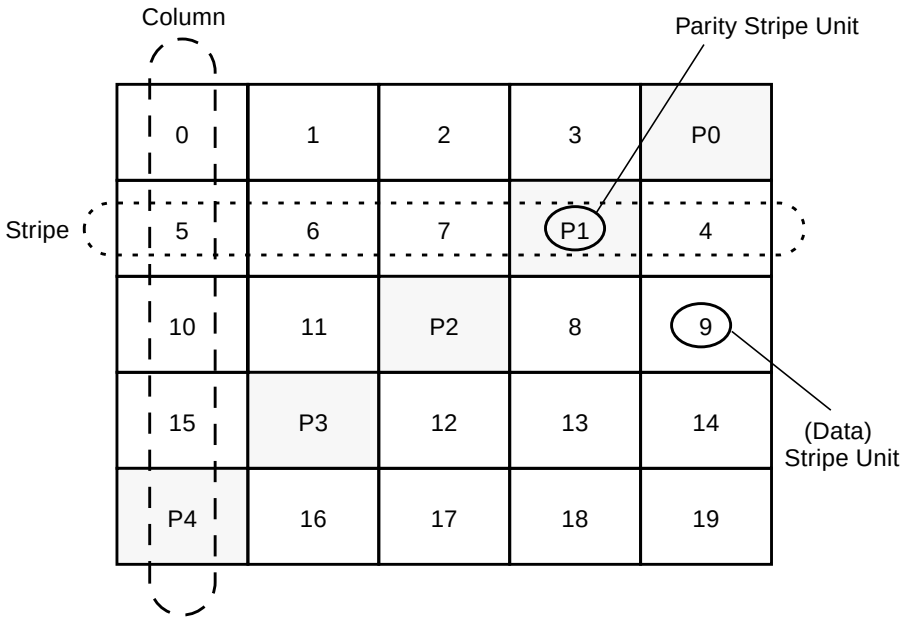
Left-Symmetric Layout

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.

The figure, "Left-Symmetric Layout," shows a left-symmetric parity layout with five disks (one per column).

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.

Note 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.

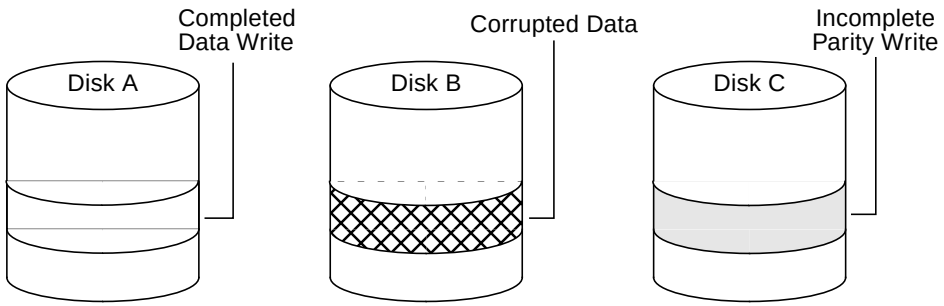
RAID-5 Logging

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.

The figure, "Incomplete Write," illustrates a RAID-5 volume configured across three disks (A, B and C). 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.

Incomplete Write



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 212 for information on how to add a RAID-5 log to a RAID-5 volume.

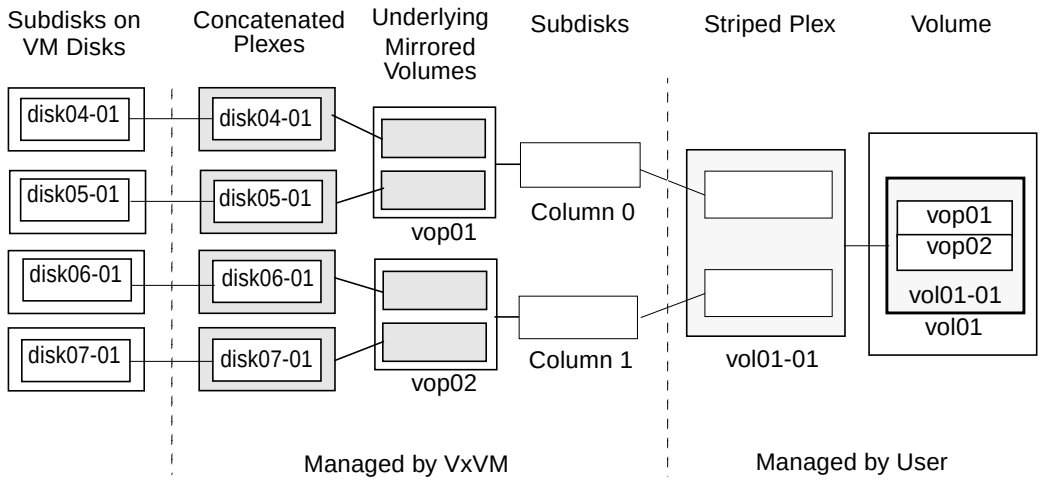
Layered Volumes

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.

The figure, "Example of a Striped-Mirror Layered Volume," illustrates the structure of a typical layered volume. It shows subdisks with two columns, built on underlying volumes with each volume internally mirrored. 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).

Example of a Striped-Mirror Layered Volume



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 188, and the `vxassist`(1M) manual page.)
- Creating concatenated-mirrors. (See "Creating a Concatenated-Mirror Volume" on page 184, and the `vxassist`(1M) manual page.)
- Online Relayout. (See "Online Relayout" on page 30, and the `vxrelayout`(1M) and `vxassist`(1M) manual pages.)
- RAID-5 subdisk moves. (See the `vxsd`(1M) manual page.)
- Snapshots. (See "Backing Up Volumes Online Using Snapshots" on page 228, and the `vxassist`(1M) manual page.)

Note The VERITAS Enterprise Administrator (VEA) terms a striped-mirror as `Striped-Pro`, and a concatenated-mirror as `Concatenated-Pro`.

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 235 for details of how to perform online relayout of volumes in VxVM. Also see "Converting Between Layered and Non-Layered Volumes" on page 238 for information about the additional volume conversion operations that are possible.

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.

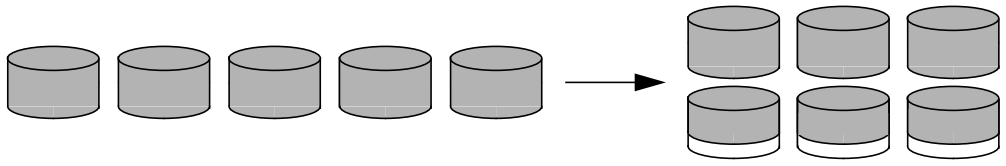
The following error message displays the number of blocks required if there is insufficient free space available in the disk group for the temporary area:

```
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 for more information.

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. The figure, "Example of Decreasing the Number of Columns in a Volume," shows how decreasing the number of columns can require disks to be added to a volume. The size of the volume remains the same but an extra disk is needed to extend one of the columns.

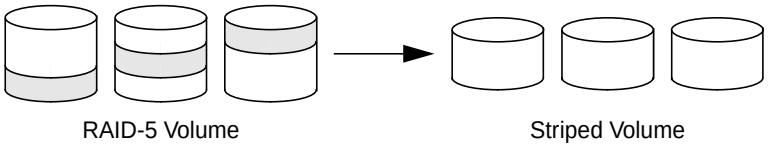
Example of Decreasing the Number of Columns in a Volume



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

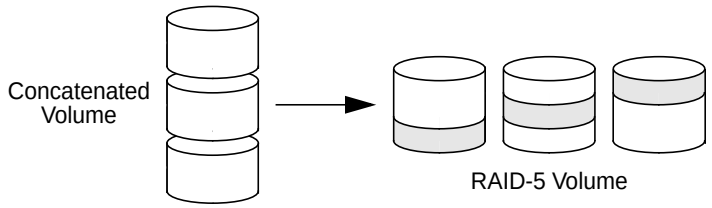
- Change a RAID-5 volume to a concatenated, striped, or layered volume (remove parity). See "Example of Relayout of a RAID-5 Volume to a Striped Volume" below. Note that removing parity (shown by the shaded area) decreases the overall storage space that the volume requires.

  Example of Relayout of a RAID-5 Volume to a Striped Volume

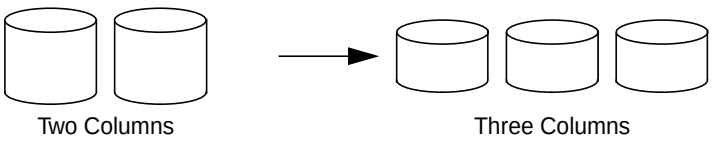
- Change a volume to a RAID-5 volume (add parity). See "Example of Relayout of a Concatenated Volume to a RAID-5 Volume" below. Note that adding parity (shown by the shaded area) increases the overall storage space that the volume requires.

  Example of Relayout of a Concatenated Volume to a RAID-5 Volume

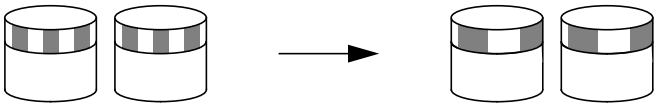
- Change the number of columns in a volume. See "Example of Increasing the Number of Columns in a Volume" below. Note that the length of the columns is reduced to conserve the size of the volume.

  Example of Increasing the Number of Columns in a Volume

- Change the column stripe width in a volume. See "Example of Increasing the Stripe Width for the Columns in a Volume" below.

  Example of Increasing the Stripe Width for the Columns in a Volume

For details of how to perform online relayout operations, see "Performing Online Relayout" on page 235.

Permitted Relayout Transformations

The tables below give details of the relayout operations that are possible for each type of source storage layout.

Supported Relayout Transformations for Unmirrored Concatenated Volumes

Relayout to	From concat
concat	No.
concat-mirror	No. Add a mirror, and then use `vxassist convert` instead.
mirror-concat	No. Add a mirror instead.
mirror-stripe	No. Use `vxassist convert` after relayout to 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.

Supported Relayout Transformations for Layered Concatenated-Mirror Volumes

Relayout to	From concat-mirror
concat	No. Use `vxassist convert`, and then remove 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 striped-mirror volume instead.
raid5	Yes.
stripe	Yes. This removes a mirror and adds striping. The stripe width and number of columns may be defined.
striped-mirror	Yes. The stripe width and number of columns may be defined.

Supported Relayout Transformations for RAID-5 Volumes

Relayout to	From raid5
concat	Yes.
concat-mirror	Yes.
mirror-concat	No. Use `vxassist convert` after relayout to concatenated-mirror volume instead.
mirror-stripe	No. Use `vxassist convert` after relayout to striped-mirror volume instead.
raid5	Yes. The stripe width and number of columns may be changed.
stripe	Yes. The stripe width and number of columns may also be changed.
stripe-mirror	Yes. The stripe width and number of columns may also be changed.

Supported Relayout Transformations for Mirrored-Concatenated Volumes

Relayout to	From mirror-concat
concat	No. Remove unwanted mirrors instead.
concat-mirror	No. Use `vxassist convert` instead.
mirror-concat	No.
mirror-stripe	No. Use `vxassist convert` after relayout to 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.

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 concatenated-mirror volume instead.
mirror-stripe	No. Use `vxassist convert` after relayout to 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`.

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 concatenated-mirror volume instead.
mirror-stripe	No. Use `vxassist convert` after relayout to 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.

Transformations are not supported for the following objects:

- Log plexes.
- Volume snapshot when there is an online relayout operation running on the volume.

Also note the following limitations:

- 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 238 for more information.
- Online relayout can be used only with volumes that have been created using the `vxassist` command or the VERITAS Enterprise Administrator (VEA).
- 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.
- Online relayout involving RAID-5 volumes is not supported for shareable disk groups in a cluster environment.
- Online relayout cannot transform sparse plexes, nor can it make any plex sparse. (A sparse plex is not the same size as the volume, or has regions that are not mapped to any subdisk.)

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. Any existing transformation in the volume must be stopped before doing a reversal.

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(1M)` command to shrink or grow a file system as described in “Resizing a Volume” on page 214.

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 needs to ensure 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 disk groups that are automatically imported at boot time (such as `rootdg`), the resynchronization process 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 when the system reboots require resynchronization.

Resynchronization Process

The process of resynchronization depends on the type of volume. 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. 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.

The process of 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 addressed by using dirty region logging (DRL) and FastResync (fast mirror resynchronization) for mirrored volumes, or by ensuring that RAID-5 volumes have valid RAID-5 logs. See the following sections, "Dirty Region Logging (DRL)" and "FastResync," for more information.

For raw volumes used by database applications, the SmartSync™ Recovery Accelerator can be used (see "SmartSync Recovery Accelerator" on page 47).

Dirty Region Logging (DRL)

Dirty region logging (DRL), if enabled, speeds recovery of mirrored volumes after a system crash. DRL keeps track of 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 that need to be recovered.

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. It may also be necessary to recover the areas of volumes that are already consistent.

Dirty Region Logs

DRL logically divides a volume into a set of consecutive regions, and maintains a dirty region log on disk where each region is represented by one status bit. Before any data is written to any region, DRL synchronously marks the corresponding bit in the log as dirty if it was previously clean. The log is only used to represent regions of the volume on which writes are pending. Once a write has been completed, the dirty bit for a region is not cleared immediately. If another write to the same region occurs, this means it is not necessary to write the log to the disk before the write operation can occur. The bit remains marked as dirty until the corresponding volume region becomes the least recently accessed for writing.

On restarting a system after a crash, VxVM recovers only those regions of the volume that are marked as dirty in the dirty region log.

Log subdisks

Log subdisks are used to 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 should a crash occur. 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 the tunable `voldrl_max_seq_dirty` as described in the description of "voldrl_max_seq_dirty" on page 311.

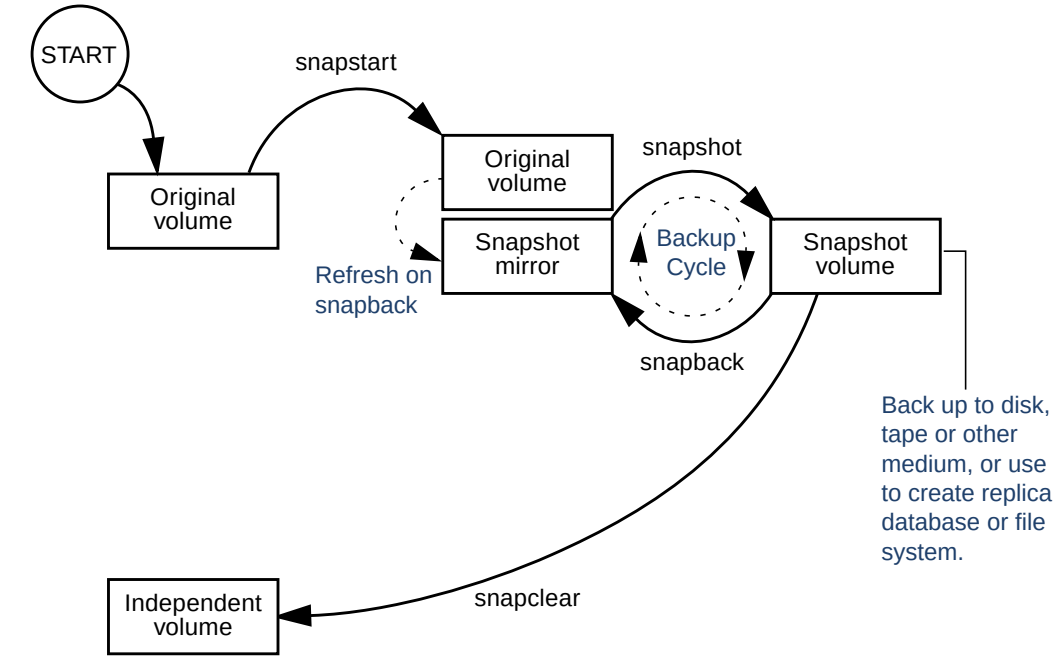
Note DRL adds a small I/O overhead for most write access patterns.

For details of how to configure DRL and sequential DRL, see "Adding DRL Logging to a Mirrored Volume" on page 211.

Volume Snapshots

The volume snapshot model is shown in "Snapshot Creation and the Backup Cycle." This figure also shows the transitions that are supported by the `snapback` and `snapclear` commands to `vxassist`.

Snapshot Creation and the Backup Cycle



The `vxassist snapstart` command creates a mirror to be used for the snapshot, and attaches it to the volume as a snapshot mirror. (The `vxassist snapabort` command can be used to cancel this operation and remove the snapshot mirror.)

When the attachment is complete, the `vxassist snapshot` command is used to create a new snapshot volume by taking one or more snapshot mirrors to use as its data plexes. The snapshot volume contains a copy of the original volume's data at the time that you took the snapshot. If more than one snapshot mirror is used, the snapshot volume is itself mirrored.

The command, `vxassist snapback`, can be used to return snapshot plexes to the original volume from which they were snapped, and to resynchronize the data in the snapshot mirrors from the data in the original volume. This enables you to refresh the data in a snapshot after each time that you use it to make a backup. As described in "FastResync" on page 41, you can use the FastResync feature of VxVM to minimize the time needed to resynchronize the data in the snapshot mirror. If FastResync is not enabled, a full resynchronization of the data is required.

Alternatively, you can use the `vxassist snapclear` command to break the association between the original volume and the snapshot volume. The snapshot volume then has an existence that is independent of the original volume. This is useful for applications that do not require the snapshot to be resynchronized with the original volume.

For more information about taking snapshots of a volume, see "Backing Up Volumes Online Using Snapshots" on page 228, and the `vxassist`(1M) manual page.

FastResync

Note You may need an additional 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 clustered VxVM 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.

For details of how to enable FastResync on a per-volume basis, see "Enabling FastResync on a Volume" on page 221.

FastResync Enhancements

FastResync provides two fundamental enhancements to VxVM:

- 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.

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

Non-Persistent FastResync

Non-Persistent FastResync, introduced in VxVM 3.1, allocates its change maps in memory. If Non-Persistent FastResync is enabled, a separate FastResync map is kept for the original volume and for each snapshot volume. Unlike a dirty region log (DRL), 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.

Persistent FastResync

In VxVM 3.2, Non-Persistent FastResync was augmented by the introduction of Persistent FastResync. Unlike Non-Persistent FastResync, Persistent FastResync keeps the FastResync maps on disk so that they can survive both system reboots, system crashes and cluster crashes. 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.

The DCO object manages information about the FastResync maps. These maps track writes to the original volume and to each of up to 32 snapshot volumes since the last `snapshot` operation. The DCO volume on disk holds the 33 maps, each of which is 4 blocks in size by default.

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 133 for details.

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 shapshot 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.

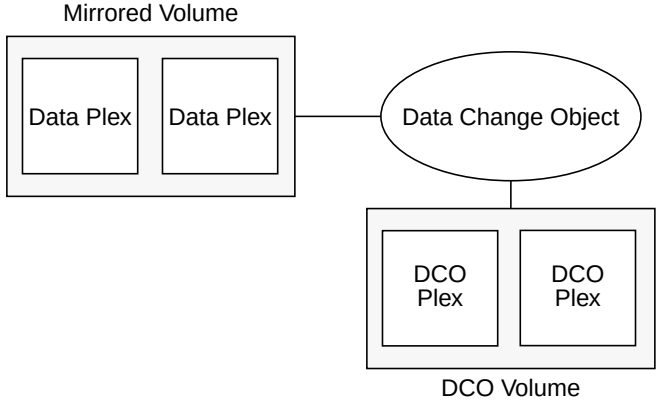
Non-Persistent FastResync uses a map in memory to implement change tracking. Each bit in the map represents a contiguous number of blocks in a volume's address space. The default size of the map is 4 blocks. The kernel tunable `vol_fmr_logsz` can be used to limit the maximum size in blocks of the map as described on page 308.

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. The default size of the map is 1 block. This can be increased 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 should 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.

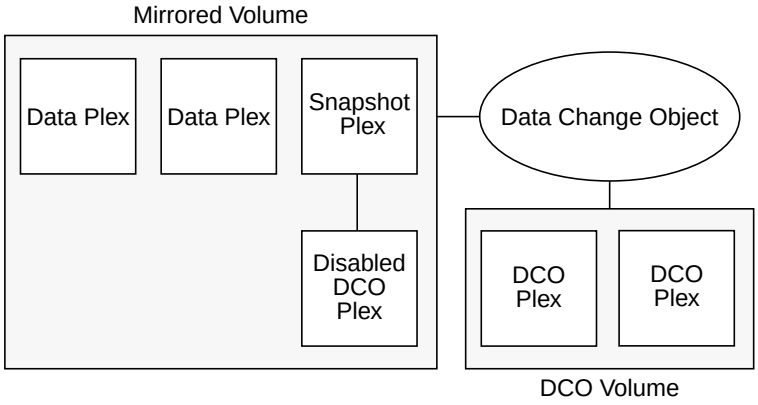
"Mirrored Volume with Persistent FastResync Enabled" shows an example of a mirrored volume with two plexes on which Persistent FastResync is enabled. Associated with the volume are a DCO object and a DCO volume with two plexes.

Mirrored Volume with Persistent FastResync Enabled



When the `snapstart` operation is performed on the volume, this sets up a snapshot plex in the volume and associates a disabled DCO plex with it, as shown in "Mirrored Volume After Completion of a snapstart Operation."

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 `snapstart` operation. You can create up to a total of 32 plexes (data and log) in a volume.

A snapshot volume can now be created from a snapshot plex by running the `snapshot` operation on the volume. As illustrated in "Mirrored Volume and Snapshot Volume After Completion of a snapshot Operation" on page 45, this also sets up a DCO object and a DCO volume for the snapshot volume. The new 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.

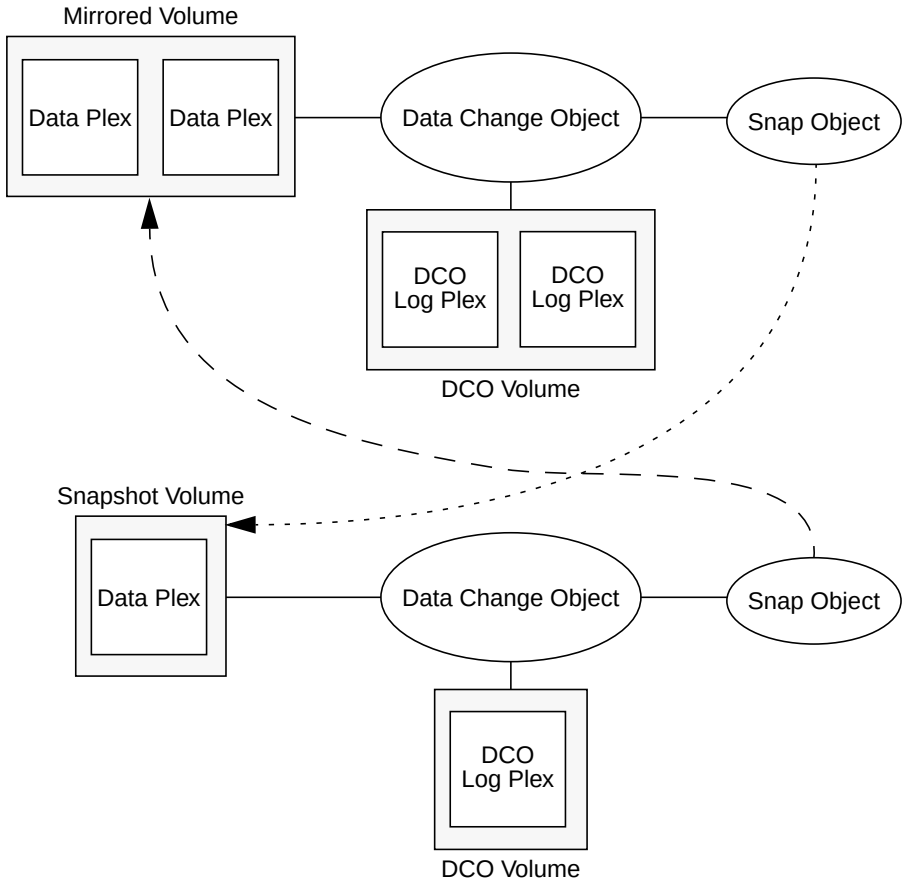
Associated with both the original volume and the snapshot volume are *snap objects*. The snap object for the original volume points to the snapshot volume, and the snap object for the snapshot volume points to the original volume. This allows VxVM to track the relationship between volumes and their snapshots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

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

- The `snapback` operation is run to return all of the plexes of the snapshot volume to the original volume.
- The `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, `snapclear` must be run separately on each volume.

See “Merging a Snapshot Volume (snapback)” on page 232, “Dissociating a Snapshot Volume (snapclear)” on page 233, and the vxassist(1M) manual page for more information.

Mirrored Volume and Snapshot Volume After Completion of a snapshot Operation



Additional Snapshot Features

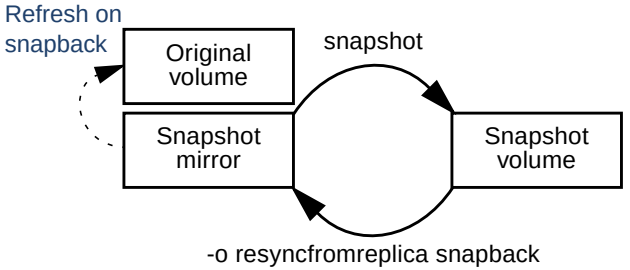
To make it easier to create snapshots of several volumes at the same time, the snapshot option accepts more than one volume name as its argument. By default, each replica volume is named SNAP*number*-*volume*, where number is a unique serial number, and *volume* is the name of the volume being snapshotted. This default can be overridden by using the option -o name=*pattern*, as described on the vxassist(1M) manual page.

To snapshot all the volumes in a single disk group, specify the option `-o allvols` to `vxassist`. However, this fails if any of the volumes in the disk group do not have a complete snapshot plex.

It is also possible to take several snapshots of the same volume. A new FastResync change map is produced for each snapshot taken to minimize the resynchronization time for each snapshot.

By default, the snapshot plex is resynchronized from the data in the original volume during a `snapback` operation. Alternatively, you can choose the snapshot plex as the preferred copy of the data when performing a snapback as illustrated in "Resynchronizing the Original Volume from the Snapshot." Specifying the option `-o resyncfromreplica` to `vxassist` resynchronizes the original volume from the data in the snapshot.

Resynchronizing the Original Volume from the Snapshot



Note The original volume must not be in use during a `snapback` operation that specifies the option `-o resyncfromreplica` to resynchronize the volume from a snapshot. Stop any application, such as a database, and unmount any file systems that are configured to use the volume.

It is possible to grow the replica volume, or the original volume, and still use FastResync. Growing the volume extends the map that FastResync uses to track changes to the original volume. This can change either the size of the map, or the size of the region represented by each bit in the map. 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 `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.
- 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 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.
- 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. For more information, see the `vxvol` (1M), `vxassist` (1M), and `vxplex` (1M) manual pages.

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. The SmartSync feature uses an extended interface between VxVM volumes and the database software to avoid unnecessary work during mirror resynchronization. For example, Oracle® automatically takes advantage of SmartSync to perform database resynchronization when it is available.

Note SmartSync is only applicable to databases that are configured on raw volumes. You cannot use SmartSync with volumes that contain file systems. Use an alternative solution such as DRL with such volumes.

You must configure volumes correctly to use SmartSync. For VxVM, there are two types of volumes used by the database, as follows:

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

SmartSync works with these two types of volumes differently, and they must be configured correctly to take full advantage of the extended interfaces. The only difference between the two types of volumes is that redo log volumes have dirty region logs, while data volumes do not.

To enable the use of SmartSync with database volumes in shared disk groups, set the value of the `volcvm_smartsync` tunable to `1` as described in "Tuning VxVM" on page 305. See "volcvm_smartsync" on page 311 for more information about this tunable.

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 which 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.

For details of how to configure sequential DRL, see "Adding DRL Logging to a Mirrored Volume" on page 211.

Hot-Relocation

Note You may need an additional license to use this feature.

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* and/or free space within the disk group. VxVM then reconstructs the objects that existed before the failure and makes them accessible again.

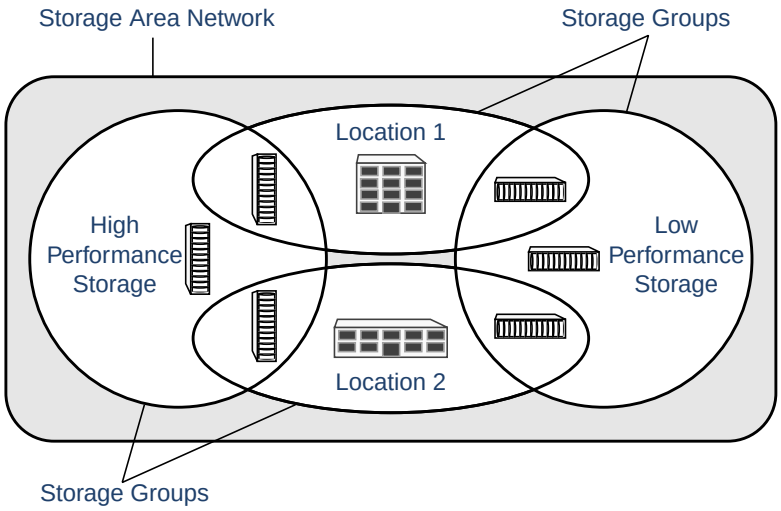
When a partial disk failure occurs (that is, a failure affecting only some subdisks on a disk), redundant data on the failed portion of the disk is relocated. Existing volumes on the unaffected portions of the disk remain accessible. For further details, see "Administering Hot-Relocation" on page 241.

Configuring Volumes on SAN Storage

Storage Area Networks (SANs) provide a networking paradigm that provides easily reconfigurable connectivity between any subset of computers, disk storage and interconnecting hardware such as switches, hubs and bridges. A SAN can contain a huge number of devices connected using either arbitrated or switched fabric. A SAN that has thousands or tens of thousands of connected devices is difficult to administer using a simple disk group model. VERITAS SANPoint™ Control 2.0 allows you to configure storage groups and storage accounts. Using SANPoint Control, you can allocate SAN storage more prudently and administer your complex SAN environments more effectively.

The figure "Dividing a Storage Area Network into Storage Groups" on page 50 illustrates how you might choose to set up storage groups within a SAN. In this example, the boundaries of the storage groups are based on the performance characteristics of different makes of disk array and on geographic location.

Dividing a Storage Area Network into Storage Groups



The `vxassist` utility in VERITAS Volume Manager understands storage groups that you have defined using SANPoint Control. `vxassist` supports a simple language that you can use to specify how disks are to be allocated from pre-defined storage groups. This specification language defines the confinement and separation criteria that `vxassist` applies to the available storage to choose disks for creating, resizing or moving a volume.

The following steps outline how to use SANPoint Control storage groups with `vxassist`:

1. Use SANPoint Control to define one or more storage groups. Note that zoning is not an issue as it is completely independent of storage group creation.

2. Use SANPoint Control to attach attribute-value pairs to each storage group's property sheet. Typically, you would assign values for the following attributes: location, storage group, and protection.

3. Use the `vxshowspc` command to discover the device names of disks that have a specified set of attributes, or to list the attributes of specified disks. For more information, see the `vxspcshow`(1M) manual page.

4. Use the `vxdiskadm` command or the VEA to configure the disks that you found in the previous step into VxVM disk groups.

5. Use `vxassist` to create volumes on disks that are selected by matching specified criteria for the values of storage group attributes. The usual restriction applies that a volume may only be created using disks from a single disk group. For more information about specifying the selection criteria for storage group attributes, see the `vxassist`(1M) manual page.

Note This feature of `vxassist` is designed to work in conjunction with SAL (SAN Access Layer) in VERITAS SANPoint Control 2.0. When VxVM with SAN-aware `vxassist` is installed on a host where SAL is also installed, it is recommended that you create a user named `root` under SAL. This allows `vxassist` to use the `root` login to contact the SAL daemon (`sald`) on the primary SAL server without needing to specify the `sal_username` attribute to `vxassist`. For more information, see the `vxassist`(1M) manual page.

Administering Disks 2

Introduction

This chapter describes the operations for managing disks used by the VERITAS Volume Manager (VxVM). This includes placing disks under VxVM control, initializing disks, encapsulating disks, mirroring the root disk, and removing and replacing disks.

Note Most VxVM commands require superuser or equivalent privileges.

For information about configuring and administering the Dynamic Multipathing (DMP) feature of VxVM that is used with multiported disk arrays, see "Administering Dynamic Multipathing (DMP)" on page 95.

Disk Devices

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

When a disk is placed under VxVM control, a VM disk is assigned to it. You can define a symbolic *disk name* (also known as a *disk media name*) to refer to a VM disk for the purposes of administration. A disk name can be up to 31 characters long. If you do not assign a disk name, it defaults to `disk##` if the disk is being added to `rootdg` (where ## is a sequence number). Otherwise, the default disk name is *`groupname`*`##`, where *`groupname`* is the name of the disk group to which the disk is added. Your system may use a device name that differs from the examples.

The *device name* (sometimes referred to as *devname* or *disk access name*) defines where the disk is located in a system.

Note The slice `s2` represents the entire disk. The entire disk is also implied if the slice is omitted from the device name.

The boot disk (which contains the root file system and is used when booting the system) is often identified to VxVM by the device name `c0t0d0`.

Note The full pathname of a device is `/dev/vx/[r]dsk/`*devicename*. In this document, only the device name is listed and `/dev/vx/[r]dsk` is assumed.

Disk Device Naming in VxVM

Prior to VxVM 3.2, all disks were named according to the c#t#d#s# format. Fabric mode disks were not supported by VxVM. From VxVM 3.2 onward, there are two different methods of naming disk devices:

- c#t#d#s# Based Naming Scheme
- Enclosure Based Naming Scheme

Note Disk devices controlled by MPXIO are always in fabric mode (irrespective of its hardware configuration), and are therefore named in the *enclosure name* format. This is true for both naming schemes.

c#t#d#s# Based Naming Scheme

In this naming scheme, all disk devices except fabric mode disks are named using the c#t#d#s# format.

The syntax of a device name is `c#t#d#s#`, where `c#` represents a controller on a host bus adapter, `t#` is the target controller ID, `d#` identifies a disk on the target controller, and `s#` represents a partition (or *slice)* on the disk.

Fabric mode disk devices are named as follows:

- Disk in supported disk arrays are named using the *enclosure name*`_#` format. For example, disks in the supported disk array name `FirstFloor` are named `FirstFloor_0`, `FirstFloor_1`, `FirstFloor_2` and so on. (You can use the `vxdmpadm` command to administer enclosure names.)
- Disks in the DISKS category (formerly know as JBOD disks) are named using the `Disk_#` format.
- Disks in the OTHER_DISKS category are named using the `fabric_#` format

Enclosure Based Naming Scheme

The enclosure-based naming scheme 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 "Administering DMP Using vxdmpadm" on page 111 and the `vxdmpadm`(1M) manual page for more information.)

- Disks in the DISKS category (formerly known as JBOD disks) are named using the `Disk_#` format.
- Disks in the OTHER_DISKS category are named as follows:
  - Non-fabric disks are named using the c#t#d#s# format.
  - Fabric disks are named using the `fabric_#` format.

See "Changing the Disk-Naming Scheme" on page 61 for details of how to switch between the two naming schemes.

To display the native OS device names of a VM disk (such as `disk01`), use the following command:

```
# vxdisk list diskname
```

For information on how to rename an enclosure, see "Renaming an Enclosure" on page 115.

Private and Public Disk Regions

A VM disk usually has two regions:

- *private region*—a small area where configuration information is stored. A disk header label, configuration records for VxVM objects (such as volumes, plexes and subdisks), and an intent log for the configuration database are stored here. The default private region size is 2048 blocks (1024 kilobytes), which is large enough to record the details of about 4000 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 (as described in "Splitting Disk Groups" on page 142). If required, the value for the private region size may be overridden at installation time by choosing the Custom Installation path, or when you add or replace a disk using the `vxdiskadm` command.

Note 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 is used to store subdisks (and allocate storage space).

The following basic disk types are used by VxVM:

- *sliced*—the public and private regions are on different disk partitions. Typically, most or all disks on your system are configured as this disk type.
- *simple*—the public and private regions are on the same disk area (with the public area following the private area).
- *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. Nopriv 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. Such 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.

On some systems, VxVM asks the operating system for a list of known disk device addresses. These device addresses are auto-configured into the `rootdg` disk group when `vxconfigd` is started. Auto-configured disks are always of type sliced, with default attributes. VxVM initializes each new disk with the fewest number of partitions possible. For VM disks of type `sliced`, VxVM usually configures partition `s3` as the private region, `s4` as the public region, and `s2` as the entire physical disk. An exception is an encapsulated root disk, on which `s3` is usually configured as the public region and `s4` as the private region.

For more information about disk types and their configuration, see the `vxdisk`(1M) manual page.

Metadevices

Two classes of disk device files can be used with VxVM: standard devices, and special devices known as *metadevices*. Metadevices are only used with operating systems that support dynamic multipathing (DMP). Such devices represent the physical disks that a system can access via more than one physical path. The available access paths depend on whether the disk is a single disk, or part of a multiported disk array that is connected to a system.

You can use the `vxdisk` utility to display the paths subsumed by a metadevice, and to display the status of each path (for example, whether it is enabled or disabled). For more information, see “Administering Dynamic Multipathing (DMP)” on page 95.

Discovering and Configuring Newly Added Disk Devices

The `vxdiskconfig` utility scans and configures new disk devices attached to the host, disk devices that become online, or fibre channel devices that are zoned to host bus adapters connected to this host. The command calls platform specific interfaces to configure new disk devices and brings them under control of the operating system. It scans for disks that were added since VxVM's configuration daemon was last started. These disks are then dynamically configured and recognized by VxVM.

`vxdiskconfig` should be used whenever disks are physically connected to the host or when fibre channel devices are zoned to the host.

`vxdiskconfig` calls `vxdctl enable` to rebuild volume device node directories and update the DMP internal database to reflect the new state of the system.

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. See the `vxdisk`(1M) manual page for more information.

Discovering Disks and Dynamically Adding Disk Arrays

You can dynamically add support for a new type of disk array which has been developed by a third-party vendor. The support comes in the form of vendor supplied libraries, and is added to a Solaris system by using the `pkgadd` command.

Adding Support for a New Disk Array

The following example illustrates how to add support for a new disk array named `vrtsda` to a Solaris system using a vendor-supplied package on a mounted CD-ROM:

```
# pkgadd -d /cdrom/pkgdir vrtsda
```

The new disk array does not need to be already connected to the system when the package is installed. 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.

Device Discovery Function

To have VxVM discover a new disk array, use the following command:

```
# vxdctl enable
```

This 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.

Removing Support for a Disk Array

To remove support for the `vrtsda` disk array, use the following command:

```
# pkgrm vrtsda
```

If the arrays remain physically connected to the host after support has been removed, they are listed in the OTHER_DISKS category, and the volumes remain available.

Administering the Device Discovery Layer

Dynamic addition of disk arrays is possible because of the existence of the Device Discovery Layer (DDL) which is a facility for discovering disks and their attributes that are required for VXVM and DMP operations.

Administering the DDL is the role of the `vxddladm` utility which is an administrative interface to the DDL. You can use `vxddladm` to perform the following tasks:

- 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 the supported JBODs.
- Add JBOD support for disks from different vendors.
- Remove support for a JBOD.

The following sections explain these tasks in more detail. For further information, see the `vxddladm`(1M) manual page.

Listing Details of Supported Disk Arrays

To list all currently supported disk arrays, use the following command

```
# vxddladm listsupport
```

Note Use this command to obtain values for the `vid` and `pid` attributes that are used with other forms of the `vxddladm` command.

Excluding Support for a Disk Array

To exclude a particular array library from participating in device discovery, use the following command:

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

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

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

This array is also excluded from device discovery.

For more information about excluding disk array support, see the `vxddladm` (1M) manual page.

Re-including Support for an Excluded Disk Array

If you have excluded support for a particular disk array, 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
```

This command adds the array library to the database so that the library can once again be used in device discovery. If `vxconfigd` is running, you can use the `vxdisk scandisks` command to discover the array and add its details to the database.

Listing Excluded Disk Arrays

To list all disk arrays that are currently excluded from use by VxVM, use the following command:

```
# vxddladm listexclude
```

Listing Supported Disks in the JBOD Category

To list supported disks in the JBOD category, use the following command:

```
# vxddladm listjbod
```

Adding Support for Disks in the JBOD Category

To add support for disks that are in the JBOD category, use the `vxddladm` command with the `addjbod` keyword. The following example illustrates the command for adding disks from the vendor, Seagate:

```
# vxddladm addjbod vid=SEAGATE
```


To add support for X1 disks from ACME, use the following command:

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

Removing Support for Disks in the JBOD Category

To remove support for disks that are in the JBOD category, use the `vxddladm` command with the `rmjbod` keyword. The following example illustrates the command for removing disks supplied by the vendor, Seagate:

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

To remove support for X1 disks from ACME, use the following command:

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

Placing 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. Unless another disk group is specified, VxVM places new disks in the default disk group, `rootdg`.

The method by which you place a disk under VxVM control depends on the 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.

Caution 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, 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*.

Note 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.

- When initializing or encapsulating multiple disks at once, it is possible to exclude certain disks or controllers.

  To exclude disks, list the names of the disks to be excluded in the file `/etc/vx/disks.exclude` before the initialization or encapsulation. The following is an example of the contents of a `disks.exclude` file:

  ```
  c0t1d0
  ```

  You can exclude all disks on specific controllers from initialization or encapsulation by listing those controllers in the file `/etc/vx/cntrls.exclude`. The following is an example of an entry in a `cntrls.exclude` file:

  ```
  c0
  ```

  You can exclude all disks in specific enclosures from initialization or encapsulation by listing those enclosures in the file `/etc/vx/enclr.exclude`. The following is an example of an entry in a `enclr.exclude` file:

  ```
  enc1
  ```

Note Only the `vxinstall` and `vxdiskadm` commands use the contents of the `/etc/vx/disks.exclude`, `/etc/vx/cntrls.exclude` and `/etc/vx/enclr.exclude` files. You may need to create these files if they do not already exist on the system.

Changing the Disk-Naming Scheme

You can either use enclosure-based naming for disks or the traditional naming scheme (such as `c#t#d#s#`). Select menu item 20 from the `vxdiskadm` main menu to change the disk-naming scheme that you want VxVM to use. Selecting this option displays the following screen:

```
Change the disk naming scheme
Menu: VolumeManager/Disk/NamingScheme

Use this screen to change the disk naming scheme (from the c#t#d#
format to the enclosure based format and vice versa).

NOTE: This operation will result in vxconfigd being stopped and
restarted.

Volume Manager is currently using the enclosure based format to
name disks on the system.

Do you want to change the naming scheme ? [y,n,q,?] (default: n)
```


Enter **y** to change the naming scheme. This restarts the `vxconfig` daemon to bring the new disk naming scheme into effect.

Using vxprint with Enclosure-Based Disk Names

If you enable enclosure-based naming, and use the `vxprint` command to display the structure of a volume, it shows enclosure-based disk device names (disk access names) rather than c#t#d# names. To discover the c#t#d# names that are associated with a given enclosure-based disk name, use either of the following commands:

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

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

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

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

Issues Regarding Simple/Nopriv Disks with Enclosure-Based Naming

If you change from the *c#t#d#s#* based naming scheme to the enclosure-based naming scheme, simple or nopriv disks may be put in the "`error`" state and cause VxVM objects on those disks to fail. If this happens, use the following procedures to correct the problem:

- Simple / Nopriv Disks in the Root Disk Group
- Simple / Nopriv Disks in Non-Root Disk Groups

These procedures use the `vxdarestore` utility to handle simple / nopriv disk failures that arise from changing to the enclosure-based naming scheme. You do not need to perform either procedure if your system does not have any simple or nopriv disks, or if the devices on which any simple or nopriv disks are present are not automatically configured by VxVM (for example, non-standard disk devices such as ramdisks).

Note You cannot run `vxdarestore` if the c#t#d#s# naming scheme is in use. Additionally, `vxdarestore` does not handle failures on simple / nopriv disks that are caused by renaming enclosures, by hardware reconfiguration that changes device names, or by changing the naming scheme on a system that includes persistent sliced disk records.

For more information about the vxdarestore command, see the vxdarestore(1M) manual page.

Simple/Nopriv Disks in the Root Disk Group

If the root disk group (rootdg) is comprised of only simple and/or nopriv disks, the vxconfigd daemon goes into the disabled state after the naming scheme change. In such a case, perform the following steps:

1. Use vxdiskadm to change back to the c#t#d#s# naming scheme.
2. Either shut down and reboot the system, or enter the following command to restart the VxVM configuration daemon:

   # **vxconfigd -kr reset**
3. If you want to use the enclosure-based naming scheme, use vxdiskadm to add a sliced disk to the rootdg disk group, change back to the enclosure-based naming scheme, and then run the following command:

   # **/usr/bin/vxvm/bin/vxdarestore**

Simple/Nopriv Disks in Non-Root Disk Groups

If an imported disk group, other than rootdg, is comprised of only simple and/or nopriv disks, the disk group is in the "online dgdisabled" state after the change to the enclosure-based naming scheme. In such a case, perform the following steps:

1. Deport the disk group using the following command:

   # **vxdg deport *diskgroup***
2. Use the vxdarestore command to restore the failed disks, and to recover the objects on those disks:

   # **/usr/bin/vxvm/bin/vxdarestore**
3. Re-import the disk group using the following command:

   # **vxdg import *diskgroup***

Installing and Formatting Disks

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.

The following sections provide detailed examples of how to use the `vxdiskadm` utility to place disks under VxVM control in various ways and circumstances.

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 the system, 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 may be initialized, while others may be encapsulated.

Caution Initialization does not preserve data on disks.

When initializing or encapsulating multiple disks at one time, it is possible to exclude certain disks or certain controllers. To exclude disks, list the names of the disks to be excluded in the file `/etc/vx/disks.exclude` before the initialization or encapsulation. You can exclude all disks on specific controllers from initialization or encapsulation by listing those controllers in the file `/etc/vx/cntrls.exclude`.

Initialize disks for VxVM use as follows:

1. Select menu item `1 (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):

```
Add or initialize disks
Menu: VolumeManager/Disk/AddDisks

Use this operation to add one or more disks to a disk group.
You can add the selected disks to an existing disk group or to
a new disk group that will be created as a part of the
operation. The selected disks may also be added to a disk
group as spares. Or they may be added as nohotuses to be excluded
from hot-relocation use. The selected disks may also be initialized
without adding them to a disk group leaving the disks
available for use as replacement disks.

More than one disk or pattern may be entered at the prompt.
Here are some disk selection examples:

all:       all disks
c3 c4t2:   all disks on both controller 3 and controller
           4,target 2
c3t4d2:    a single disk (in the c#t#d# naming scheme)
xyz_0 :    a single disk (in the enclosure based naming scheme)
xyz_ :     all disks on the enclosure whose name is xyz

Select disk devices to add:
[<pattern-list>,all,list,q,?]
```


<pattern-list> can be a single disk, or a series of disks and / or controllers (with optional targets). If *<pattern-list>* consists of multiple items, separate them using white space, for example:

```
c3t0d0 c3t1d0 c3t2d0 c3t3d0
```

specifies fours disks at separate target IDs on controller 3.

If you enter **list** at the prompt, the `vxdiskadm` program displays a list of the disks available to the system:

```
DEVICE       DISK         GROUP        STATUS
c0t0d0       disk01       rootdg       online
c0t1d0       disk02       rootdg       online
c0t2d0       disk03       rootdg       online
c0t3d0       -            -            online
c1t0d0       disk10       rootdg       online
c1t0d1       -            -            error
.
.
.
c3t0d0       -            -            error
sena0_0      disk33      rootdg       online
sena0_1      disk34      rootdg       online
sena0_2      disk35      rootdg       online

Select disk devices to add:
[<pattern-list>,all,list,q,?]
```

The phrase `error` 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, **none** to reserve the disks for future use, or press Return to accept `rootdg`:

   ```
   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,?] (default: rootdg)
```

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
   ```

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)
   ```

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 disk from hot-relocation use? [y,n,q,?} (default: n) n
   ```

9. 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
   ```

10. The following prompt lists any disks that have already been initialized for use by VxVM; enter **y** to indicate that all of these disks should now be used:

    ```
    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
    ```

    Note that 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 re-initialize all of these disks, enter **Y** at the following prompt:

```
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

```
Are you sure you want to re-initialize these disks?
[Y,N,S(elect),q,?] (default: N) Y
```

11. `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.

```
The following disk device has a valid VTOC, 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)
```

- 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:

```
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:

The disk device *device name* will be encapsulated and added to the disk group rootdg with the disk name *disk name*.

At the following prompt, `vxdiskadm` asks if you want to use the default private region size of 2048 blocks. Enter **y** to confirm that you want to use the default value. Enter **n** if you want to use a different value. (The maximum value that you can specify is 524288 blocks.)

```
Use a default private region length for the disk? [y,n,q,?]
(default: y)
```

`vxdiskadm` then proceeds to encapsulate the disks.

```
The first stage of encapsulation has completed successfully.
You should now reboot your system at the earliest possible
opportunity.

The encapsulation will require two or three reboots which will
happen automatically after the next reboot.  To reboot execute
the command:

shutdown -g0 -y -i6


This will update the /etc/vfstab file so that volume devices are
used to mount the file systems on this disk device.  You will
need to update any other references such as backup scripts,
databases, or manually created swap devices.
```

- 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.

  ```
  Initializing device device name.
  ```

  At the following prompt, `vxdiskadm` asks if you want to use the default private region size of 2048 blocks. Enter **y** to confirm that you want to use the default value. Enter **n** if you want to use a different value. (The maximum value that you can specify is 524288 blocks.)

  ```
  Use a default private region length for the disk? [y,n,q,?]
  (default: y)
  ```

  `vxdiskadm` then proceeds to add the disks.

  ```
  Adding disk device device name to disk group disk group name with disk
  name disk name.
  ...
  ```


12. 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)
    ```

Reinitializing a Disk

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 65 for details.

Caution 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 VxVM, encapsulate the disk and preserve its information.

Using vxdiskadd to Place a Disk Under Control of VxVM

As an alternative to `vxdiskadm`, you can use the `vxdiskadd` command to put a disk under VxVM control. For example, to initialize the second disk on the first controller, use the following command:

```
# vxdiskadd c0t1d0
```

The `vxdiskadd` command examines your disk to determine whether it has been initialized and also checks for disks that can be encapsulated (see "Using nopriv Disks for Encapsulation" on page 74), disks that have been added to VxVM, and for other conditions.

Note If you are adding an uninitialized disk, warning and error messages are displayed on the console during 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`, described in "Adding a Disk to VxVM" on page 65.

Encapsulating a Disk for Use in VxVM

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.

Caution Encapsulating a disk requires that the system be rebooted several times. Schedule performance of this procedure for a time when this does not inconvenience users.

To prevent the encapsulation failing, make sure that:

- The disk has a small amount of free space (at least 1 megabyte at the beginning or end of the disk) that does not belong to any partition.
- The disk has two free partitions.
- The disk has an `s2` slice that represents the whole disk.

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. See "Rootability" on page 76 and following sections for more information.

To encapsulate a disk for use in VxVM, use the following procedure:

1. Select menu item `2 (Encapsulate one or more disks)` from the `vxdiskadm` main menu.

Note 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:

```
Encapsulate one or more disks
Menu: VolumeManager/Disk/Encapsulate

Use this operation to convert one or more disks to use the
Volume Manager. This adds the disks to a disk group and
replaces existing partitions with volumes.  Disk encapsulation
requires a reboot for the changes to take effect.

More than one disk or pattern may be entered at the prompt.
Here are some disk selection examples:

all:    all disks
c3 c4t2:all disks on both controller 3 and controller
        4,target 2
c3t4d2:    a single disk (in the c#t#d# naming scheme)
xyz_0 :    a single disk (in the enclosure based naming scheme)
xyz_ :     all disks on the enclosure whose name is xyz
```


```
Select disk devices to encapsulate:
[<pattern-list>,all,list,q,?] device name
```

Where <*pattern-list*> can be a single disk, or a series of disks and / or controllers (with optional targets). 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.

2. 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
```

3. To add the disk to the default disk group, `rootdg`, press Return at the following prompt; otherwise, enter the name of another disk group to which the disk should be added (this disk group need not already exist):

```
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,?] (default: rootdg)
```

4. 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)
```

5. To continue with the operation, enter **`y`** (or press Return) at the following prompt:

```
The selected disks will be encapsulated and added to the rootdg
disk group with default disk names.

device name

Continue with operation? [y,n,q,?] (default: y) y
```

6. 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 rootdg with the disk name disk01.
```

7. At the following prompt, `vxdiskadm` asks if you want to use the default private region size of 2048 blocks. Enter **y** to confirm that you want to use the default value. Enter **n** if you want to use a different value. (The maximum value that you can specify is 524288 blocks.)

   ```
   Use a default private region length for the disk? [y,n,q,?]
   (default: y)
   ```

8. `vxdiskadm` then proceeds to encapsulate the disks.

   ```
   The device name disk has been configured for encapsulation.

   The first stage of encapsulation has completed successfully.
   You should now reboot your system at the earliest possible
   opportunity.

   The encapsulation will require two or three reboots which
   will happen automatically after the next reboot.  To reboot
   execute the command:

   shutdown -g0 -y -i6

   This will update the /etc/vfstab file so that volume devices
   are used to mount the file systems on this disk device.  You
   will need to update any other references such as backup
   scripts, databases, or manually created swap devices.
   ```

Note The original `/etc/vfstab` file is saved as `/etc/vfstab.prevm`.

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
```


Failure of Disk Encapsulation

Under some circumstances, encapsulation of a disk can fail. Usually this is because there is not enough free space available on the disk to accommodate the private region. If this happens, the procedure outlined above ends abruptly with an error message similar to the following:

```
The encapsulation operation failed with the following error:
It is not possible to encapsulate device, for the following reason:
<vxvm:vxslicer: ERROR: Unsupported disk layout.>
```

See the following section, "Using nopriv Disks for Encapsulation" for advice about what you can do if this occurs.

Using nopriv Disks for Encapsulation

Encapsulation converts existing partitions on a specified disk to volumes. If any partitions contain file systems, their `/etc/vfstab` 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, 1 megabyte) 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 *VERITAS Volume Manager Installation Guide* and the `vxencap`(1M) manual page for more information.)

You can encapsulate a disk that does not have space available for the VxVM private region partition by using the `vxdisk` utility. This is done by configuring the disk as a `nopriv` devices that does not have a private region.

To create a `nopriv` device:

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 `c0t4d0`, use the following command:

   ```
   # vxdisk define c0t4d0s3 type=nopriv
   ```

To create volumes for other partitions on the disk:

1. Add the partition to a disk group.
2. Determine where the partition resides within the encapsulated partition.
3. Use `vxassist` to create a volume with that offset and length.

Note `vxassist`, by default, reinitializes 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`.

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.

Note 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.

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*.

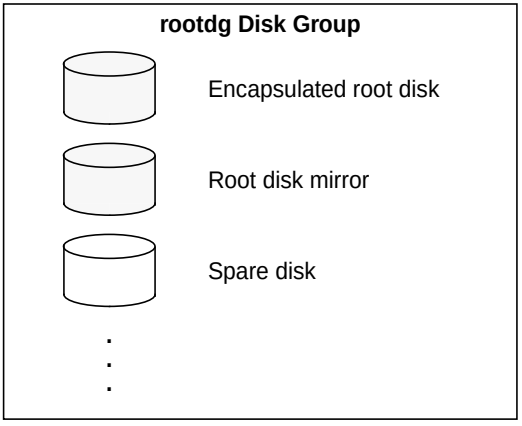
Encapsulation converts existing partitions on that disk to volumes. Once under VxVM control, the `root` and `swap` devices appear as volumes and provide the same characteristics as other VxVM volumes. A volume that is configured for use as a swap area is referred to as a *swap volume*, and a volume that contains the root file system is referred to as a *root volume*.

Caution Only encapsulate your root disk if you also intend to mirror it. There is no benefit in root-disk encapsulation for its own sake.

It is possible to 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 VxVM rootability, 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.

Suggested Rootability Configuration



The figure, "Suggested Rootability Configuration" on page 76, shows one possible assignment of disks in the `rootdg` disk group. This arrangement consists of the encapsulated root disk, the root disk mirror, and at least one spare disk. If hot-relocation is enabled and either the root disk or its mirror fails during use, VxVM automatically recreates the failed plexes on the spare disk by copying from the plexes on remaining disk. If the spare disk is suitable, it may then be configured to be booted from, or you can use it to recover the contents of the failed disk when it is replaced.

For more information on system recovery procedures for the boot disk, see the chapter "Recovery from Boot Disk Failure" in the *VERITAS Volume Manager Troubleshooting Guide*.

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.

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, `rootdg`. Although other volumes named `rootvol` can be created in disk groups other than `rootdg`, only the `rootvol` in `rootdg` 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 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).

Encapsulating and Mirroring the Root Disk

VxVM allows you to 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.

Note Use the `format` or `fdisk` 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, for example, the section "Repairing root or /usr File Systems on Mirrored Volumes" in the chapter "Recovery from Boot Disk Failure" in the *VERITAS Volume Manager Troubleshooting Guide*.

Encapsulation requires that the root disk:

- has two free partitions for the private and public regions
- has an `s2` slice that represents the whole disk

It is also helpful if the root disk has a small amount of free space (at least 1 megabyte) at the beginning or end of the disk to use for the private region. If not, a similar sized portion of the swap partition is used instead.

To mirror your `root` disk onto another disk:

1. Choose a disk that is at least as large as the existing `root` disk.

2. If the selected disk is not already under VxVM control, use the `vxdiskadd` or `vxdiskadm` command, or the VERITAS Enterprise Administrator (VEA) to add it to the `rootdg` disk group.

3. Select menu item `6 (Mirror Volumes on a Disk)` from the `vxdiskadm` main menu, or use the VEA to create a mirror of the root disk. (These automatically invoke the `vxrootmir` command if the mirroring operation is performed on the `root` disk.)

   Alternatively, to mirror only those file systems on the root disk that are required to boot the system, run the following command:

   # **`/etc/vx/bin/vxrootmir` *`altboot_disk`***

   where *altboot_disk* is the disk media name of the mirror for the root disk. `vxrootmir` creates a mirror for `rootvol` (the volume for the `root` file system on an alternate disk). The alternate `root` disk is configured to enable booting from it if the primary `root` disk fails.

4. Set the value of the EEPROM variable `use-nvramrc?` to `true`. This enables the use of VxVM boot disk aliases, which identify mirrors of the `root` disk from which the system can be booted. If the system is up and running, set `use-nvramrc?` to `true` using the following command:

   # **`eeprom use-nvramrc?=true`**

   You can also set `use-nvramrc?` at the `ok` boot prompt:

   `ok` **`setenv use-nvramrc? true`**

After following these steps, you should be able to boot the system from an alternate boot disk, `vx-`*altboot_disk*, by entering the following command at the `ok` boot prompt:

`ok` **`boot vx-`*`altboot_disk`***

You can use the `devalias` command at the boot prompt to discover the alternate disks from which the system may be booted:

`ok` **`devalias`**

Defining Alternate Boot Disks

If required, you can define an alternate boot disk by entering the following command:

```
ok nvramrc=devalias vx-altboot_disk
```

where *altboot_disk* is the device name of an alternate disk from which the system can be booted.

Alternatively, if the system is already up and running, enter the following command to define an alternate boot disk:

```
# eeprom nvramrc=devalias vx-altboot_disk
```

Mirroring Other File Systems on the Root Disk

There may be other volumes on the root disk, such as volumes for `/home` or `/tmp` file systems. If necessary, these can be mirrored separately using the `vxassist` utility. For example, if you have a `/home` file system on a volume `homevol`, you can mirror it to *alternate_disk* using the command:

```
# vxassist mirror homevol alternate_disk
```

If you do not have space for a copy of some of these file systems on your alternate boot disk, you can mirror them to other disks. You can also span or stripe these other volumes across other disks attached to your system.

To list all volumes on your primary boot disk, use the command:

```
# vxprint -t -v -e'aslist.aslist.sd_disk="boot_disk"'
```

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.

To remove rootability from a system, follow these steps:

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 `rootvol-02`, `swapvol-02`, and `home-02` that are configured on disk `disk01`:

   ```
   # vxplex -o rm dis rootvol-02 swapvol-02 home-02
   ```

Caution Do not remove the plexes on the root disk that correspond to the original disk partitions.

2. Run the `vxunroot` utility:

   ```
   # /etc/vx/bin/vxunroot
   ```

   `vxunroot` does not perform any conversion to disk partitions if any plexes remain on other disks.

Note This operation requires a reboot of the system.

Using RAM Disks with VxVM

Note This section only applies to systems which support RAM disks.

Some systems support creation of RAM disks. A RAM disk is a device made from system RAM 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.

Note To use a RAM disk with VxVM, block and character device nodes must exist for the RAM disk, for example, `/dev/dsk/ramd0` and `/dev/rdsk/ramd0`.

To define the RAM disk device to VxVM, use the following command:

```
# vxdisk define ramd0 type=nopriv volatile
```


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 for more information on the `vxvol set` operation and the `norecov` option.

Removing Disks

You can remove a disk from a system and move it to another system if the disk is failing or has failed. Before removing the disk from the current system, you must:

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 stop volume1 volume2 ...
   ```

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.

Before removing a disk, make sure any data on that disk has either been moved to other disks or is no longer needed. Then remove the disk using the `vxdiskadm` utility, as follows:

1. Select menu item `3 (Remove a disk)` from the `vxdiskadm` main menu.

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

2. At the following prompt, enter the disk name of the disk to be removed:

   ```
   Remove a disk
   Menu: VolumeManager/Disk/RemoveDisk
   ```

```
Use this operation to remove a disk from a disk group. This
operation takes a disk name as input. This is the same name
that you gave to the disk when you added the disk to the disk
group.

Enter disk name [<disk>,list,q,?] disk01
```

3. 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**.

4. At the following verification prompt, press Return to continue:

   ```
   Requested operation is to remove disk disk01 from group rootdg.
   Continue with operation? [y,n,q,?] (default: y)
   ```

   The `vxdiskadm` utility removes the disk from the disk group and displays the following success message:

   ```
   Removal of disk disk01 is complete.
   ```

   You can now remove the disk or leave it on your system as a replacement.

5. 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. To do this, run the `vxdiskadm` program and select item `3 (Remove a disk)` from the main menu.

If the disk is used by some subdisks, the following message is displayed:

```
The following volumes currently use part of disk disk02:

   home usrvol

   Subdisks must be moved from disk02 before it can be removed.

   Move subdisks to other disks? [y,n,q,?] (default: n)
```


If you choose **y**, then all subdisks are moved off the disk, if possible. 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 subdisk's 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 219 and "Taking Plexes Offline" on page 166 for information on how to remove volumes and plexes.

Removing a Disk with No Subdisks

To remove a disk that contains no subdisks from its disk group, run the `vxdiskadm` program and select item `3 (Remove a disk)` from the main menu, and respond to the prompts as shown in this example to remove `disk02`:

```
Enter disk name [<disk>,list,q,?] disk02

  Requested operation is to remove disk disk02 from group rootdg.

Continue with operation? [y,n,q,?] (default: y) y
  Removal of disk disk02 is complete.
```

Removing and Replacing Disks

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.

To replace a disk, use the following procedure:

1. Select menu item `4 (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):

   ```
   Remove a disk for replacement
   Menu: VolumeManager/Disk/RemoveForReplace

   Use this menu operation to remove a physical disk from a disk
   group, while retaining the disk name. This changes the state
   ```

```
for the disk name to a removed disk. If there are any
initialized disks that are not part of a disk group, you will be
given the option of using one of these disks as a replacement.

Enter disk name [<disk>,list,q,?] disk02
```

3. When you select a disk to remove for replacement, all volumes that are affected by the operation are displayed, for example:

```
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 `disk02`, use this command:

   ```
   # vxassist move mkting !disk02
   ```

   After backing up or moving the data in the volumes, start again from step 1 above.

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** to defer replacing the disk until a later date:

```
The following devices are available as replacements:
c0t1d0s2

You can choose one of these disks now, to replace disk02.
Select “none” if you do not wish to select a replacement disk.

Choose a device, or select “none”
[<device>,none,q,?] (default: c0t1d0s2)
```


Note 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.

If you choose to defer replacing the failed disk, see the following section, "Replacing a Failed or Removed Disk."

5. If you chose to replace the disk in step 4, press Return at the following prompt to confirm this:

```
Requested operation is to remove disk02 from group rootdg.
The removed disk will be replaced with disk device c0t1d0s2.
Continue with operation? [y,n,q,?] (default: y)
```

If the disk was previously an encapsulated root disk, `vxdiskadm` displays the following message. Enter **y** to confirm that you want to reinitialize the disk:

```
The disk c1t0d0s2 was a previously encapsulated root disk. Due to
the disk layout that results from root disk encapsulation, the
preferred action is to reinitialize and reorganize this disk.
However, if you have any non-redundant data on this disk you should
not reorganize this disk, as the data will be lost.
Reorganize the disk? [y,n,q,?] (default: n) y
```

6. If the disk has not been initialized with a private region, `vxdiskadm` asks if you want to use the default private region size of 2048 blocks. Enter **y** to confirm that you want to use the default value. Enter **n** if you want to use a different value. (The maximum value that you can specify is 524288 blocks.)

```
Use a default private region length for the disk? [y,n,q,?]
(default: y)
```

7. `vxdiskadm` displays the following success messages:

```
Replacement of disk disk02 in group rootdg with disk device
c0t1d0s2 completed successfully.
```

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)
```

Note If removing a disk causes one or more volumes to be disabled, see the section, "Restarting a Disabled Volume" in the chapter "Recovery from Hardware Failure" in the *VERITAS Volume Manager Troubleshooting Guide*, for information on how to restart a disabled volume so that you can restore its data from a backup.

If you wish to move hot-relocate subdisks back to a replacement disk, see "Configuring Hot-Relocation to Use Only Spare Disks" on page 252.

Replacing a Failed or Removed Disk

Use the following procedure to replace a failed or removed disk with a new disk:

1. Select menu item `5 (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):

   ```
   Replace a failed or removed disk
   Menu: VolumeManager/Disk/ReplaceDisk

   Use this menu operation to specify a replacement disk for a disk
   that you removed with the “Remove a disk for replacement” menu
   operation, or that failed during use. You will be prompted for
   a disk name to replace and a disk device to use as a replacement.
   You can choose an uninitialized disk, in which case the disk
   will be initialized, or you can choose a disk that you have
   already initialized using the Add or initialize a disk menu
   operation.

   Select a removed or failed disk [<disk>,list,q,?] disk02
   ```

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:
   c0t1d0s2 c1t1d0s2

   You can choose one of these disks to replace disk02.
   Choose "none" to initialize another disk to replace disk02.

   Choose a device, or select "none"
   [<device>,none,q,?] (default: c0t1d0s2)
   ```

4. If the disk has not previously been initialized, press Return at the following prompt to replace the disk:

   ```
   The requested operation is to initialize disk device
   c0t1d0s2 and to then use that device to replace the removed or
   failed disk disk02 in disk group rootdg.
   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:

```
The requested operation is to use the initialized device
c0t1d0s2 to replace the removed or failed disk disk02 in disk group
rootdg.
Continue with operation? [y,n,q,?] (default: y)
```

If the disk was previously an encapsulated root disk, `vxdiskadm` displays the following message. Enter **y** to confirm that you want to reinitialize the disk:

```
The disk c0t1d0s2 was a previously encapsulated root disk. Due to
the disk layout that results from root disk encapsulation, the
preferred action is to reinitialize and reorganize this disk.
However, if you have any non-redundant data on this disk you should
not reorganize this disk, as the data will be lost.
Reorganize the disk? [y,n,q,?] (default: n) y
```

5. If the disk has not been initialized with a private region, `vxdiskadm` asks if you want to use the default private region size of 2048 blocks. Enter **y** to confirm that you want to use the default value. Enter **n** if you want to use a different value. (The maximum value that you can specify is 524288 blocks.)

   ```
   Use a default private region length for the disk? [y,n,q,?]
   (default: y)
   ```

6. The `vxdiskadm` program then proceeds to replace the disk, and returns the following message on success:

   ```
   Replacement of disk disk02 in group rootdg with disk device
   c0t1d0s2 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 Physical 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, use the following procedure:

1. Select menu item `10 (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):

   ```
   Enable (online) a disk device
   Menu: VolumeManager/Disk/OnlineDisk

   Use this operation to enable access to a disk that was disabled
   with the “Disable (offline) a disk device” operation.

   You can also use this operation to re-scan a disk that may have
   been changed outside of the Volume Manager. For example, if a
   disk is shared between two systems, the Volume Manager running
   on the other system may have changed the disk. If so, you can
   use this operation to re-scan the disk.

   NOTE: Many vxdiskadm operations re-scan disks without user
      intervention. This will eliminate the need to online a
      disk directly, except when the disk is directly offlined.

   Select a disk device to enable [<address>,list,q,?] c0t2d0s2
   ```

   `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)
   ```

You can also issue the command `vxdctl enable` after a hot disk swap. This enables VxVM to recognize the new disk and any paths to it that are available through Dynamic Multipathing (DMP).

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.

Note Taking a disk offline is only useful on systems that support *hot-swap* removal and insertion of disks without needing to shut down and reboot the system.

To take a disk offline, use the `vxdiskadm` command:

1. Select menu item `11 (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:

```
Disable (offline) a disk device
Menu: VolumeManager/Disk/OfflineDisk

Use this menu operation to disable all access to a disk device
by the Volume Manager. This operation can be applied only to
disks that are not currently in a disk group. Use this operation
if you intend to remove a disk from a system without rebooting.

NOTE: Many systems do not support disks that can be removed from
   a system during normal operation. On such systems, the
   offline operation is seldom useful.
Select a disk device to disable [<address>,list,q,?] c0t2d0s2
```

   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 change the disk name to reflect a change of use or ownership, use the following command:

```
# vxedit rename old_diskname new_diskname
```

To rename `disk01` to `disk03`, use the following command:

```
# vxedit rename disk01 disk03
```

To confirm that the name change took place, use the following command:

```
# vxdisk list
```

VxVM returns the following display:

```
DEVICE       TYPE      DISK      GROUP     STATUS
c0t0d0s2     sliced    disk04    rootdg    online
c1t0d0s2     sliced    disk03    rootdg    online
c1t1d0s2     sliced    -         -         online
enc0_5       sliced    disk05    rootdg    online
enc0_8       sliced    disk08    rootdg    online
```

Note 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.

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 for special purposes, use the following command:

```
# vxedit 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 `disk03` is reserved, use the following command:

```
# vxassist make vol03 20m disk03
```

The `vxassist` command overrides the reservation and creates a 20 megabyte volume on `disk03`. However, the command:

```
# vxassist make vol04 20m
```

does not use `disk03`, even if there is no free space on any other disk.

To turn off reservation of a disk, use the following command:

```
# vxedit set reserve=off diskname
```

See Special Attribute Values for Disk Media in `vxedit`(1M) for more information.

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, use the following command:

```
# vxdisk list
```

VxVM returns a display similar to the following:

```
DEVICE      TYPE     DISK     GROUP    STATUS
c0t0d0s2    sliced   disk04   rootdg   online
c1t0d0s2    sliced   disk03   rootdg   online
c1t1d0s2    sliced   -        -        error
enc0_2      sliced   disk02   rootdg   online
enc0_3      sliced   disk05   rootdg   online
sena0_0     sliced   -        -        online
sena0_1     sliced   -        -        online
```

Note The phrase `error` 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 details on a particular disk defined to VxVM, use the following command:

```
# vxdisk list diskname
```

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, use the following procedure:

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

   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.

Administering Dynamic Multipathing (DMP) 3

Introduction

The Dynamic Multipathing (DMP) feature of VERITAS Volume Manager (VxVM) provides greater reliability and performance by using path failover and load balancing. This feature is available for multiported disk arrays from various vendors. See the *VERITAS Volume Manager Hardware Notes* for information about supported disk arrays.

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 type. DMP can also differentiate between different enclosures of a supported array type that are connected to the same host system.

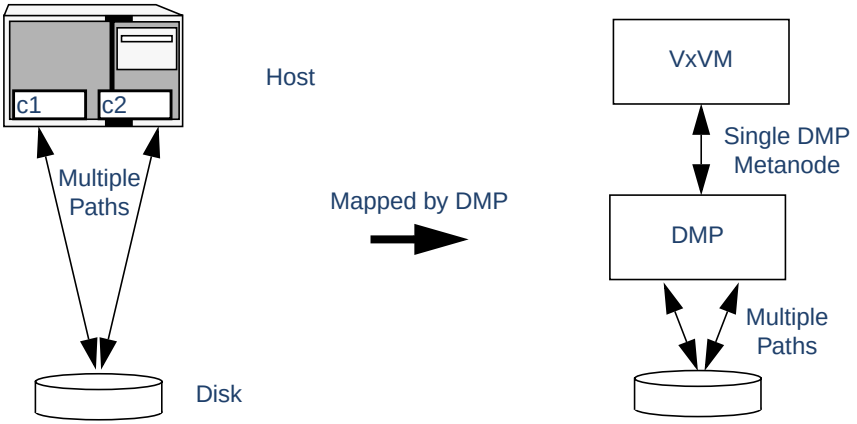
The multipathing policy used by DMP depends on the characteristics of the disk array:

- *Active/active* disk arrays (*A/A arrays*) permit several paths to be used concurrently for I/O. Such arrays enable DMP to provide greater I/O throughput by balancing the I/O load uniformly across the multiple paths to the disk devices. In the event of a loss of one connection to an array, DMP automatically routes I/O over the other available connections to the array.
- *Active/passive* arrays in auto-trespass mode (*A/P arrays*) allow I/O on the *primary* (active) path, and the *secondary* (passive) path is used if the primary path fails. Failover occurs when I/O is received or sent on the secondary path.
- Active/passive arrays in explicit failover mode (*A/PF arrays*) require a special command to be issued to the array for failover to occur.
- Active/passive arrays with LUN group failover (*A/PG arrays*) treat a group of LUNs that are connected through a controller as a single failover entity. Failover occurs at the controller level, and not at the LUN level (as would be the case for an A/P array in auto-trespass mode). The primary and 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's passive LUN group.

VxVM uses DMP metanodes to access disk devices connected to the system. For each disk in a supported array, DMP maps one metanode to the set of paths that are connected to the disk. Additionally, DMP associates the appropriate multipathing policy for the disk array with the metanode. For disks in an unsupported array, DMP maps a separate metanode to each path that is connected to a disk.

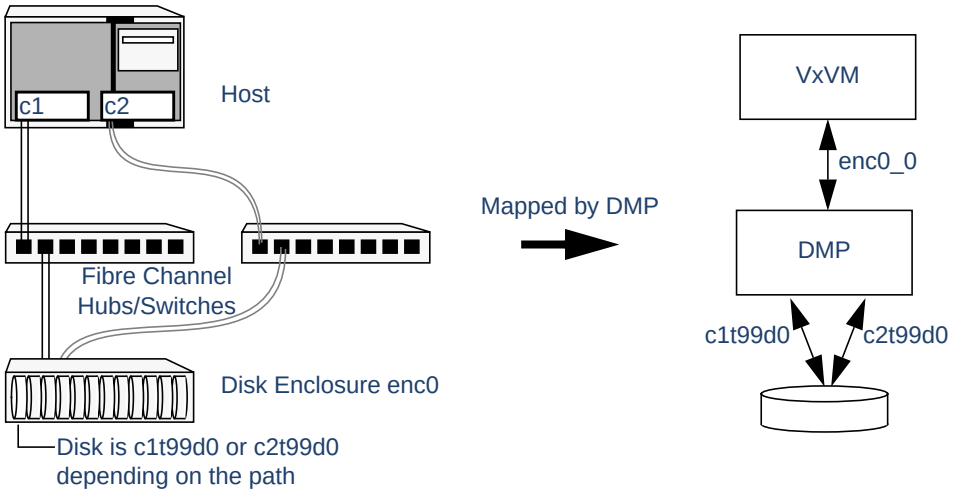
See the figure "How DMP Represents Multiple Physical Paths to a Disk as one Metanode" on page 96 for an illustration of how DMP sets up a metanode for a disk in a supported disk array.

How DMP Represents Multiple Physical Paths to a Disk as one Metanode



As described in "Enclosure-Based Naming" on page 6, VxVM implements a disk device naming scheme that allows you to easily recognize to which array a disk belongs. The figure, "Example of Multipathing for a Disk Enclosure in a SAN Environment" on page 97, shows that two paths, `c1t99d0` and `c2t99d0`, exist to a single disk in the enclosure, but VxVM uses the single DMP metanode, `enc0_0`, to access it.

Example of Multipathing for a Disk Enclosure in a SAN Environment



See “Changing the Disk-Naming Scheme” on page 61 for details of how to change the naming scheme that VxVM uses for disk devices.

Note The operation of DMP relies on the `vxdmp` device driver. Unlike prior releases, from VxVM 3.1.1 onwards, the `vxdmp` driver must always be present on the system.

See “Discovering and Configuring Newly Added Disk Devices” on page 57 for a description of how to make newly added disk hardware known to a host system.

Path Failover Mechanism

The DMP feature of VxVM enhances system reliability when used with multiported disk arrays. In the event of the loss of one connection to the disk array, DMP automatically selects the next available I/O path for I/O requests dynamically without action from the administrator.

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

Load Balancing

DMP uses the *balanced path mechanism* to provide load balancing across paths for active / active disk arrays. Load balancing maximizes I / O throughput by using the total bandwidth of all available paths. Sequential I / O starting within a certain range is sent down the same path in order to benefit from disk track caching. Large sequential I / O that does not fall within the range is distributed across the available paths to reduce the overhead on any one path.

For active / passive disk arrays, I / O is sent down the primary path. If the primary path fails, I / O is switched over to the other available primary paths or secondary paths. As the continuous transfer of ownership of LUNs from one controller to another results in severe I / O slowdown, load balancing across paths is not performed for active / passive disk arrays.

Note Both paths of an active / passive array are not considered to be on different controllers when mirroring across controllers (for example, when creating a volume using `vxassist make` specified with the `mirror=ctlr` attribute).

Dynamic Reconfiguration

Dynamic reconfiguration (DR) is a feature that is available on some high-end SUN Enterprise systems. The system board to be reconfigured contains disks controlled by VxVM (in addition to CPUs, memory, and other controllers or I / O boards) that can be taken offline while the system is still running. You can dynamically reconfigure your system using one of the relevant procedures described in the *VERITAS Volume Manager Hardware Notes*.

Booting From DMP Devices

When the root disk is placed under VxVM control, it is automatically accessed as a DMP device with one path if it is a single disk, or with multiple paths if the disk is part of a multiported disk array. By encapsulating the root disk, system reliability is enhanced against loss of one or more of the existing physical paths to a disk.

Disabling and Enabling Multipathing for Specific Devices

You can use `vxdiskadm` menu options 17 and 18 to disable or enable multipathing. These menu options also allow you to exclude devices from or include devices in the view of VxVM. For more information, see "Disabling Multipathing and Making Devices Invisible to VxVM," and "Enabling Multipathing and Making Devices Visible to VxVM" on page 104.

Disabling Multipathing and Making Devices Invisible to VxVM

Note Some of the operations described in this section require a reboot of the system.

Select menu task 17 (`Prevent multipathing/Suppress devices from VxVM's view`) from the `vxdiskadm` main menu to prevent a device from being multipathed by the VxVM DMP driver (`vxdmp`), or to exclude a device from the view of VxVM:

1. At the following prompt, confirm that you want to continue:

```
Exclude Devices
Menu: VolumeManager/Disk/ExcludeDevices

This operation might lead to some devices being suppressed from
VxVM's view or prevent them from being multipathed by vxdmp (This
operation can be reversed using the vxdiskadm command).
Do you want to continue ? [y,n,q,?] (default: y)
```

2. Select the operation you want to perform from the displayed list:

```
Exclude Devices
Menu: VolumeManager/Disk/ExcludeDevices

1  Suppress all paths through a controller from VxVM's view
2  Suppress a path from VxVM's view
3  Suppress disks from VxVM's view by specifying a VID:PID
   combination
4  Suppress all but one paths to a disk
5  Prevent multipathing of all disks on a controller by VxVM
6  Prevent multipathing of a disk by VxVM
7  Prevent multipathing of disks by specifying a VID:PID
   combination
8  List currently suppressed/non-multipathed devices

?  Display help about menu
?? Display help about the menuing system
q  Exit from menus

Select an operation to perform:
```


- Select option 1 to exclude all paths through the specified controller from the view of VxVM. These paths remain in the disabled state until the next reboot, or until the paths are re-included.

  ```
  Exclude controllers from VxVM
  Menu: VolumeManager/Disk/ExcludeDevices/CTLR-VXVM

  Use this operation to exclude all paths through a controller from
  VxVM.

  This operation can be reversed using the vxdiskadm command.

  You can specify a controller name at the prompt. A controller name
  is of the form c#, example c3, c11 etc. Enter 'all' to exclude all
  paths on all the controllers on the host. To see the list of
  controllers on the system, type 'list'.

  Enter a controller name:[ctlr_name,all,list,list-exclude,q,?]
  ```

- Select option 2 to exclude specified paths from the view of VxVM.

  ```
  Exclude paths from VxVM
  Menu: VolumeManager/Disk/ExcludeDevices/PATH-VXVM

  Use this operation to exclude one or more paths from VxVM.

  As a result of this operation, the specified paths will be excluded
  from the view of VxVM. This operation can be reversed using the
  vxdiskadm command.

  You can specify a pathname or a pattern at the prompt. Here are
  some path selection examples:

  all:          all paths
  c3t4d2:       a single path
  list:         list all paths on the system

  Enter a pathname or pattern:[<Pattern>,all,list,list-exclude,q?]
  ```

- Select option 3 to exclude disks from the view of VxVM that match a specified Vendor ID and Product ID.

Note This option requires a reboot of the system.

```
Exclude VID:PID from VxVM
Menu: VolumeManager/Disk/ExcludeDevices/VIDPID-VXVM

Use this operation to exclude disks returning a specified
VendorID:ProductID combination from VxVM.

As a result of this operation, all disks that return
VendorID:ProductID matching the specified combination will be
excluded from the view of VxVM. This operation can be reversed
using the vxdiskadm command.

You can specify a VendorID:ProductID pattern at the prompt. The
specification can be as follows:

VID:PID    where  VID stands for Vendor ID
                  PID stands for Product ID
(The command vxdmpinq in /etc/vx/diag.d can be used to obtain the
Vendor ID and Product ID)

Both VID and PID can have an optional '*' (asterisk) following
them.
If a '*' follows VID, it will result in the exclusion of all disks
returning Vendor ID starting with the specified VID. The same is
true for Product ID as well. Both VID and PID should be non NULL.
The maximum allowed lengths for VendorID and ProductID are 8 and 16
characters respectively.

Some examples of VID:PID specification are:

all       - Exclude all disks
aaa:123   - Exclude all disks having VID 'aaa' and PID '123'
aaa*:123  - Exclude all disks having VID starting with 'aaa'
              and PID '123'
aaa:123*  - Exclude all disks having VID 'aaa' and PID starting
              with '123'
aaa:*     - Exclude all disks having VID 'aaa' and any PID

Enter a VID:PID combination:[<Pattern>,all,list-exclude,q,?]
```


- Select option 4 to define a pathgroup for disks that are not multipathed by VxVM. (A pathgroup explicitly defines alternate paths to the same disk.) Only one path is made visible to VxVM.

  ```
  Exclude all but one paths to a disk
  Menu: VolumeManager/Disk/ExcludeDevices/PATHGROUP-VXVM

  Use this operation to exclude all but one paths to a disk. In case
  of disks which are not multipathed by vxdmp, VxVM will see each
  path as a disk. In such cases, creating a pathgroup of all paths to
  the disk will ensure that only one of the paths from the group is
  made visible to VxVM. The pathgroup can be removed using the
  vxdiskadm command.

  Example: If c1t30d0 and c2t30d0 are paths to the same disk and both
  are seen by VxVM as separate disks, c1t30d0 and c2t30d0 can be put
  in a pathgroup so that only one of these paths is visible to VxVM.

  The pathgroup can be specified as a list of blank separated paths,
  for example, c1t30d0 c2t30d0.

  Enter pathgroup: [<pattern>,list,list-exclude,q,?]
  ```

- Select option 5 to disable multipathing for all disks on a specified controller.

  ```
  Exclude controllers from DMP
  Menu: VolumeManager/Disk/ExcludeDevices/CTLR-DMP

  Use this operation to exclude all disks on a controller from being
  multipathed by vxdmp.

  As a result of this operation, all disks having a path through the
  specified controller will be claimed in the OTHER_DISKS category
  and hence, not multipathed by vxdmp. This operation can be reversed
  using the vxdiskadm command.

  You can specify a controller name at the prompt. A controller name
  is of the form c#, example c3, c11 etc. Enter 'all' to exclude all
  paths on all the controllers on the host. To see the list of
  controllers on the system, type 'list'.

  Enter a controller name:[<ctlr-name>,all,list,list-exclude,q,?]
  ```

- Select option 6 to disable multipathing for specified paths.

```
Exclude paths from DMP
Menu: VolumeManager/Disk/ExcludeDevices/PATH-DMP

Use this operation to exclude one or more disks from DMP.

As a result of this operation, the disks corresponding to the
specified paths will not be multipathed by DMP. This operation can
be reversed using the vxdiskadm command.

You can specify a pathname or a pattern at the prompt.  Here are
some path selection examples:

all:          all paths
c3t4d2:       a single path
list:         list all paths on the system

Enter a pathname or pattern :[<pattern>,all,list,list-exclude,q,?]
```

  If a path is specified, the corresponding disk are claimed in the OTHER_DISKS category and hence not multipathed.

- Select option 7 to disable multipathing for disks that match a specified Vendor ID and Product ID.

Note This option requires a reboot of the system.

```
Exclude VID:PID from DMP
Menu: VolumeManager/Disk/ExcludeDevices/VIDPID-DMP

Use this operation to prevent vxdmp from multipathing disks
returning a specific VID:PID combination.

As a result of this operation, all disks that return
VendorID:ProductID matching the specified combination will be
claimed in the OTHER DISKS category(i.e. they will not be
multipathed by vxdmp). This operation can be reversed using the
vxdiskadm command.

You can specify a VendorID:ProductID combination at the prompt. The
specification can be as follows:

VID:PID    where  VID stands for Vendor ID
                  PID stands for Product ID
(The command vxdmpinq in /etc/vx/diag.d can be used to obtain the
Vendor ID and Product ID)
```


```
Both VID and PID can have an optional '*' (asterisk) following
them.
If a '*' follows VID, it will result in the exclusion of all disks
returning Vendor ID starting with the specified VID. The same is
true for Product ID as well. Both VID and PID should be non NULL.
The maximum allowed lengths for Vendor ID and Product ID are 8 and
16 characters respectively.

Some examples of VID:PID specification are:

all       - Exclude all disks
aaa:123   - Exclude all disks having VID 'aaa' and PID '123'
aaa*:123  - Exclude all disks having VID starting with 'aaa'
              and PID '123'
aaa:123*  - Exclude all disks having VID 'aaa' and PID starting
              with '123'
aaa:*     - Exclude all disks having VID 'aaa' and any PID

Enter a VID:PID combination:[<pattern>,all,list,list-exclude,q,?]
```

All disks returning the specified Vendor ID and Product ID combination are claimed in OTHER_DISKS category and so are not multipathed.

Enabling Multipathing and Making Devices Visible to VxVM

Note Some of the operations described in this section require a reboot of the system.

Select menu item 18 (`Allow multipathing/Unsuppress devices from VxVM's view`) from the `vxdiskadm` main menu to re-enable multipathing for a device, or to make a device visible to VxVM again.

1. At the following prompt, confirm that you want to continue:

```
Include Devices
Menu: VolumeManager/Disk/IncludeDevices

The devices selected in this operation will become visible to VxVM
and/or will be multipathed by vxdmp again. Only those devices which
were previously excluded can be included again.
Do you want to continue ? [y,n,q,?] (default: y)
```

2. Select the operation you want to perform from the displayed list:

```
Volume Manager Device Operations
Menu: VolumeManager/Disk/IncludeDevices

1  Unsuppress all paths through a controller from VxVM's view
2  Unsuppress a path from VxVM's view
3  Unsuppress disks from VxVM's view by specifying a VID:PID
   combination
4  Remove a pathgroup definition
5  Allow multipathing of all disks on a controller by VxVM
6  Allow multipathing of a disk by VxVM
7  Allow multipathing of disks by specifying a VID:PID combination
8  List currently suppressed/non-multipathed devices

?  Display help about menu
?? Display help about the menuing system
q  Exit from menus

Select an operation to perform:
```

- Select option 1 to make all paths through a specified controller visible to VxVM.

```
Re-include controllers in VxVM
Menu: VolumeManager/Disk/IncludeDevices/CTLR-VXVM

Use this operation to make all paths through a controller visible
to VxVM again.

As a result of this operation, all paths through the specified
controller will be made visible to VxVM again.

You can specify a controller name at the prompt. A controller name
is of the form c#, example c3, c11 etc. Enter 'all' to exclude all
paths on all the controllers on the host. To see the list of
controllers on the system, type 'list'.

Enter a controller name:[<ctlr-name>,all,list,list-exclude,q,?]
```


- Select option 2 to make specified paths visible to VxVM.

```
Re-include paths in VxVM
Menu: VolumeManager/Disk/IncludeDevices/PATH-VXVM

Use this operation to make one or more paths visible to VxVM again.

As a result of this operation,the specified paths will become
visible to VxVM again.

You can specify a pathname or a pattern at the prompt. Here are
some path selection examples:

all:          all paths
c3t4d2:       a single path
list:         list all paths on the system

Enter a pathname or pattern :[<pattern>,all,list,list-exclude,q,?]
```

- Select option 3 to make disks visible to VxVM that match a specified Vendor ID and Product ID.

```
Make VID:PID visible to VxVM
Menu: VolumeManager/Disk/IncludeDevices/VIDPID-VXVM

Use this operation to make all disks returning a specified
VendorID:ProductID combination visible to VxVM again.

As a result of this operation, disks that return VendorID:ProductID
matching the specified combination will be made visible to VxVM
again.

You can specify a VID:PID combination at the prompt. The
specification can be as follows:

VID:PID    where  VID stands for Vendor ID
                  PID stands for Product ID
(The command vxdmpinq in /etc/vx/diag.d can be used to obtain the
Vendor ID and Product ID)

Both VID and PID can have an optional '*' (asterisk) following
them.
If a '*' follows VID, it will result in the inclusion of all disks
returning Vendor ID starting with VID. The same is true for Product
ID as well. Both VID and PID should be non NULL. The maximum
allowed lengths for Vendor ID and Product ID are 8 and 16
characters respectively.
```

```
Some examples of VID:PID specification are:

all       - Include all disks
aaa:123   - Include all disks having VID ‘aaa’ and PID ‘123’
aaa*:123  - Include all disks having VID starting with ‘aaa’
              and  PID ‘123’
aaa:123*  - Include all disks having VID ‘aaa’ and PID starting
              with ‘123’
aaa:*     - Include all disks having VID ‘aaa’ and any PID

Enter a VID:PID combination:[<pattern>,all,list,list-exclude,q,?]
```

All disks returning the specified Vendor ID and Product ID combination are made visible to VxVM.

- Select option 4 to remove 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.

```
Remove a pathgroup definition
Menu: VolumeManager/Disk/IncludeDevices/PATHGROUP-VXVM

Use this operation to remove the definition of pathgroup. Specify
the serial numbers of the pathgroups at the prompt. This can be
obtained by typing list-exclude at the prompt.

Enter pathgroup number(s):[<number>,list-exclude,q,?]
```

- Select option 5 to enable multipathing for all disks that have paths through the specified controller.

Note This option requires a reboot of the system.

```
Re-include controllers in DMP
Menu: VolumeManager/Disk/IncludeDevices/CTLR-DMP

Use this operation to make vxdmp multipath all disks on a
controller again.

As a result of this operation, all disks having a path through the
specified controller will be multipathed by vxdmp again.

You can specify a controller name at the prompt. A controller name
is of the form c#, example c3, c11 etc. Enter ’all’ to exclude all
paths on all the controllers on the host. To see the list of
controllers on the system, type ’list’.

Enter a controller name:[<ctlr-name>,all,list,list-exclude,q,?]
```


- Select option 6 to enable multipathing for specified paths.

Note This option requires a reboot of the system.

```
Re-include paths in DMP
Menu: VolumeManager/Disk/IncludeDevices/PATH-DMP

Use this operation to make vxdmp multipath one or more disks again.

As a result of this operation, all disks corresponding to the
specified paths will be multipathed by vxdmp again.

You can specify a pathname or a pattern at the prompt. Here are
some path selection examples:

all:          all paths
c3t4d2:       a single path
list:         list all paths on the system

Enter a pathname or pattern :[<pattern>,all,list,list-exclude,q,?]
```

- Select option 7 to enable multipathing for disks that match a specified Vendor ID and Product ID.

Note This option requires a reboot of the system.

```
Make VID:PID visible to DMP
Menu: VolumeManager/Disk/IncludeDevices/VIDPID-DMP

Use this operation to make vxdmp multipath disks returning a
specified VendorID:ProductID combination again.

As a result of this operation, all disks that return VID:PID
matching the specified combination will be multipathed by vxdmp
again.

You can specify a VID:PID combination at the prompt. The
specification can be as follows:

VID:PID    where  VID stands for Vendor ID
                  PID stands for Product ID
(The command vxdmpinq in /etc/vx/diag.d can be used to obtain the
Vendor ID and Product ID)

Both VID and PID can have an optional '*' (asterisk) following
them.
```

```
If a '*' follows VID, it will result in the inclusion of all disks
returning Vendor ID starting with the specified VID. The same is
true for Product ID as well. Both VID and PID should be non NULL.
The maximum allowed lengths for Vendor ID and Product ID are 8 and
16 characters respectively.

Some examples of VID:PID specification are:

all      - Exclude all disks
aaa:123  - Exclude all disks having VID 'aaa' and PID '123'
aaa*:123 - Exclude all disks having VID starting with 'aaa'
           and PID '123'
aaa:123* - Exclude all disks having VID 'aaa' and PID starting
           with '123'
aaa:*    - Exclude all disks having VID 'aaa' and any PID

Enter a VID:PID combination:[<pattern>,all,list-exclude,q,?]
```

Enabling and Disabling Input/Output (I/O) Controllers

DMP allows you to turn off I/O to a host I/O controller so that you can perform administrative operations. This feature can be used for maintenance of controllers attached to the host or of disk arrays supported by VxVM. I/O operations to the host I/O controller can be turned back on after the maintenance task is completed. You can accomplish these operations using the `vxdmpadm` command provided with VxVM.

In active/active type disk arrays, VxVM uses a balanced path mechanism to schedule I/O to multipathed disks. As a result, I/O may go through any available path at any given point in time. For example, if a system has a StorEdge A5000(TM) array and you need to change an A5000 interface board that is connected to this disk array, you can use the `vxdmpadm` command to list the host I/O controllers that are connected to the interface board. Disable the host I/O controllers to stop further I/O to the disks that are accessed through the interface board. You can then replace the board without causing disruption to any ongoing I/O to disks in the disk array.

In active/passive type disk arrays, VxVM schedules I/O to use the primary path until a failure is encountered. To change an interface card on the disk array or a card on the host (if supported by the hardware) that is connected to the disk array, disable I/O operations to the host I/O controllers. This shifts all I/O over to an active secondary path or to an active primary path on another I/O controller so that you can change the hardware.

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

Note VxVM does not allow you to disable the last active path to the root disk.

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.

For more information, see "Administering DMP Using vxdmpadm" on page 111.

Displaying Multipaths to a VM Disk

The `vxdisk` command is used to display the multipathing information for a particular metadevice. The metadevice is a device representation of a particular physical disk having multiple physical paths from the I/O controller of the system. In VxVM, all the physical disks in the system are represented as metadevices with one or more physical paths.

To view multipathing information for a particular metadevice, use the following command:

```
# vxdisk list devicename
```

For example, to view multipathing information for `c2t0d0s2`, use the following command:

```
# vxdisk list c2t0d0s2
```

Typical output is as follows:

```
Device     c2t0d0
devicetag  c2t0d0
type       sliced
hostid     aparajita
disk       name=disk01 id=861086917.1052.aparajita
group      name=rootdg id=861086912.1025.aparajita
flags      online ready autoconfig autoimport imported
pubpaths   block=/dev/vx/dmp/c2t0d0s4 char=/dev/vx/rdmp/c2t0d0s4
privpaths  block=/dev/vx/dmp/c2t0d0s3 char=/dev/vx/rdmp/c2t0d0s3
version    2.1
iosize     min=512 (bytes) max=2048 (blocks)
public     slice=4 offset=0 len=1043840
private    slice=3 offset=1 len=1119
update     time=861801175 seqno=0.48
headers    0 248
```

```
configs    count=1 len=795
logs       count=1 len=120
Defined regions
config     priv 000017-000247[000231]:copy=01 offset=000000 enabled
config     priv 000249-000812[000564]:copy=01 offset=000231 enabled
log        priv 000813-000932[000120]:copy=01 offset=000000 enabled
Multipathing information:
numpaths:       2
c2t0d0s2        state=enabled       type=primary
c1t0d0s2        state=disabled      type=secondary
```

In the `Multipathing information` section of this output, the `numpaths` line shows that there are 2 paths to the device, and the following two lines show that the path to `c2t0d0s2` is active (`state=enabled`) and that the other path `c1t0d0s2` has failed (`state=disabled`).

The `type` field is shown for disks on active / passive type disk arrays such as Nike, DG Clariion, and Hitachi DF350. 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 StorEdge A5000. Such arrays have no concept of primary and secondary paths.

Administering DMP Using vxdmpadm

The `vxdmpadm` utility is a command line administrative interface to the DMP feature of VxVM.

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

- Retrieve the name of the DMP device corresponding to a particular path
- List all paths under a DMP device
- List all controllers connected to disks attached to the host
- List all the paths connected to a particular controller
- Enable or disable a host controller on the system
- Rename an enclosure
- Control the operation of the DMP restore daemon

The following sections cover these tasks in detail along with sample output. For more information, 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=c3t2d1s2
```

The physical path can be specified as the `nodename` attribute, which must be a valid path listed in the `/dev/rdsk` directory.

The above command displays output such as the following:

```
NAME       STATE     ENCLR-TYPE PATHS  ENBL   DSBL   ENCLR-NAME
============================================================
c3t2d1s2   ENABLED   T300       2      2      0      enc0
```

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
============================================================
c2t1d0s2   ENABLED   T300       2      2      0      enc0
c2t1d1s2   ENABLED   T300       2      2      0      enc0
c2t1d2s2   ENABLED   T300       2      2      0      enc0
c2t1d3s2   ENABLED   T300       2      2      0      enc0
```

Displaying All Paths Controlled by a DMP Node

The following command displays the paths controlled by the specified DMP node:

```
# vxdmpadm getsubpaths dmpnodename=c2t1d0s2
```

```
NAME       STATE     PATH-TYPE   CTLR-NAME   ENCLR-TYPE   ENCLR-NAME
============================================================
c2t1d0s2   ENABLED   PRIMARY     c2          T300         enc0
c3t2d0s2   ENABLED   SECONDARY   c3          T300         enc0
```

The specified DMP node must be a valid node in the `/dev/vx/rdmp` directory.

You can also use `getsubpaths` to obtain all paths through a particular host disk controller:

```
# vxdmpadm getsubpaths ctlr=c2
```

```
NAME       STATE     PATH-TYPE   DMPNODENAME  ENCLR-TYPE ENCLR-NAME
============================================================
c2t1d0s2   ENABLED   PRIMARY     c2t1d0s2     T300       enc0
c2t1d1s2   ENABLED   PRIMARY     c2t1d1s2     T300       enc0
c2t1d2s2   ENABLED   SECONDARY   c2t1d2s2     T300       enc0
```

```
c2t1d3s2   ENABLED   SECONDARY   c2t1d3s2     T300        enc0
```

Listing Information About Host I/O Controllers

The following command lists attributes of all host I/O controllers on the system:

```
# vxdmpadm listctlr all

CTLR-NAME       ENCLR-TYPE      STATE      ENCLR-NAME
=====================================================
c0              OTHER           ENABLED    others0
c1              SEAGATE         ENABLED    seagate0
c2              T300            ENABLED    enc0
c3              T300            ENABLED    enc0
```

This form of the command lists controllers belonging to a specified enclosure and enclosure type:

```
# vxdmpadm listctlr enclosure=enc0 type=T300

CTLR-NAME       ENCLR-TYPE      STATE      ENCLR-NAME
=====================================================
c2              T300            ENABLED    enc0
c3              T300            ENABLED    enc0
```

Disabling a Controller

Disabling I/O to a host disk controller prevents DMP from issuing I/O through the specified controller. The command blocks until all pending I/O issued through the specified disk controller are completed.

To disable a controller, use the following command:

```
# vxdmpadm disable ctlr=ctlr
```

Before detaching a system board, stop all I/O to the disk controllers connected to the board. To do this, execute the `vxdmpadm disable` command, and then run the Dynamic Reconfiguration (DR) facility provided by Sun. Do this for every controller connected to the system board being detached. The disable operation fails if it is issued to a controller connected to the root disk through a single path. If there is a single path connected to a disk, the disable command fails with an error message. Use the `-f` option to forcibly disable the controller.

Enabling a Controller

Enabling a controller allows a previously disabled host disk controller to accept I/O. This operation succeeds only if the controller is accessible to the host and I/O can be performed on it. When connecting active/passive disk arrays in a non-clustered environment, 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.

To enable a controller, use the following command:

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

Listing Information About Enclosures

To display the attributes of a specified enclosure, use the following command:

```
# vxdmpadm listenclosure enc0
```

The following example displays all attributes associated with the enclosure named enc0:

```
ENCLR_NAME ENCLR_TYPE ENCLR_SNO                    STATUS
============================================================
enc0       T300       60020f20000001a90000        CONNECTED
```

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

```
# vxdmpadm listenclosure all
```

The following is example output from this command:

```
ENCLR_NAME  ENCLR_TYPE   ENCLR_SNO             STATUS
=========================================================
others0     OTHER        OTHER_DISKS           CONNECTED
seagate0    SEAGATE      SEAGATE_DISKS         CONNECTED
enc0        T300         60020f20000001a90000  CONNECTED
```

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 25 characters. The name must not contain an underbar character (_).

The following output from the command `vxdmpadm listenclosure all` shows the changed name.

```
ENCLR_NAME   ENCLR_TYPE   ENCLR_SNO              STATUS
=========================================================
others0      OTHER        OTHER_DISKS            CONNECTED
seagate0     SEAGATE      SEAGATE_DISKS          CONNECTED
GRP1         T300         60020f20000001a90000   CONNECTED
```

Starting the DMP Restore Daemon

The DMP restore daemon re-examines the condition of paths at a specified interval. The type of analysis it performs on the paths depends on the specified checking policy.

Note The DMP restore daemon does not change the disabled state of the path through a controller that you have disabled using `vxdmpadm disable`.

Use the start restore command to start the restore daemon and specify one of the following policies:

- `check_all`

  The restore daemon 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 start the restore daemon with this policy is:

  ```
  # vxdmpadm start restore policy=check_all [interval=seconds]
  ```

- `check_alternate`

  The restore daemon 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 start the restore daemon with this policy is:

  ```
  # vxdmpadm start restore policy=check_alternate [interval=seconds]
  ```


- `check_disabled`

  This is the default policy. The restore daemon checks the condition of paths that were previously disabled due to hardware failures, and revives them if they are back online. The command to start the restore daemon with this policy is:

  ```
  # vxdmpadm start restore policy=check_disabled [interval=seconds]
  ```

- `check_periodic`

  The restore daemon 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 are a large number of paths available. The command to start the restore daemon with this policy is:

  ```
  # vxdmpadm start restore policy=check_periodic interval=seconds \
    [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 restore daemon examines the paths. For example, after stopping the restore daemon, the polling interval can be set to 400 seconds using the following command:

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

The default interval is 300 seconds. Decreasing this interval can adversely affect system performance.

Note To change the interval or policy, you must first stop the restore daemon, and then restart it with new attributes.

See the `vxdmpadm`(1M) manual page for more information about DMP restore policies.

Stopping the DMP Restore Daemon

Use the following command to stop the DMP restore daemon:

```
# vxdmpadm stop restore
```

Note Automatic path failback stops if the restore daemon is stopped.

Displaying the Status of the DMP Restore Daemon

Use the following command to display the status of the automatic path restoration daemon, 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 Daemons

To display the number of error daemons that are running, use the following command:

```
# vxdmpadm stat errord
```


Creating and Administering Disk Groups 4

Introduction

This chapter describes how to create and manage *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.

A system with VERITAS Volume Manager (VxVM) installed has a default disk group configured, `rootdg`. By default, operations are directed to the `rootdg` disk group. As system administrator, you can create additional disk groups to arrange your system's disks for different purposes. Many systems do not use more than one disk group, unless they have a large number of disks. Disks are not added to disk groups until the disks are needed to create VxVM objects. Disks can be initialized, reserved, and added to disk groups later. However, at least one disk must be added to `rootdg` when you initially configure VxVM after installation.

When a disk is added to a disk group, it is given a name (for example, `disk02`). This name identifies a disk for volume operations: volume creation or mirroring. This name relates directly to the physical disk. If a physical disk is moved to a different target address or to a different controller, the name `disk02` continues to refer to it. Disks can be replaced 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 large disk groups can cause the private region to fill. In the case of larger disk groups, disks should be set up 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 takes up 256 bytes (or half a block), the number of records that can be created in a disk group can be estimated from the configuration database copy size. The copy size in blocks can be obtained from the output of the command `vxdg list` *diskgroup* as 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 amount of remaining free space in the configuration database is shown as the value of the `free` parameter. An example is shown in "Displaying Disk Group Information" on

page 121. 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 133 for details.

Caution Before making any changes to disk groups, use the commands `vxprint -hrm` and `vxdisk list` to record the current configuration.

Specifying a Disk Group to Commands

Many VxVM commands allow you to specify a disk group using the `–g` option. For example, to create a volume in disk group `mktdg`, use the following command:

```
# vxassist -g mktdg make mktvol 50m
```

The block special device for this volume is:

```
/dev/vx/dsk/mktdg/mktvol
```

The disk group does not have to be specified if the object names are unique. Most VxVM commands use object names specified on the command line to determine the disk group for the operation. For example, to create a volume on disk `mktdg01` without specifying the disk group name, use the following command:

```
# vxassist make mktvol 50m mktdg01
```

Many commands work this way as long as two disk groups do not have objects with the same names. For example, VxVM allows you to create volumes named `mktvol` in both `rootdg` and in `mktdg`. If you do this, you must add `–g mktdg` to any command where you want to manipulate the volume in the mktdg disk group.

Note Most VxVM commands require superuser or equivalent privileges.

Displaying Disk Group Information

To display information on existing disk groups, enter the following command:

```
# vxdg list
```

VxVM returns the following listing of current disk groups:

```
NAME    STATE     ID
rootdg  enabled   730344554.1025.tweety
newdg   enabled   731118794.1213.tweety
```

To display more detailed information on a specific disk group (such as `rootdg`), use the following command:

```
# vxdg list rootdg
```

The output from this command is similar to the following:

```
Group:     rootdg
dgid:      962910960.1025.bass
import-id: 0.1
flags:
version:   90
local-activation: read-write
detach-policy: local
copies:    nconfig=default nlog=default
config:    seqno=0.1183 permlen=727 free=722 templen=2 loglen=110
config disk c0t10d0 copy 1 len=727 state=clean online
config disk c0t11d0 copy 1 len=727 state=clean online
log disk c0t10d0 copy 1 len=110
log disk c0t11d0 copy 1 len=110
```

To verify the disk group ID and name 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 `c0t12d0` as follows:

```
Disk: c0t12d0
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 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
rootdg disk01   c0t10d0     c0t10d0     0       4444228  -
rootdg disk02   c0t11d0     c0t11d0     0       4443310  -
newdg  newdg01  c0t12d0     c0t12d0     0       4443310  -
newdg  newdg02  c0t13d0     c0t13d0     0       4443310  -
oradg  oradg01  c0t14d0     c0t14d0     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 default disk group, `rootdg`, use the following command:

```
# vxdg -g rootdg free
```

The following example output shows the amount of free space in sectors:

```
DISK    DEVICE      TAG         OFFSET  LENGTH   FLAGS
disk01  c0t10d0     c0t10d0     0       4444228  -
disk02  c0t11d0     c0t11d0     0       4443310  -
```

Creating a 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. Examples of this are:

- 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.

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 allows you to move only the application or user data that should be moved.

Disks must be placed in disk groups before VxVM can use the disks for volumes. VxVM always requires the `rootdg` disk group to be defined, but you can add more disk groups as required.

Note VxVM commands create all volumes in the default disk group, `rootdg`, if no alternative disk group is specified using the `-g` option (see "Specifying a Disk Group to Commands" on page 120). All commands default to the `rootdg` disk group unless the disk group can be deduced from other information such as a disk name.

A disk group must have at least one disk associated with it. A new disk group can be created when you use menu item `1 (Add or initialize one or more disks)` of the `vxdiskadm` command to add disks to VxVM control, as described in "Adding a Disk to VxVM" on page 65. Disks to be added to a disk group must not already belong to an existing disk group.

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

```
# vxdiskadd c1t1d0s2
```

where `c1t1d0s2` is the device name of a disk that is not currently assigned to a disk group.

Disk groups can also be created by using the command `vxdg init`:

```
# vxdg init diskgroup diskname=devicename
```

For example, to create a disk group named `mktdg` on device `c1t0d0s2`:

```
# vxdg init mktdg mktdg01=c1t0d0s2
```

The disk specified by the device name, `c1t0d0s2`, must have been previously initialized with `vxdiskadd` or `vxdiskadm`, and must not currently belong to a disk group.

Adding a Disk to a Disk Group

To add a disk to an existing disk group, use menu item `1 (Add or initialize one or more disks)` of the `vxdiskadm` command, as described in "Adding a Disk to VxVM" on page 65.

You can also use the `vxdiskadd` command to add a disk to a disk group, for example:

```
# vxdiskadd c1t2d0s2
```

where `c1t2d0s2` is the device name of a disk that is not currently assigned to a disk group.

Removing a Disk from a Disk Group

A disk that contains no subdisks can be removed from its disk group with this command:

```
# vxdg [-g groupname] rmdisk diskname
```

where the disk group name is only specified for a disk group other than the default, `rootdg`.

For example, to remove `disk02` from `rootdg`, use this command:

```
# vxdg rmdisk disk02
```

If the disk has subdisks on it when you try to remove it, the following error message is displayed:

```
vxdg:Disk diskname is used by one or more subdisks
```

Use the `-k` option to `vxdg` to remove device assignment. Using the `-k` option allows you to remove the disk even if subdisks are present. For more information, see the `vxdg`(1M) manual page.

Caution Use of the `-k` option to `vxdg` can result in data loss.

Once the disk has been removed from its disk group, you can (optionally) remove it from VxVM control completely, as follows:

```
# vxdisk rm devicename
```

For example, to remove `c1t0d0s2` from VxVM control, use these commands:

```
# vxdisk rm c1t0d0s2
```

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 `vxdiskadm` to remove a disk, you can choose to move volumes off that disk. To do this, run `vxdiskadm` and select item `3` (`Remove a disk`) from the main menu.

If the disk is used by some subdisks, this message is displayed:

```
The following subdisks currently use part of disk disk02:

    home usrvol

    Subdisks must be moved from disk02 before it can be removed.

    Move subdisks to other disks? [y,n,q,?] (default: n)
```

If you choose **y**, then all subdisks are moved off the disk, if possible. Some subdisks may not be movable. The most common reasons why a subdisk 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 subdisks, you may need to remove some plexes from some disks to free more space before proceeding with the disk removal operation.

Deporting a Disk Group

Deporting a disk group disables access to a disk group that is currently enabled (imported) by the system. Deport a disk group if you intend to move the disks in a disk group to another system. Also, deport a disk group if you want to use all of the disks remaining in a disk group for a new purpose.

To deport a disk group, use the following procedure:

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.

2. Use the following command to stop the volumes in the disk group:

   # **`vxvol -g`** ***`diskgroup`*** **`stopall`**

3. Select menu item `9 (Remove access to (deport) a disk group)` from the `vxdiskadm` main menu.

4. At the following prompt, enter the name of the disk group to be deported (in this example, `newdg`):

```
Remove access to (deport) a disk group
Menu: VolumeManager/Disk/DeportDiskGroup

Use this menu operation to remove access to a disk group that is
currently enabled (imported) by this system. Deport a disk
group if you intend to move the disks in a disk group to another
system. Also, deport a disk group if you want to use all of the
disks remaining in a disk group for some new purpose.

You will be prompted for the name of a disk group. You will also
be asked if the disks should be disabled (offlined). For
removable disk devices on some systems, it is important to
```


```
disable all access to the disk before removing the disk.
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:

```
The requested operation is to disable access to the removable
disk group named newdg. This disk group is stored on the
following disks:
    newdg01 on device c1t1d0

You can choose to disable access to (also known as "offline")
these disks. This may be necessary to prevent errors if
you actually remove any of the disks from the system.

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)
```

   Once the disk group is deported, the `vxdiskadm` utility displays the following message:

```
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)
```

Alternatively, you can use the `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.

To import a disk group, use the following procedure:

1. Use the following command to ensure that the disks in the deported disk group are online:

```
# vxdisk -s list
```

2. Select menu item `8 (Enable access to (import) a disk group)` from the `vxdiskadm` main menu.

3. At the following prompt, enter the name of the disk group to import (in this example, `newdg`):

```
Enable access to (import) a disk group
Menu: VolumeManager/Disk/EnableDiskGroup

Use this operation to enable access to a disk group. This can be
used as the final part of moving a disk group from one system to
another. The first part of moving a disk group is to use the
“Remove access to (deport) a disk group” operation on the
original host.

A disk group can be imported from another host that failed
without first deporting the disk group. Be sure that all disks
in the disk group are moved between hosts.

If two hosts share a SCSI bus, be very careful to ensure that
the other host really has failed or has deported the disk group.
If two active hosts import a disk group at the same time, the
disk group will be corrupted and will become unusable.

Select disk group to import [<group>,list,q,?] (default: list)
newdg
```

   Once the import is complete, the `vxdiskadm` utility displays the following success message:

```
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)
```

Alternatively, you can use the `vxdg` command to import a disk group:

```
# vxdg import diskgroup
```


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.

For example, because every system running VxVM must have a single `rootdg` default disk group, importing or deporting `rootdg` across systems is a problem. There cannot be two `rootdg` disk groups on the same system. This problem can be avoided by renaming the `rootdg` disk group during the 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 as *newdg* to the importing host. If the `-t` option is not used, the name change is permanent.

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.

To temporarily move the `rootdg` disk group from one host to another (for repair work on the root volume, for example) and then move it back, use the following procedure:

1. On the original host, identify the disk group ID of the `rootdg` disk group to be imported with the following command:

   ```
   # vxdisk -s list
   ```

   This command results in output such as the following:

   ```
   dgname: rootdg
   dgid:   774226267.1025.tweety
   ```

2. 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.

3. After the necessary work has been done on the imported `rootdg`, deport it back to its original host with this command:

```
# vxdg -h hostname deport diskgroup
```

where *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 `rootdg` from the importing host and returns locks to its original host. The original host then automatically imports its `rootdg` on the next reboot.

Moving Disks between Disk Groups

To move a disk between disk groups, remove the disk from one disk group and add it to the other. For example, to move the physical disk `c0t3d0` (attached with the disk name `disk04`) from disk group `rootdg` and add it to disk group `mktdg`, use the following commands:

```
# vxdg rmdisk disk04
# vxdg -g mktdg adddisk mktdg02=c0t3d0
```

Caution This procedure does not save the configurations nor data on the disks.

You can also move a disk by using the `vxdiskadm` command. Select item `3 (Remove a disk)` from the main menu, and then select item `1 (Add or initialize a disk)`.

See “Moving Objects Between Disk Groups” on page 140 for an alternative and preferred method of moving disks between disk groups. This method preserves VxVM objects, such as volumes, that are configured on the disks.

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, use the following procedure:

1. On the first system, stop all volumes in the disk group, then deport (disable local access to) the disk group with the following command:

   # **vxdg deport *diskgroup***

2. Move all the disks to the second system and perform the steps necessary (system-dependent) for the second 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.

3. Import (enable local access to) the disk group on the second system with this command:

   # **vxdg import *diskgroup***

Caution All disks in the disk group must be moved to the other system. If they are not moved, the import fails.

4. After the disk group is imported, start all volumes in the disk group with this 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 first system. When a disk group is created or imported on a system, that system writes a lock on all disks in the disk group.

Caution The purpose of the lock is to ensure that *dual-ported disks* (disks that can be accessed simultaneously by two systems) are not used by both systems at the same time. If two systems try to manage the same disks at the same time, configuration information stored on the disk is corrupted. The disk and its data become unusable.

When you move disks from a system that has crashed or failed to detect the group before the disk is moved, the locks stored on the disks remain and must be cleared. The system returns the following error message:

```
vxdg:disk group groupname: import failed: Disk is in use by
another host
```

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
```

Caution Be careful when using the `vxdisk clearimport` or `vxdg -C import` command on systems that have dual-ported disks. Clearing the locks allows those disks to be accessed at the same time from multiple hosts and can result in corrupted data.

You may want to import a disk group when some disks are not available. The `import` operation fails if some disks for the disk group cannot be found among the disk drives attached to the system. When the `import` operation fails, one of several error messages is displayed.

The following 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.

```
vxdg: Disk group groupname: import failed: Disk group has no valid
configuration copies
```

The following message indicates a recoverable error.

```
vxdg: Disk group groupname: import failed: Disk for disk group not
found
```

If some of the disks in the disk group have failed, force the disk group to be imported with the command:

```
# vxdg -f import diskgroup
```

Caution Be careful when using the `-f` option. It can cause the same disk group to be imported twice from different sets of disks, causing the disk group to become inconsistent.

These operations can also be performed using the `vxdiskadm` utility. To deport a disk group using `vxdiskadm`, select menu item `9 (Remove access to (deport) a disk group)`. To import a disk group, select item `8 (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 (or *collide*) 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.

To set a base volume device minor number for a disk group, use the following command:

```
# vxdg init diskgroup minor=base_minor devicename
```

VxVM chooses minor device numbers for objects created from this disk group starting at the number *base_minor*. Minor numbers can range from 0 up to 131,071. 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.

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.

For further information on minor number reservation, see the `vxdg`(1M) manual page.

Reorganizing the Contents of Disk Groups

Note You may need an additional license to use this feature.

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

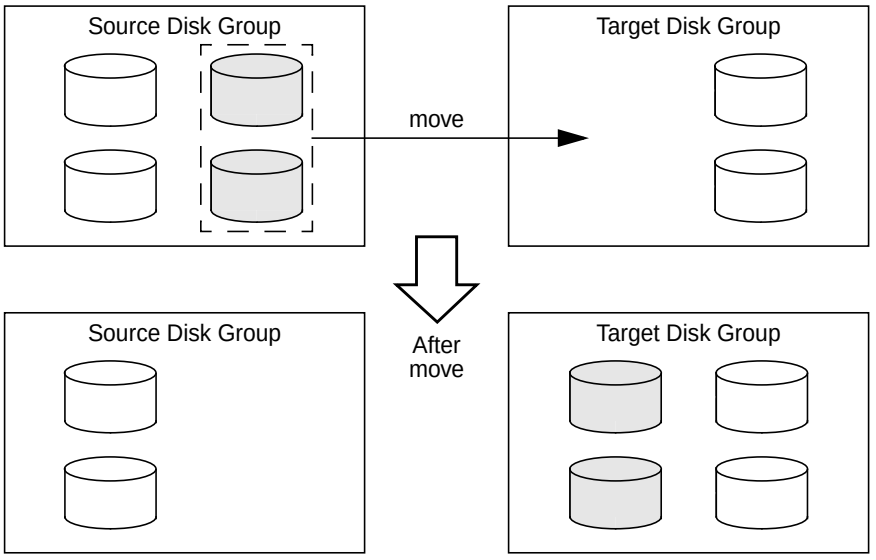
- To group volumes or disks differently as the needs of your organization change. For example, you might want to split disk groups to match the boundaries of separate departments, or to join disk groups when departments are merged.
- To reduce the size of a disk group's configuration database in the 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 (such as the Sun Netra ft 1800) that can be split into separate hosts for this purpose, and then rejoined.
- To implement off-host processing solutions for the purposes of backup or decision support in a cluster environment. This is discussed further in "Configuring Off-Host Processing" on page 285.

You can use either the VERITAS Enterprise Administrator (VEA) or the vxdg command to reorganize your disk groups. For more information about using the graphical user interface, see the *VERITAS Volume Manager (UNIX) User's Guide — VEA*. This section describes how to use the vxdg command.

The vxdg command provides the following operations for reorganizing disk groups:

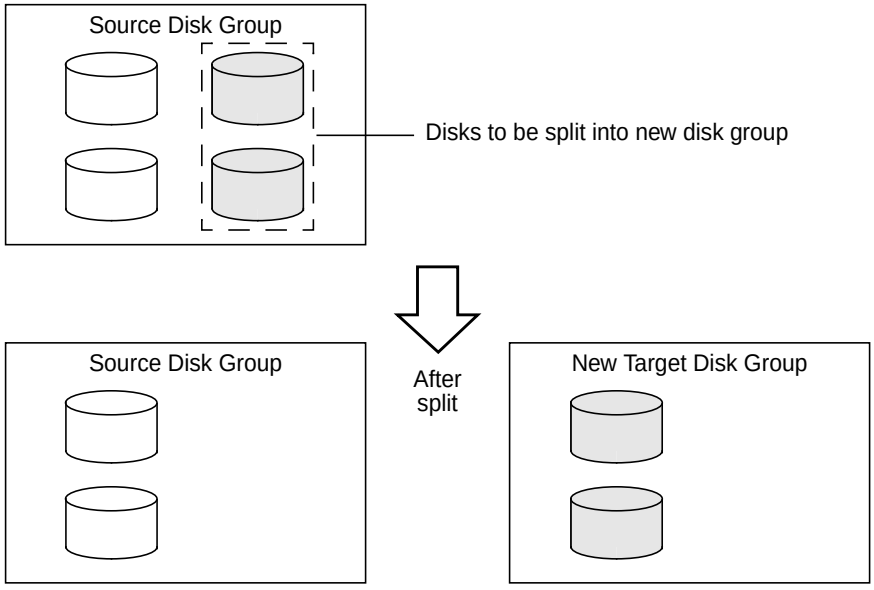
- `move`—moves a self-contained set of VxVM objects between imported disk groups. This operation fails if it would remove all the disks from the source disk group. Volume states are preserved across the move. The `move` operation is illustrated in "Disk Group Move Operation" below.

Disk Group Move Operation



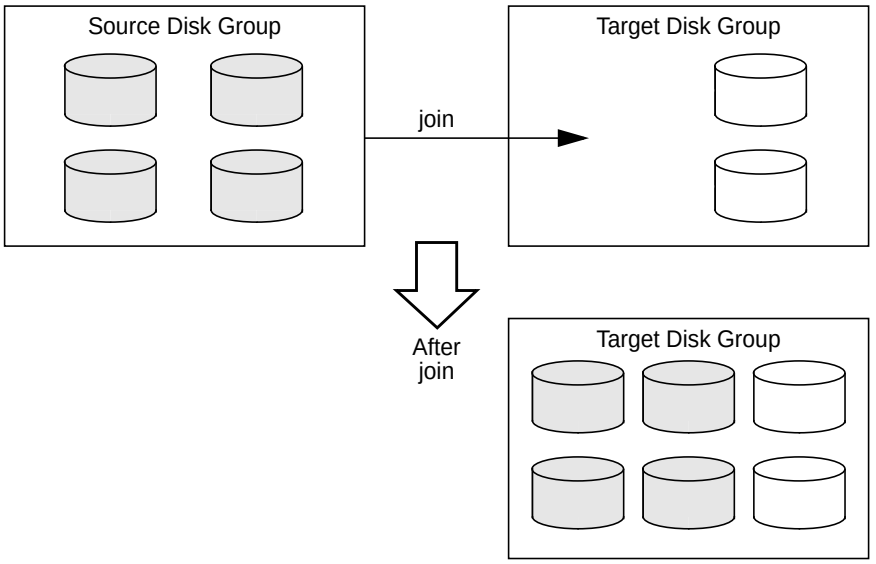
- `split`—removes a self-contained set of VxVM objects from an imported disk group, and moves them to a newly created target disk group. This operation fails if it would remove all the disks from the source disk group, or if an imported disk group exists with the same name as the target disk group. An 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). The `split` operation is illustrated in "Disk Group Split Operation" below.

Disk Group Split Operation



- `join`—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. The `join` operation is illustrated in "Disk Group Join Operation" below.

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.

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 are the self-contained set of objects that correspond to a specified set of objects.

Caution Before moving volumes between disk groups, stop all applications that are accessing the volumes, and unmount all file systems that are configured in the 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 as described in the section "Recovery from Incomplete Disk Group Moves" in the chapter "Recovery from Hardware Failure" of the *VERITAS Volume Manager Troubleshooting Guide*.

The disk group move, 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 a Disk Group" on page 145 for more information on disk group versions.
- The reconfiguration must involve an integral number of physical disks.
- Objects to be moved must not contain open volumes.
- Moved volumes are initially disabled following a disk group move, split or join. Use the `vxrecover -m` and `vxvol startall` commands to recover and restart the volumes.
- 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.

The following sections describe how to use the `vxdg` command to reorganize disk groups. For more information about the `vxdg` command, see the `vxdg`(1M) manual page.

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 `dg1` to `rootdg`:

```
# vxdg listmove dg1 rootdg vol1
disk01 c0t1d0s2 disk05 c1t96d0s2 vol1 vol1-01 vol1-02 disk01-01
disk05-01
```

However, the following command produces an error because only part of the volume `vol1` is configured on `disk01`:

```
# vxdg listmove dg1 rootdg disk01
vxvm:vxdg: ERROR: vxdg listmove dg1 rootdg failed
vxvm:vxdg: ERROR: disk05 : Disk not moving, but subdisks on it are
```

Specifying the `-o expand` option ensures that the list of objects encompasses other disks (in this case, `disk05`) that contain subdisks from `vol1`.

```
# vxdg -o expand listmove dg1 rootdg disk01
disk01 c0t1d0s2 disk05 c1t96d0s2 vol1 vol1-01 vol1-02 disk01-01
disk05-01
```

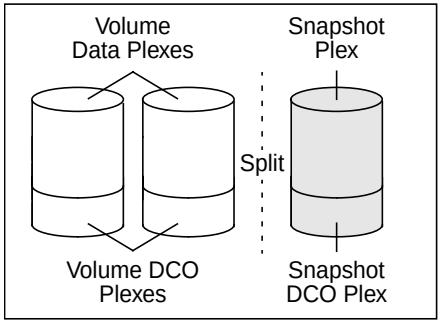
Considerations for Placing DCO Plexes

If you use the `vxassist` command or the VERITAS Enterprise Administrator (VEA) to create a volume, or to enable Persistent FastResync on a volume, the DCO plexes are automatically placed on the same disks as the data plexes of the parent volume. When you move the parent volume (such as a snapshot volume) to a different disk group, this ensures that the DCO volume automatically accompanies 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. Use the `vxprint` command on a volume to examine the configuration of its associated DCO volume.

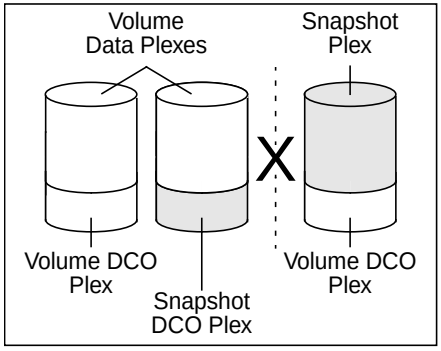
"Examples of Disk Groups That Can and Cannot be Split" on page 139 illustrates some instances in which it is not be possible to split a disk group because of the location of the DCO plexes.

For more information about relocating DCO plexes, see "Specifying Storage for DCO Plexes" on page 210.

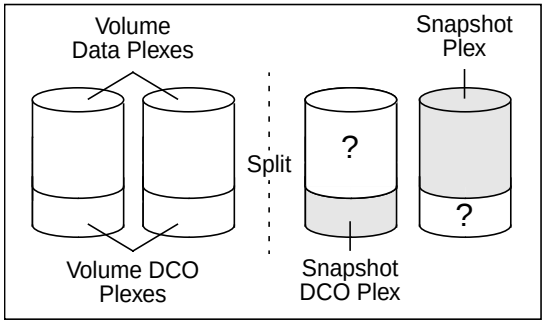
Examples of Disk Groups That Can and Cannot be Split



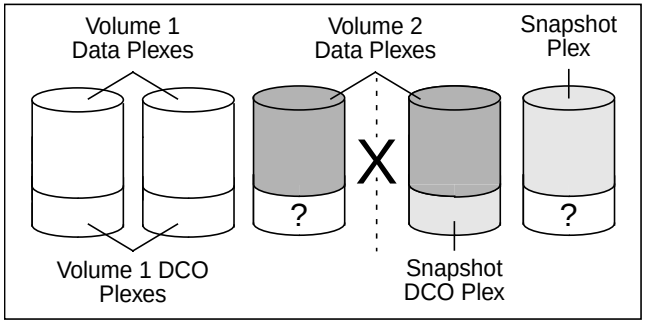
The disk group can be split as the DCO plexes are on the same disks as the data plexes and can therefore accompany their volumes.



The disk group cannot be split as the DCO plexes have been separated from their data plexes and so cannot accompany their volumes. One solution is to relocate the DCO plexes. In this example, it may be necessary to use an additional disk in the disk group as an intermediary to swap the plexes.



The disk group can be split as the DCO plexes can accompany their volumes even though they are on different disks. 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 that contain the data plexes of Volume 2. Possible solutions are to relocate the snapshot DCO plex to the disk containing the snapshot plex, 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 a 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.

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.

Note The `-o override` and `-o verify options` require a valid EMC license.

See "Moving Objects Between Disk Groups" on page 280 for information on how to move objects between disk groups in a cluster.

For example, the following output from `vxprint` shows the contents of disk groups `rootdg` and `dg1`:

```
# vxprint
Disk group: rootdg
TY NAME      ASSOC      KSTATE  LENGTH    PLOFFS  STATE  TUTIL0  PUTIL0
dg rootdg    rootdg     -       -         -       -      -       -
dm disk02    c1t97d0s2  -       17678493  -       -      -       -
dm disk03    c1t112d0s2 -       17678493  -       -      -       -
dm disk04    c1t114d0s2 -       17678493  -       -      -       -
dm disk06    c1t98d0s2  -       17678493  -       -      -       -

Disk group: dg1
TY NAME      ASSOC      KSTATE  LENGTH    PLOFFS  STATE  TUTIL0  PUTIL0
dg dg1       dg1        -       -         -       -      -       -
dm disk01    c0t1d0s2   -       17678493  -       -      -       -
dm disk05    c1t96d0s2  -       17678493  -       -      -       -
dm disk07    c1t99d0s2  -       17678493  -       -      -       -
dm disk08    c1t100d0s2 -       17678493  -       -      -       -
v  vol1      fsgen      ENABLED 2048      -       ACTIVE -       -
pl vol1-01   vol1       ENABLED 3591      -       ACTIVE -       -
```

```
sd disk01-01 vol1-01    ENABLED 3591     0       -      -       -
pl vol1-02   vol1       ENABLED 3591     -       ACTIVE -       -
sd disk05-01 vol1-02    ENABLED 3591     0       -      -       -
```

The following command moves the self-contained set of objects implied by specifying disk `disk01` from disk group `dg1` to `rootdg`:

```
# vxdg -o expand move dg1 rootdg disk01
```

The moved volumes are initially disabled following the 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 `disk01` but also volume `vol1` and `disk05` have moved to `rootdg`, leaving only `disk07` and `disk08` in disk group `dg1`:

```
# vxprint
Disk group: rootdg
TY NAME      ASSOC      KSTATE  LENGTH    PLOFFS  STATE  TUTIL0  PUTIL0
dg rootdg    rootdg     -       -         -       -      -       -
dm disk01    c0t1d0s2   -       17678493  -       -      -       -
dm disk02    c1t97d0s2  -       17678493  -       -      -       -
dm disk03    c1t112d0s2 -       17678493  -       -      -       -
dm disk04    c1t114d0s2 -       17678493  -       -      -       -
dm disk05    c1t96d0s2  -       17678493  -       -      -       -
dm disk06    c1t98d0s2  -       17678493  -       -      -       -
v  vol1      fsgen      ENABLED 2048      -       ACTIVE -       -
pl vol1-01   vol1       ENABLED 3591      -       ACTIVE -       -
sd disk01-01 vol1-01    ENABLED 3591      0       -      -       -
pl vol1-02   vol1       ENABLED 3591      -       ACTIVE -       -
sd disk05-01 vol1-02    ENABLED 3591      0       -      -       -

Disk group: dg1
TY NAME      ASSOC      KSTATE  LENGTH    PLOFFS  STATE  TUTIL0  PUTIL0
dg dg1       dg1        -       -         -       -      -       -
dm disk07    c1t99d0s2  -       17678493  -       -      -       -
dm disk08    c1t100d0s2 -       17678493  -       -      -       -
```

The following commands would also achieve the same result:

```
# vxdg move dg1 rootdg disk01 disk05
# vxdg move dg1 rootdg vol1
```


Splitting Disk Groups

To remove a self-contained set of VxVM objects from an imported source disk group to a new target disk group, use the following command:

```
# vxdg [-o expand] [-o override|verify] split sourcedg targetdg \
  object ...
```

For a description of the `-o expand`, `-o override`, and `-o verify` options, see "Moving Objects Between Disk Groups" on page 140.

See "Splitting Disk Groups" on page 280 for more information on splitting shared disk groups in clusters.

For example, the following output from `vxprint` shows the contents of disk group `rootdg`:

```
# vxprint
Disk group: rootdg
TY NAME      ASSOC      KSTATE  LENGTH    PLOFFS  STATE  TUTIL0  PUTIL0
dg rootdg    rootdg     -       -         -       -      -       -
dm disk01    c0t1d0s2   -       17678493  -       -      -       -
dm disk02    c1t97d0s2  -       17678493  -       -      -       -
dm disk03    c1t112d0s2 -       17678493  -       -      -       -
dm disk04    c1t114d0s2 -       17678493  -       -      -       -
dm disk05    c1t96d0s2  -       17678493  -       -      -       -
dm disk06    c1t98d0s2  -       17678493  -       -      -       -
dm disk07    c1t99d0s2  -       17678493  -       -      -       -
dm disk08    c1t100d0s2 -       17678493  -       -      -       -
v  vol1      fsgen      ENABLED 2048      -       ACTIVE -       -
pl vol1-01   vol1       ENABLED 3591      -       ACTIVE -       -
sd disk01-01 vol1-01    ENABLED 3591      0       -      -       -
pl vol1-02   vol1       ENABLED 3591      -       ACTIVE -       -
sd disk05-01 vol1-02    ENABLED 3591      0       -      -       -
```

The following command removes disks `disk07` and `disk08` from `rootdg` to form a new disk group, `dg1`:

```
# vxdg -o expand split rootdg dg1 disk07 disk08
```

The moved volumes are initially disabled following the split. Use the following commands to recover and restart the volumes in the new target disk group:

```
# vxrecover -g targetdg -m [volume ...]
# vxvol -g targetdg startall
```

The output from `vxprint` after the split shows the new disk group, `dg1`:

```
# vxprint
Disk group: rootdg
TY NAME      ASSOC      KSTATE  LENGTH    PLOFFS  STATE  TUTIL0  PUTIL0
dg rootdg    rootdg     -       -         -       -      -       -
dm disk01    c0t1d0s2   -       17678493  -       -      -       -
dm disk02    c1t97d0s2  -       17678493  -       -      -       -
dm disk03    c1t112d0s2 -       17678493  -       -      -       -
dm disk04    c1t114d0s2 -       17678493  -       -      -       -
dm disk05    c1t96d0s2  -       17678493  -       -      -       -
dm disk06    c1t98d0s2  -       17678493  -       -      -       -
v  vol1      fsgen      ENABLED 2048      -       ACTIVE -       -
pl vol1-01   vol1       ENABLED 3591      -       ACTIVE -       -
sd disk01-01 vol1-01    ENABLED 3591      0       -      -       -
pl vol1-02   vol1       ENABLED 3591      -       ACTIVE -       -
sd disk05-01 vol1-02    ENABLED 3591      0       -      -       -

Disk group: dg1
TY NAME      ASSOC      KSTATE  LENGTH    PLOFFS  STATE  TUTIL0  PUTIL0
dg dg1       dg1        -       -         -       -      -       -
dm disk07    c1t99d0s2  -       17678493  -       -      -       -
dm disk08    c1t100d0s2 -       17678493  -       -      -       -
```

Joining Disk Groups

To remove all VxVM objects from an imported source disk group to an imported target disk group, use the following command:

```
# vxdg [-o override|verify] join sourcedg targetdg
```

Note You cannot specify `rootdg` as the source disk group for a `join` operation.

For a description of the `-o override` and `-o verify` options, see "Moving Objects Between Disk Groups" on page 140.

See "Joining Disk Groups" on page 280 for information on joining disk groups in a cluster.

For example, the following output from `vxprint` shows the contents of the disk group `rootdg` and `dg1`:

```
# vxprint
Disk group: rootdg
TY NAME      ASSOC      KSTATE  LENGTH    PLOFFS  STATE  TUTIL0  PUTIL0
dg rootdg    rootdg     -       -         -       -      -       -
dm disk01    c0t1d0s2   -       17678493  -       -      -       -
dm disk02    c1t97d0s2  -       17678493  -       -      -       -
```


```
dm disk03    c1t112d0s2 -        17678493  -       -       -       -
dm disk04    c1t114d0s2 -        17678493  -       -       -       -
dm disk07    c1t99d0s2  -        17678493  -       -       -       -
dm disk08    c1t100d0s2 -        17678493  -       -       -       -

Disk group: dg1
TY NAME      ASSOC      KSTATE   LENGTH    PLOFFS  STATE  TUTIL0  PUTIL0
dg dg1       dg1        -        -         -       -      -       -
dm disk05    c1t96d0s2  -        17678493  -       -      -       -
dm disk06    c1t98d0s2  -        17678493  -       -      -       -
v  vol1      fsgen      ENABLED  2048      -       ACTIVE -       -
pl vol1-01   vol1       ENABLED  3591      -       ACTIVE -       -
sd disk01-01 vol1-01    ENABLED  3591      0       -      -       -
pl vol1-02   vol1       ENABLED  3591      -       ACTIVE -       -
sd disk05-01 vol1-02    ENABLED  3591      0       -      -       -
```

The following command joins disk group `dg1` to `rootdg`:

```
# vxdg join dg1 rootdg
```

The moved volumes are initially disabled following the 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 `dg1` has been removed:

```
# vxprint
Disk group: rootdg
TY NAME      ASSOC      KSTATE   LENGTH    PLOFFS  STATE  TUTIL0  PUTIL0
dg rootdg    rootdg     -        -         -       -      -       -
dm disk01    c0t1d0s2   -        17678493  -       -      -       -
dm disk02    c1t97d0s2  -        17678493  -       -      -       -
dm disk03    c1t112d0s2 -        17678493  -       -      -       -
dm disk04    c1t114d0s2 -        17678493  -       -      -       -
dm disk05    c1t96d0s2  -        17678493  -       -      -       -
dm disk06    c1t98d0s2  -        17678493  -       -      -       -
dm disk07    c1t99d0s2  -        17678493  -       -      -       -
dm disk08    c1t100d0s2 -        17678493  -       -      -       -
v  vol1      fsgen      ENABLED  2048      -       ACTIVE -       -
pl vol1-01   vol1       ENABLED  3591      -       ACTIVE -       -
sd disk01-01 vol1-01    ENABLED  3591      0       -      -       -
pl vol1-02   vol1       ENABLED  3591      -       ACTIVE -       -
sd disk05-01 vol1-02    ENABLED  3591      0       -      -       -
```

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
```

Caution 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.

Upgrading a Disk Group

Note This information is not applicable for platforms whose first release was VERITAS Volume Manager 3.0. However, it is applicable for subsequent releases.

Prior to the release of VERITAS Volume Manager 3.0, the disk group version was automatically upgraded (if needed) when the disk group was imported.

From release 3.0 of VERITAS Volume Manager, the two operations of importing a disk group and upgrading its version are separate. You can import a disk group from a previous version and use it without upgrading it.

When you want to use new features, the disk group can be upgraded. The upgrade is an explicit operation. Once the upgrade occurs, the disk group becomes incompatible with earlier releases of VxVM that do not support the new version.

Before the imported disk group is upgraded, no changes are made to the disk group to prevent its use on the release from which it was imported until you explicitly upgrade it to the current release.

Until completion of the upgrade, the disk group can be used “as is” provided there is no attempt to use the features of the current version. Attempts to use a feature of the current version that is not a feature of the version from which the disk group was imported results in an error message similar to this:

```
vxvm:vxedit: ERROR:  Disk group version doesn't support feature
```

To use any of the new features, you must run the `vxdg upgrade` command to explicitly upgrade the disk group to a version that supports those features.

All disk groups have a version number associated with them. VERITAS Volume Manager releases support a specific set of disk group versions. VxVM can import and perform operations on a disk group of that version. The operations are limited by what features and operations the disk group version supports.

The table, “Disk Group Version Assignments,” summarizes the VERITAS Volume Manager releases that introduce and support specific disk group versions:

Disk Group Version Assignments

VxVM Release	Introduces Disk Group Version	Supports Disk Group Versions
1.2	10	10
1.3	15	15
2.0	20	20
2.2	30	30
2.3	40	40
2.5	50	50
3.0	60	20-40, 60
3.1	70	20-70
3.1.1	80	20-80
3.2, 3.5	90	20-90

Importing the disk group of a previous version on a VERITAS Volume Manager 3.5 system prevents the use of features introduced since that version was released. The table, “Features Supported by Disk Group Versions,” summarizes the features that are supported by disk group versions 20 through 90:

Features Supported by Disk Group Versions

Disk Group Version	New Features Supported	Previous Version Features Supported
90	- Cluster Support for Oracle Resilvering - Disk Group Move, Split and Join - Device Discovery Layer (DDL) - Layered Volume Support in Clusters - Ordered Allocation - OS Independent Naming Support - Persistent FastResync	20, 30, 40, 50, 60, 70, 80
80	- VERITAS Volume Replicator (VVR) Enhancements	20, 30, 40, 50, 60, 70
70	- Non-Persistent FastResync - VERITAS Volume Replicator (VVR) Enhancements - Unrelocate	20, 30, 40, 50, 60
60	- Online Relayout - Safe RAID-5 Subdisk Moves	20, 30, 40
50	- SRVM (now known as VERITAS Volume Replicator or VVR)	20, 30, 40
40	- Hot-Relocation	20, 30
30	- VxSmartSync Recovery Accelerator	20
20	- Dirty Region Logging - Disk Group Configuration Copy Limiting, - Mirrored Volumes Logging - New-Style Stripes - RAID-5 Volumes - Recovery Checkpointing	

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
```

By default, VxVM creates a disk group of the highest version supported by the release. For example, VERITAS Volume Manager 3.5 creates disk groups with version 90.

It may sometimes be necessary to create a disk group for an older version. The default disk group version for a disk group created on a system running VERITAS Volume Manager 3.5 is 90. Such a disk group would not be importable on a system running VERITAS Volume Manager 2.3, which only supports up to version 40. Therefore, to create a disk group on a system running VERITAS Volume Manager 3.5 that can be imported by a system running VERITAS Volume Manager 2.3, the disk group must be created with a version of 40 or less.

To create a disk group with a previous version, specify the `-T` *version* option to the `vxdg init` command. For example, to create a disk group with version 40 that can be imported by a system running VxVM 2.3, use the following command:

```
# vxdg -T 40 init newdg newdg01=c0t3d0s2
```

This creates a disk group, `newdg`, which can be imported by VERITAS Volume Manager 2.3. Note that while this disk group can be imported on the VxVM 2.3 system, attempts to use features from VERITAS Volume Manager 3.0 and later releases will fail.

Managing 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 interface to the `vxconfigd` daemon.

You can use `vxdctl` to:

- control the operation of the `vxconfigd` daemon
- manage the initialization of the `rootdg` disk group configuration
- manipulate the contents of the `volboot` file which contains a list of disks that have `rootdg` disk group configuration databases

  If only simple disks exist in `rootdg`, the `vxconfigd` daemon cannot read the `rootdg` configuration without the existence of a `/etc/vx/volboot` file. The `volboot` file contains entries for disks that contain `rootdg` configuration databases. To add an entry for a disk to the `volboot` file, use the following command where *device* is the disk access name of the disk device to be added:

  ```
  # vxdctl add disk device
  ```

If your system is configured to use Dynamic Multipathing (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 directories `/dev/vx/dmp` and `/dev/vx/rdmp`
- 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

For more information about how to use `vxdctl`, refer to the `vxdctl`(1M) manual page.

Creating and Administering Subdisks 5

Introduction

This chapter describes how to create and maintain *subdisks*. Subdisks are the low-level building blocks in a VERITAS Volume Mananger (VxVM) configuration that are required to create plexes and volumes.

Note Most VxVM commands require superuser or equivalent privileges.

Creating Subdisks

Note Subdisks are created automatically if you use the `vxassist` command or the VERITAS Enterprise Administrator (VEA) to create volumes. For more information, see "Creating a Volume" on page 173.

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 `disk02-01` that starts at the beginning of disk `disk02` and has a length of 8000 sectors, use the following command:

```
# vxmake sd disk02-01 disk02,0,8000
```

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 154). 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 disk01-01 vol1-01  disk01  0        102400 0         c0t10d0 ENA
sd disk02-01 vol2-01  disk02  0        102400 0 c0t10d0ENA
```

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

```
# vxprint -l subdisk
```

For example, the following command displays all information for subdisk `disk02-01`:

```
# vxprint -l disk02-01
```

This command provides the following output:

```
Disk group: rootdg

Subdisk:  disk02-01
info:     disk=disk02 offset=0 len=205632
assoc:    vol=mvol plex=mvol-02 (offset=0)
flags:    enabled
device:   device=c2t0d1c0t10d0s2 path=/dev/vx/dmp/c2t0d1c0t10d0s4
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 mv old_subdisk new_subdisk [new_subdisk ...]
```

For example, if `disk03` is to be evacuated and `disk22` has enough room on two of its subdisks, use the following command:

```
# vxsd mv disk03-01 disk22-01 disk22-02
```

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.

See "Configuring Hot-Relocation to Use Only Spare Disks" on page 252 for information about manually relocating subdisks after hot-relocation.

Splitting Subdisks

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

```
# vxsd –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 `disk03-02`, with size 2000 megabytes into subdisks `disk03-02`, `disk03-03`, `disk03-04` and `disk03-05`, each with size 500 megabytes, use the following commands:

```
# vxsd -s 1000m split disk03-02 disk03-02 disk03-04
# vxsd -s 500m split disk03-02 disk03-02 disk03-03
# vxsd -s 500m split disk03-04 disk03-04 disk03-05
```

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 join subdisk1 subdisk2 ... new_subdisk
```


For example, to join the contiguous subdisks `disk03-02`, `disk03-03`, `disk03-04` and `disk03-05` as subdisk `disk03-02`, use the following command:

```
# vxsd join disk03-02 disk03-03 disk03-04 disk03-05 disk03-02
```

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 plex plex sd=subdisk,...
```

For example, to create the plex `home-1` and associates subdisks `disk02-01`, `disk02-00`, and `disk02-02` with plex `home-1`, use the following command:

```
# vxmake plex home-1 sd=disk02-01,disk02-00,disk02-02
```

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 `disk02-01` is first, `disk02-00` is second, and `disk02-02` 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 assoc plex subdisk1 [subdisk2 subdisk3 ...]
```

For example, to associate subdisks named `disk02-01`, `disk02-00`, and `disk02-02` with a plex named `home-1`, use the following command:

```
# vxsd assoc home-1 disk02-01 disk02-00 disk02-01
```

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 gaps between its 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 -l offset assoc sparse_plex exact_size_subdisk
```

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 -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.

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 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

```
# vxsd assoc vol01-01 disk10-01:0 disk11-01:1 disk12-01:2
```

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.

For a description of DRL, see “Dirty Region Logging (DRL)” on page 38, and “Dirty Region Logging (DRL) in Cluster Environments” on page 274. Log subdisks are ignored as far as the usual plex policies are concerned, and are only used to hold the dirty region log.

Note 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.

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

```
# vxsd 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 `disk02-01` with a plex named `vol01-02`, which is already associated with volume `vol01`, use the following command:

```
# vxsd aslog vol01-02 disk02-01
```

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

```
# vxassist 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 dis subdisk
```

For example, to dissociate a subdisk named `disk02-01` from the plex with which it is currently associated, use the following command:

```
# vxsd dis disk02-01
```

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

```
# vxsd -o rm dis subdisk
```

Caution 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 rm subdisk
```

For example, to remove a subdisk named `disk02-01`, use the following command:

```
# vxedit rm disk02-01
```

Changing Subdisk Attributes

Caution 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 set attribute=value ... subdisk ...
```

Subdisk fields that can be changed using the `vxedit` command include:

- `name`
- `putil`*n*
- `tutil`*n*
- `len`
- `comment`

The `putil`*n* field attributes are maintained on reboot; `tutil`*n* fields are temporary and are not retained on reboot. VxVM sets the `putil0` and `tutil0` utility fields. Other VERITAS products, such as the VERITAS Enterprise Administrator (VEA), set the `putil1` and `tutil1` fields. The `putil2` and `tutil2` are available for you to use for site-specific purposes. The length field, `len`, can only be changed if the subdisk is dissociated.

For example, to change the comment field of a subdisk named `disk02-01`, use the following command:

```
# vxedit set comment="subdisk comment" disk02-01
```

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 set putil0="DO-NOT-USE" disk02-01
```

See the `vxedit`(1M) manual page for more information about using the `vxedit` command to change the attribute fields of VxVM objects.

Creating and Administering Plexes 6

Introduction

This chapter describes how to create and maintain *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.

Note Most VxVM commands require superuser or equivalent privileges.

Creating Plexes

Note Plexes are created automatically if you use the `vxassist` command or the VERITAS Enterprise Administrator (VEA) to create volumes. For more information, see "Creating a Volume" on page 173.

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` using two existing subdisks named `disk02-01` and `disk02-02`, use the following command:

```
# vxmake plex vol01-02 sd=disk02-01,disk02-02
```


Creating a Striped Plex

To create a striped plex, you must specify additional attributes. For example, to create a striped plex named `pl-01` with a stripe width of 32 sectors and 2 columns, use the following command:

```
# vxmake plex pl-01 layout=stripe stwidth=32 ncolumn=2 \
sd=disk01-01,disk02-01
```

To use a plex to build a volume, you must associate the plex with the volume. For more information, see the section, "Attaching and Associating Plexes" on page 165.

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 -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.

Note 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. For more information about the possible transitions between plex states and how these are applied during volume recovery, see the chapter "Understanding the Plex State Cycle" in the section "Recovery from Hardware Failure" in the *VERITAS Volume Manager Troubleshooting Guide*.

Plexes that are associated with a volume have one of the following states:

ACTIVE Plex State

A plex can be in the ACTIVE state in two 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.

Note On a system running well, ACTIVE should be the most common state you see for any volume plexes.

CLEAN Plex State

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 Plex State

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 Plex State

Volume creation sets all plexes associated with the volume to the EMPTY state to indicate that the plex is not yet initialized.

IOFAIL Plex State

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 Plex State

The state of a dirty region logging (DRL) or RAID-5 log plex is always set to LOG.

OFFLINE Plex State

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 Plex State

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 Plex State

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 for more information.

SNAPDONE Plex State

The SNAPDONE plex state indicates that a snapshot plex is ready for a snapshot to be taken using `vxassist snapshot`.

SNAPTMP Plex State

The SNAPTMP plex state is used during a `vxassist snapstart` operation when a snapshot is being prepared on a volume.

STALE Plex State

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 Plex State

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.

TEMPRM Plex State

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 Plex State

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

`vxprint` may also display one of the following condition flags in the STATE field:

IOFAIL Plex Condition

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 Plex Condition

No physical disk could 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 the condition is fixed or the affected subdisk is dissociated.

NODEVICE Plex Condition

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.

RECOVER Plex Condition

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 Plex Condition

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.

Note No user intervention is required to set these states; they are maintained internally. On a system that is operating properly, all plexes are enabled.

The following plex kernel states are defined:

DETACHED Plex Kernel State

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 Plex Kernel State

The plex is offline and cannot be accessed.

ENABLED Plex Kernel State

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.

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`, use the following command:

```
# vxplex att vol01 vol01-02
```

If the volume does not already exist, a plex (or multiple plexes) can be associated with the volume when it is created 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 -U fsgen vol home plex=home-1,home-2
```

Note 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 off plex
```

If a disk has a head crash, put all plexes that have associated subdisks on the affected disk OFFLINE. For example, if plexes vol01-02 and vol02-02 had subdisks on a drive to be repaired, use the following command to take these plexes offline:

```
# vxmend 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 det plex
```

For example, to temporarily detach a plex named vol01-02 and place it in maintenance mode, use the following command:

```
# vxplex 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, follow the procedures in the following section, "Reattaching Plexes."

Reattaching Plexes

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 att volume plex ...
  ```

  For example, for a plex named `vol01-02` on a volume named `vol01`, use the following command:

  ```
  # vxplex 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 revive 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 on plex
  ```

  For example, to re-enable a plex named `vol01-02`, enter:

  ```
  # vxmend 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 (see "Listing Unstartable Volumes" in the section "Recovery from Hardware Failure" in the *VERITAS Volume Manager Troubleshooting Guide*), set one of the plexes to CLEAN using the following command:

  ```
  # vxmend fix clean plex
  ```

  Start the volume using the following command:

  ```
  # vxvol start volume
  ```


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 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 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 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

Caution 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 160" for more information.

To dissociate a plex from the associated volume and remove it as an object from VxVM, use the following command:

```
# vxplex -o rm dis plex
```

For example, to dissociate and remove a plex named `vol01-02`, use the following command:

```
# vxplex -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 dis plex
# vxedit -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

Caution 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 set attribute=value ... plex
```

Plex fields that can be changed using the `vxedit` command include:

- `name`
- `putil`*n*
- `tutil`*n*
- `comment`

The `putil`*n* field attributes are maintained on reboot; `tutil`*n* fields are temporary and are not retained on reboot. VxVM sets the `putil0` and `tutil0` utility fields. Other VERITAS products, such as the VERITAS Enterprise Administrator (VEA), set the `putil1` and `tutil1` fields. The `putil2` and `tutil2` are available for you to use for site-specific purposes.

The following example command sets the comment field, and also sets `tutil2` to indicate that the subdisk is in use:

```
# vxedit 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 set putil0="DO-NOT-USE" vol01-02
```

See the `vxedit`(1M) manual page for more information about using the `vxedit` command to change the attribute fields of VxVM objects.

Creating Volumes 7

Introduction

This chapter describes how to create *volumes* in VERITAS Volume Manager (VxVM). 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.

Note Disks and disk groups must be initialized and defined to VxVM before volumes can be created from them. See "Administering Disks" on page 53 and "Creating and Administering Disk Groups" on page 119 for more information.

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. For more information, see "Concatenation and Spanning" on page 15.
- 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. For more information, see "Striping (RAID-0)" on page 17.

- 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. For more information, see "Mirroring (RAID-1)" on page 21.
- 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 accessed. However, in comparison to mirroring, the use of parity to implement data redundancy reduces the amount of space required. For more information, see "RAID-5 (Striping with Parity)" on page 24.
- *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. "Striping Plus Mirroring (Mirrored-Stripe or RAID-0+1)" on page 21.
- 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. For more information, see "Layered Volumes" on page 29.

  Examples of layered volumes are *striped-mirror* and *concatenated-mirror* volumes.

Note The VERITAS Enterprise Administrator (VEA) terms a striped-mirror volume as `Striped-Pro`, and a concatenated- mirror volume as `Concatenated-Pro`.

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. For more information, see "Mirroring Plus Striping (Striped-Mirror, RAID-1+0 or RAID-10)" on page 22.

A concatenated-mirror volume is created by concatenating several mirrored volumes. This provides faster recovery as the failure of a single disk does not force the entire mirror offline.

Creating a Volume

You can create volumes using either an *advanced* approach or an *assisted* approach. Each method uses different tools although you may switch from one set to another 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.

The steps to create a volume using this approach are:

1. Create subdisks using `vxmake sd`; see "Creating Subdisks" on page 151.

2. Create plexes using `vxmake plex`, and associate subdisks with them; see "Creating Plexes" on page 159, "Associating Subdisks with Plexes" on page 154 and "Creating a Volume Using vxmake" on page 191.

3. Associate plexes with the volume using `vxmake vol`; see "Creating a Volume Using vxmake" on page 191.

4. Initialize the volume using `vxvol start` or `vxvol init zero`; see "Initializing and Starting a Volume" on page 194.

See "Creating a Volume Using a vxmake Description File" on page 193 for an example of how you can combine steps 1 through 3 using a volume description file with `vxmake`.

See "Creating a Volume Using vxmake" on page 191 for an example of how to perform steps 2 and 3 to create a RAID-5 volume.

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 or the VERITAS Enterprise Administrator (VEA). `vxassist` and the VEA create the required plexes and subdisks using only the basic attributes of the desired volume as input. Additionally, they can modify existing volumes while automatically modifying any underlying or associated objects.

Both `vxassist` and the VEA use default values for many volume attributes, unless you provide specific values. They do not require you to have a thorough understanding of low-level VxVM concepts, `vxassist` and the VEA do not conflict with other VxVM commands or preclude their use. Objects created by `vxassist` and the VEA are compatible and inter-operable with objects created by other VxVM commands and interfaces.

For more information about the VEA, see the *VERITAS Volume Manager (UNIX) User's Guide — VEA*.

Using vxassist

You can use the `vxassist` command 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`.

`vxassist` 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 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 for more information on specifying units.

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.

Note By default, the `vxassist` command creates volumes in the `rootdg` disk group. To use a different disk group, specify the `-g` *diskgroup* option to `vxassist`.

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

The section, "Creating a Volume on Any Disk" on page 177 describes the simplest way to create a volume with default attributes. Later sections describe how to create volumes with specific attributes. For example, "Creating a Volume on Specific Disks" on page 178 describes how to control how `vxassist` uses the available storage space.

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.

Note 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. Refer to the `vxassist`(1M) manual page for details.

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
```

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 178 for more information.

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.

Note 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
```

Note Specify the `-b` option if you want to make the volume immediately available for use. See “Initializing and Starting a Volume” on page 194 for details.

For example, to create the concatenated volume `voldefault` with a length of 10 gigabytes in the `rootdg` 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 ...
```

Note Specify the `-b` option if you want to make the volume immediately available for use. See "Initializing and Starting a Volume" on page 194 for details.

For example, to create the volume `volspec` with length 5 gigabytes on `disk03` and `disk04`, use the following command:

```
# vxassist -b make volspec 5g disk03 disk04
```

The `vxassist` command allows you to specify storage attributes. These give you fine control over the devices, including disks, controllers and targets, which `vxassist` uses to configure a volume. For example, you can specifically exclude disk05:

```
# vxassist -b make volspec 5g !disk05
```

or exclude all disks that are on controller c2:

```
# vxassist -b make volspec 5g !ctlr:c2
```

or include only disks on controller `c1` except for target `t5`:

```
# vxassist -b make volspec 5g ctlr:c1 !target:c1t5
```

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 -b -g bigone make volmega 20g disk10 disk11
```

or alternatively, use the `diskgroup` attribute:

```
# vxassist -b make volmega 20g diskgroup=bigone disk10 disk11
```

Note 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 `disk07` and `disk08` 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 !disk07 !disk08
```

See the vxassist(1M) manual page for more information about using storage attributes. It is also possible to control how volumes are laid out on the specified storage as described in the next section "Specifying Ordered Allocation of Storage to Volumes."

If you are using VxVM in conjunction with VERITAS SANPoint Control 2.0, you can specify how vxassist should use the available storage groups when creating volumes. See "Configuring Volumes on SAN Storage" on page 49 and the vxassist(1M) manual page for more information.

Specifying Ordered Allocation of Storage to Volumes

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:

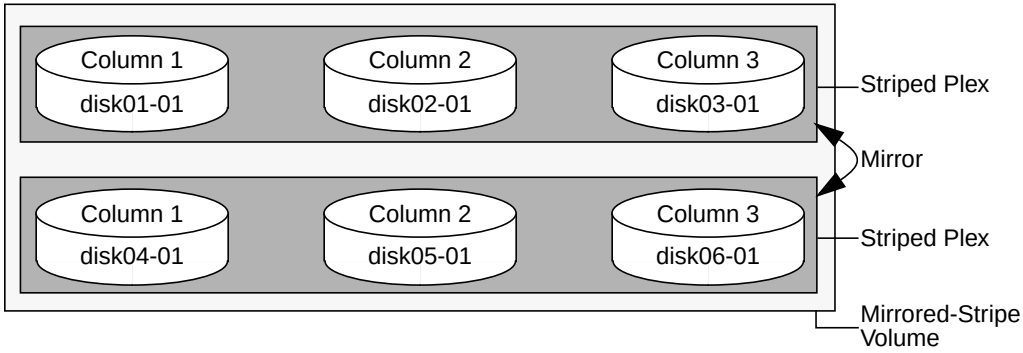
1. Concatenate disks.
2. Form columns.
3. Form mirrors.

For example, the following command creates a mirrored-stripe volume with 3 columns and 2 mirrors on 6 disks:

```
# vxassist -b -o ordered make mirstrvol 10g layout=mirror-stripe \
  ncol=3 disk01 disk02 disk03 disk04 disk05 disk06
```

This command places columns 1, 2 and 3 of the first mirror on disk01, disk02 and disk03 respectively, and columns 1, 2 and 3 of the second mirror on disk04, disk05 and disk06 respectively. This arrangement is illustrated in "Example of Using Ordered Allocation to Create a Mirrored-Stripe Volume."

Example of Using Ordered Allocation to Create a Mirrored-Stripe Volume

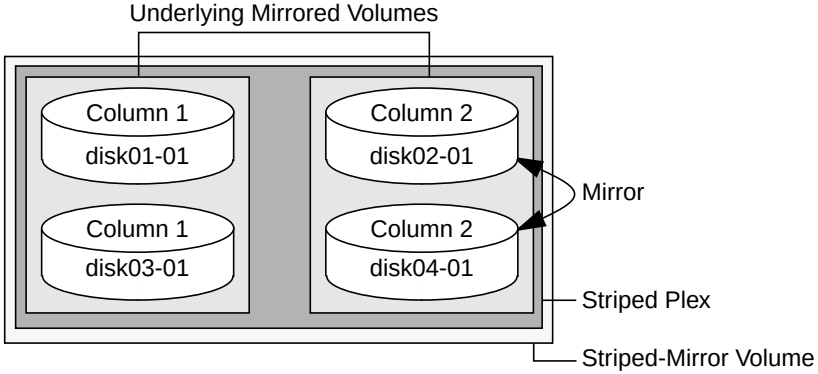


For layered volumes, `vxassist` applies the same rules to allocate storage as for non-layered volumes. For example, the following command creates a striped-mirror volume with 2 columns:

```
# vxassist -b -o ordered make strmirvol 10g layout=stripe-mirror \
  ncol=2 disk01 disk02 disk03 disk04
```

This command mirrors column 1 across `disk01` and `disk03`, and column 2 across `disk02` and `disk04` as illustrated in “Example of using Ordered Allocation to Create a Striped-Mirror Volume” on page 180.

Example of using Ordered Allocation to Create a Striped-Mirror Volume

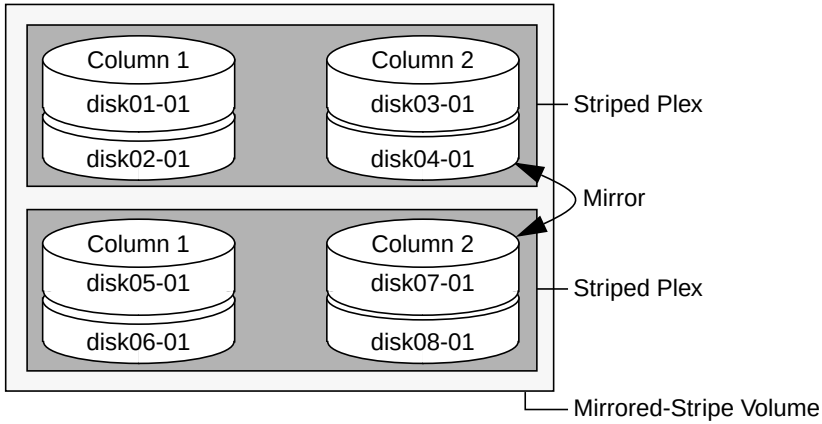


Additionally, you can use the `col_switch` attribute to specify how to concatenate space on the disks into columns. For example, the following command creates a mirrored-stripe volume with 2 columns:

```
# vxassist -b -o ordered make strmir2vol 10g layout=mirror-stripe \
  ncol=2 col_switch=3g,2g disk01 disk02 disk03 disk04 \ disk05
disk06 disk07 disk08
```

This command allocates 3 gigabytes from `disk01` and 2 gigabytes from `disk02` to column 1, and 3 gigabytes from `disk03` and 2 gigabytes from `disk04` to column 2. The mirrors of these columns are then similarly formed from disks `disk05` through `disk08`. This arrangement is illustrated in "Example of Using Concatenated Disk Space to Create a Mirrored-Stripe Volume."

Example of Using Concatenated Disk Space to Create a Mirrored-Stripe Volume

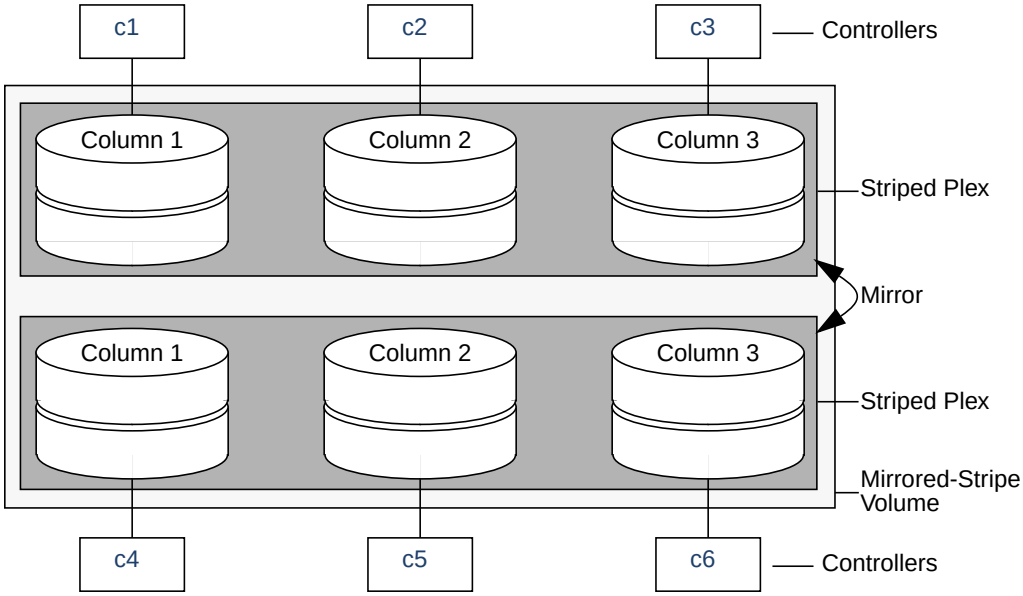


Other storage specification classes for controllers, enclosures, targets and trays can be used with ordered allocation. For example, the following command creates a 3-column mirrored-stripe volume between specified controllers:

```
# vxassist -b -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 as illustrated in"Example of Storage Allocation Used to Create a Mirrored-Stripe Volume Across Controllers."

Example of Storage Allocation Used to Create a Mirrored-Stripe Volume Across Controllers



For other ways in which you can control how `vxassist` lays out mirrored volumes across controllers, see “Mirroring across Targets, Controllers or Enclosures” on page 188.

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.

Note 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.

To create a new mirrored volume, use the following command:

```
# vxassist [-b] [-g diskgroup] make volume length layout=mirror \
  [nmirror=number] [init=active]
```

Note Specify the `-b` option if you want to make the volume immediately available for use. See “Initializing and Starting a Volume” on page 194 for details.

For example, to create the mirrored volume, `volmir`, use the following command:

```
# vxassist -b 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 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]
```

Note Specify the `-b` option if you want to make the volume immediately available for use. See “Initializing and Starting a Volume” on page 194 for details.

Alternatively, first create a concatenated volume, and then mirror it as described in “Adding a Mirror to a Volume” on page 205.

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]
```

Note Specify the `-b` option if you want to make the volume immediately available for use. See "Initializing and Starting a Volume" on page 194 for details.

Creating a Volume with a DCO and 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 43 for details of how Persistent FastResync performs fast resynchronization of snapshot mirrors when they are returned to their original volume.)

To perform fast resynchronization of mirrors after a system crash or reboot, you must also enable dirty region logging (DRL) on a mirrored volume. To add a DCO object and DCO volume to a volume on which DRL logging is enabled, follow the procedure described in "Adding a DCO and DCO Volume" on page 207.

Note You may need an additional license to use the Persistent FastResync feature. Even if you do not have a license, you can configure a DCO object and DCO volume so that snap objects are associated with the original and snapshot volumes. For more information about snap objects, see "How Persistent FastResync Works with Snapshots" on page 43.

Dirty region logging (DRL) is the default log type if you specify the `log` attribute to enable logging on a mirrored volume, but do not use the `logtype` attribute to specify the type of logging to `vxassist`.

Note Only one type of logging may initially be specified when you use `vxassist` to create a volume.

To create a volume with an attached DCO object and DCO volume, use the following procedure:

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***

   For more information, see "Upgrading a Disk Group" on page 145.

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]

   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 also 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.

For more information about configuring DCO, see the `vxassist`(1M) manual page.

Creating a Mirrored Volume with DRL Logging Enabled

To create a mirrored volume with dirty region logging (DRL) enabled, use this command:

**vxassist [-g *diskgroup*] make *volume length* layout=mirror **
logtype=drl

Note By default, the `vxassist` command creates one log plex for a mirrored volume.

For a volume that will be written to sequentially, such as a database log volume, use the following command to specify that sequential DRL is to be used:

```
# vxassist [-g diskgroup] make volume length layout=mirror \
  logtype=drlseq
```

To add DRL logging to a volume that has DCO enabled, or to change the number of DRL logs, follow the procedure that is described in "Adding DRL Logging to a Mirrored Volume" on page 211.

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 the 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.

For more information about ordered allocation, see "Specifying Ordered Allocation of Storage to Volumes" on page 179 and the `vxassist`(1M) manual page.

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. For more information on striping, see "Striping (RAID-0)" on page 17.

Note 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.

To create a striped volume, use the following command:

```
# vxassist [-b] [-g diskgroup] make volume length layout=stripe
```

Note Specify the `-b` option if you want to make the volume immediately available for use. See "Initializing and Starting a Volume" on page 194 for details.

For example, to create the 10-gigabyte striped volume `volzebra`, use the following command:

```
# vxassist -b 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, `disk03`, `disk04`, and `disk05`, use the following command:

```
# vxassist -b make stripevol 30g layout=stripe disk03 disk04 disk05
```

To change the default number of columns from 2, or the stripe width from 64 kilobytes, 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 make stripevol 30g layout=stripe stripeunit=32k \
  ncol=5
```

Creating a Mirrored-Stripe Volume

A mirrored-stripe volume mirrors several striped data plexes.

Note 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_mirrors] \
  [ncol=number_columns] [stripewidth=size]
```

Note Specify the `-b` option if you want to make the volume immediately available for use. See “Initializing and Starting a Volume” on page 194 for details.

Alternatively, first create a striped volume, and then mirror it as described in “Adding a Mirror to a Volume” on page 205. In this case, the additional data plexes may be either striped or concatenated.

Creating a Striped-Mirror Volume

A striped-mirror volume is an example of a layered volume which stripes several underlying mirror volumes.

Note 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_mirrors] \
  [ncol=number_columns] [stripewidth=size]
```

Note Specify the `-b` option if you want to make the volume immediately available for use. See “Initializing and Starting a Volume” on page 194 for details.

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, you can use either of the commands described in this section.

```
# vxassist [-b] [-g diskgroup] make volume length layout=layout \
  mirror=target [attributes]
```

Note Specify the `-b` option if you want to make the volume immediately available for use. See “Initializing and Starting a Volume” on page 194 for details.

The attribute `mirror=target` specifies that volumes should be mirrored between identical target IDs on different controllers.

```
# vxassist [-b] [-g diskgroup] make volume length layout=layout \
  mirror=ctlr [attributes]
```

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.

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:

```
# vxassist -b 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 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.

See “Specifying Ordered Allocation of Storage to Volumes” on page 179 for a description of other ways in which you can control how volumes are laid out on the specified storage.

Creating a RAID-5 Volume

Note VxVM supports this feature for private disk groups, but not for 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.

Note 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.

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. For more information on RAID-5 volumes, see "RAID-5 (Striping with Parity)" on page 24.

Caution 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_columns] [stripewidth=size] [nlog=number] \
  [loglen=log_length]
```

Note Specify the `-b` option if you want to make the volume immediately available for use. See "Initializing and Starting a Volume" on page 194 for details.

For example, to create the RAID-5 volume `volraid` together with 2 RAID-5 logs, use the following command:

```
# vxassist -b 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.

Note If you require RAID-5 logs, you must use the `logdisk` attribute to specify the disks to be used for the log plexes.

RAID-5 logs can be concatenated or striped plexes, and each RAID-5 log associated with a RAID-5 volume has a complete copy of the logging information for the volume. To support concurrent access to the RAID-5 array, the log should be several times the stripe size of the RAID-5 plex.

It is suggested that you configure a minimum of two RAID-5 log plexes for each RAID-5 volume. These log plexes should be located on different disks. Having two RAID-5 log plexes for each RAID-5 volume protects against the loss of logging information due to the failure of a single disk.

If you use ordered allocation when creating a RAID-5 volume on specified storage, you must use the `logdisk` attribute to specify on which disks the RAID-5 log plexes should be created. Use the following form of the `vxassist` command to specify the disks from which space for the logs is to be allocated:

```
# vxassist [-b] [-g diskgroup] -o ordered make volume length \
layout=raid5 [ncol=number_columns] [nlog=number] \
[loglen=log_length] logdisk=disk[,disk,...] storage_attributes
```

For example, the following command creates a 3-column RAID-5 volume with the default stripe unit size on disks `disk04`, `disk05` and `disk06`. It also creates two RAID-5 logs on disks `disk07` and `disk08`.

```
# vxassist -b make volraid 10g layout=raid5 ncol=3 nlog=2 \
  logdisk=disk07,disk08 disk04 disk05 disk06
```

Note The number of logs must equal the number of disks specified to `logdisk`.

For more information about ordered allocation, see "Specifying Ordered Allocation of Storage to Volumes" on page 179 and the `vxassist`(1M) manual page.

If you need to add more logs to a RAID-5 volume at a later date, follow the procedure described in "Adding a RAID-5 Log" on page 212.

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. Subdisks can be created using the method described in "Creating Subdisks" on page 151. 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 plex raidplex layout=raid5 stwidth=32 \
  sd=disk00-01,disk01-00,disk02-00,disk03-00
```

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 plex raidplex layout=raid5 ncolumn=4 stwidth=32
```

Note 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 154).

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 plex raidplex layout=raid5 stwidth=32 \
  sd=disk00-00:0,disk01-00:1,disk02-00:2,disk03-00:0, \
  disk04-00:1,disk05-00:2
```

This command stacks subdisks `disk00-00` and `disk03-00` consecutively in column 0, subdisks `disk01-00` and `disk04-00` consecutively in column 1, and subdisks `disk02-00` and `disk05-00` 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 plex raidlog1 disk06-00
# vxmake plex raidlog2 disk07-00
```

To create a RAID-5 volume, specify the usage type to be `RAID-5` using the following command:

```
# vxmake -Uraid5 vol raidvol
```

RAID-5 plexes and RAID-5 log plexes are associated with the volume `raidvol` using the following command:

```
# vxmake -Uraid5 vol raidvol plex=raidplex,raidlog1,raidlog2
```

Note 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. The procedure is described in “Initializing and Starting a Volume” on page 194.

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 < description_file
```

Alternatively, you can specify the file to vxmake using the -d option:

```
# vxmake -d description_file
```

The following sample description file defines a volume, db, with two plexes:

```
#rectyp #name       #options
sd      disk3-01    disk=disk3 offset=0 len=10000
sd      disk3-02    disk=disk3 offset=25000 len=10480
sd      disk4-01    disk=disk4 offset=0 len=8000
sd      disk4-02    disk=disk4 offset=15000 len=8000
sd      disk4-03    disk=disk4 offset=30000 len=4480
plex    db-01       layout=STRIPE ncolumn=2 stwidth=16k
                    sd=disk3-01:0/0,disk3-02:0/10000,disk4-01:1/0,
disk4-02:1/8000,disk4-03:1/16000
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 first plex, db-01, is striped and has five subdisks on two physical disks, disk3 and disk4. 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. The procedure is described in the following section, "Initializing and Starting a Volume."

Initializing and Starting a Volume

A volume must be initialized if it was created by the `vxmake` command and has not yet been initialized, or if the volume has been set to an uninitialized state.

Note If you create a volume using the `vxassist` command, `vxassist` initializes and starts the volume automatically unless you specify the attribute `init=none`.

To initialize and start a volume, use the following command:

```
# vxvol start volume
```

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 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 make volume length layout=mirror init=active
```

Caution 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 unitialized following a system crash. If in doubt, use the `-b` option to `vxassist` instead.

The following command can be used to enable a volume without initializing it:

```
# vxvol 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 init active volume
```

If you want to zero out the contents of an entire volume, use this command to initialize it:

```
# vxvol init zero volume
```

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 make volume length layout=raid5 init=zero
```

Note You cannot use the `-b` option to make this operation a background task.

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 the disk group `rootdg` sets up block and character (raw) device files that can be used to access the volume:

`/dev/vx/dsk/`*volume*—block device file for *volume*

`/dev/vx/rdsk/`*volume*—character device file for *volume*

For volumes in disk groups other than `rootdg`, the pathnames include a directory named for the disk group:

`/dev/vx/dsk/`*diskgroup/volume*—block device file for *volume*

`/dev/vx/rdsk/`*diskgroup/volume*—character device file for *volume*

Use the appropriate device node to create, mount and repair file systems, and to lay out databases that require raw partitions.

Administering Volumes 8

Introduction

This chapter describes how to perform common maintenance tasks on volumes in VERITAS Volume Manager (VxVM). This includes displaying volume information, monitoring tasks, adding and removing logs, resizing volumes, removing mirrors, removing volumes, backing up volumes using mirrors and snapshots, and changing the layout of volumes without taking them offline.

Note Most VxVM commands require 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 -ht
```

The following is example output from the `vxprint` command:

```
Disk group: rootdg

DG NAME      NCONFIG   NLOG    MINORS    GROUP-ID
DM NAME      DEVICE    TYPE    PRIVLEN   PUBLEN  STATE
V  NAME      USETYPE   KSTATE  STATE     LENGTH  READPOL PREFPLEX
PL NAME      VOLUME    KSTATE  STATE    LENGTH   LAYOUT  NCOL/WID  MODE
SD NAME      PLEX      DISK    DISKOFFS  LENGTH  [COL/]OFF  DEVICE MODE
dm disk11    c1t0d0s2  sliced  559       1044400 -
dm disk12    c1t1d0s2  sliced  559       1044400 -

v  pubs      fsgen     ENABLED ACTIVE    2288    SELECT  -
pl pubs-01   pubs      ENABLED ACTIVE    2288    CONCAT  -         RW
sd disk11-01 pubs-01   disk11  0         2288    0          c1t0d0  ENA
```


```
v  voldef     sgen    ENABLED ACTIVE    20480     SELECT  -
pl voldef-01  voldef  ENABLED ACTIVE    20480     CONCAT  -       RW
sd disk12-02  voldef-0disk12  2288      20480     0         c1t1d0  ENA
```

where `dg` is a disk group, `dm` is a disk, `v` is a volume, `pl` is a plex, and `sd` is a subdisk. The top 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.

To display volume-related information for a specific volume, use the following command:

```
# vxprint -t volume
```

For example, to display information about the `voldef` volume, use the following command:

```
# vxprint -t voldef
```

This is example output from this command:

```
Disk group: rootdg

V  NAME    USETYPE  KSTATE   STATE    LENGTH  READPOL  PREFPLEX

v  voldef  fsgen    ENABLED  ACTIVE   20480   SELECT   -
```

Note If you enable enclosure-based naming, and use the `vxprint` command to display the structure of a volume, it shows enclosure-based disk device names (disk access names) rather than c#t#d#s# names. See "Using vxprint with Enclosure-Based Disk Names" on page 62 for information on how to obtain the true device names.

The following section describes the meaning of the various volume states that may be displayed.

Volume States

The following volume states may be displayed by VxVM commands such as `vxprint`:

ACTIVE Volume State

The volume has been started (kernel state is currently ENABLED) or was in use (kernel state was ENABLED) when the machine was rebooted. If the volume is currently ENABLED, the state of its plexes at any moment is not certain (since the volume is in use).

If the volume is currently DISABLED, this means that 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 currently DISABLED, parity cannot be guaranteed to be synchronized.

CLEAN Volume State

The volume is not started (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 Volume State

The volume contents are not initialized. The kernel state is always DISABLED when the volume is EMPTY.

NEEDSYNC Volume State

The volume requires a resynchronization operation the next time it is started. For a RAID-5 volume, a parity resynchronization operation is required.

REPLAY Volume State

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.

SYNC Volume State

The volume is either in read-writeback recovery mode (kernel state is currently ENABLED) or was in read-writeback mode when the machine was rebooted (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, this means that the plexes are being resynchronized through the read-writeback recovery. If the volume is DISABLED, it means that the plexes were being resynchronized through read-writeback when the machine rebooted and therefore still need to be synchronized.

For a RAID-5 volume, the volume is either undergoing a parity resynchronization (kernel state is currently ENABLED) or was having its parity resynchronized when the machine was rebooted (kernel state is DISABLED).

Note The interpretation of these flags 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. The clean flag is always set in any case where the volume is marked CLEAN.

Volume Kernel States

The *volume kernel state* indicates the accessibility of the volume. The volume kernel state allows a volume to have an offline (DISABLED), maintenance (DETACHED), or online (ENABLED) mode of operation.

Note No user intervention is required to set these states; they are maintained internally. On a system that is operating properly, all volumes are enabled.

The following volume kernel states are defined:

DETACHED Volume Kernel State

Maintenance is being performed on the volume. The volume cannot be read or written, but plex device operations and `ioctl` function calls are accepted.

DISABLED Volume Kernel State

The volume is offline and cannot be accessed.

ENABLED Volume Kernel State

The volume is online and can be read from or written to.

Monitoring and Controlling Tasks

Note VxVM supports this feature for private disk groups, but not for shareable disk groups in a cluster environment.

The VxVM task monitor tracks the progress of system recovery by monitoring task creation, maintenance, and completion. The task monitor allows you to monitor task progress and to modify characteristics of tasks, such as pausing and recovery rate (for example, to reduce the impact on system performance).

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 allow you to specify a tag using the `-t` option:

`vxassist`, `vxevac`, `vxplex`, `vxreattach`, `vxrecover`, `vxresize`, `vxsd`, and `vxvol`

For example, to execute a `vxrecover` command and track all the resulting tasks as a group with the task tag `myrecovery`, use the following command:

```
# vxrecover -t myrecovery -b disk05
```

Any tasks started by the utilities invoked by `vxrecover` also inherit its task ID and task tag, so 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

Note 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.

You can use the `vxtask` command to administer operations on VxVM tasks that are running on the system. Operations include listing tasks, modifying the state of a task (pausing, resuming, aborting) and modifying the rate of progress of a task. For detailed information about how to use `vxtask`, refer to the `vxtask`(1M) manual page.

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. The administrator can change the state of a task, giving coarse-grained control over the progress of the operation. For those operations that support it, the rate of progress of the task can be changed, giving more fine-grained control over the task.

vxtask Operations

The `vxtask` command supports the following operations:

`abort` Causes the specified task to cease operation. 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` Lists tasks running on the system in one-line summaries. 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 a `taskid` argument is supplied, the output is limited to those tasks whose `taskid` or task tag match `taskid`. The remaining arguments are used to filter tasks and limit the tasks actually listed.

`monitor` Prints information continuously about a task or group of tasks as task information changes. This allows you to track the progression of tasks. Specifying `-l` causes a long listing to be printed. By default, short 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` Puts a running task in the paused state, causing it to suspend operation.

`resume` Causes a paused task to continue operation.

`set` Changes modifiable parameters of a task. 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 `slow` is specified, this introduces a delay between such operations with a default value for *iodelay* of 250 milliseconds. The larger the value of *iodelay* that is specified, the slower is the progress of the task and the fewer system resources that it consumes in a given time. (The `slow` attribute is also accepted by the `vxplex`, `vxvol` and `vxrecover` commands.)

vxtask Usage

To list all tasks currently running on the system, use the following command:

```
# vxtask list
```

To print tasks hierarchically, with child tasks following the parent tasks, us the `-h` option, as follows:

```
# vxtask -h list
```

To trace all tasks in the disk group `foodg` that are currently paused, as well as any tasks with the tag `sysstart`, use the following command:

```
# vxtask -G foodg -p -i sysstart list
```

Use the `vxtask -p list` command lists all paused tasks, and use `vxtask resume` to continue execution (the task may be specified by its ID or by its tag):

```
# 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 attempt to reverse the progress of the operation so far. For an example of how to use `vxtask` to monitor and modify the progress of the Online Relayout feature, see “Controlling the Progress of a Relayout” on page 237.

Stopping a Volume

Stopping a volume renders it unavailable to the user, and changes the volume 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 stop volume ...
```

For example, to stop a volume named `vol01`, use the following command:

```
# vxvol stop vol01
```

To stop all ENABLED volumes, use the following command:

```
# vxvol stopall
```

To stop all ENABLED volumes in a specified disk group, use the following command:

```
# vxvol -g diskgroup stopall
```


Putting a Volume in Maintenance Mode

If all mirrors of a volume become STALE, you can place the volume in maintenance mode. 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 maint volume
```

To assist in choosing the revival source plex, use `vxprint` to list the stopped volume and its plexes.

To take a plex (in this example, `vol01-02`) offline, use the following command:

```
# vxmend off vol01-02
```

The `vxmend on` command can change the state of an OFFLINE plex of a DISABLED volume to STALE. For example, to put a plex named `vol01-02` in the STALE state, use the following command:

```
# vxmend on vol01-02
```

Running the `vxvol start` command on the volume then revives the plex as described in the next section.

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:

```
# vxvol -g diskgroup startall
```

Alternatively, to start a DISABLED volume, use the following command:

```
# vxrecover -g diskgroup -s volume ...
```

To start all DISABLED volumes, enter:

```
# vxrecover -s
```

To prevent any recovery operations from being performed on the volumes, additionally specify the `-n` option to `vxrecover`.

Adding a Mirror to a Volume

A mirror can be added to an existing volume with the `vxassist` command, as follows:

```
# vxassist [-b] [-g diskgroup] mirror volume
```

Note If specified, the `-b` option makes synchronizing the new mirror a background task.

For example, to create a mirror of the volume `voltest`, use the following command:

```
# vxassist -b mirror voltest
```

Another way to mirror an existing volume is by first creating a plex, and then attaching it to a volume, using the following commands:

```
# vxmake plex plex sd=subdisk ...
# vxplex 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:

```
# /etc/vx/bin/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`, use the following command:

```
# vxassist make nomirror 20g nmirror=1
```

Mirroring Volumes on a VM Disk

Mirroring volumes on a VM disk gives you one or more copies of your volumes in another disk location. By creating mirror copies of your volumes, you protect your system against loss of data in case of a disk failure. You can use this task on your root disk to make a second copy of the boot information available on an alternate disk. This allows you to boot your system even if your root disk is corrupted.

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, make sure that the target disk has an equal or greater amount of space as the originating disk and then do the following:

1. Select menu item `6 (Mirror volumes on a disk)` from the `vxdiskadm` main menu.

2. At the following prompt, enter the disk name of the disk that you wish to mirror:

```
Mirror volumes on a disk
Menu: VolumeManager/Disk/Mirror

This operation can be used to mirror volumes on a disk. These
volumes can be mirrored onto another disk or onto any
available disk space. Volumes will not be mirrored if they are
already mirrored. Also, volumes that are comprised of more than
one subdisk will not be mirrored.

Enter disk name [<disk>,list,q,?] disk02
```

3. At the following prompt, enter the target disk name (this disk must be the same size or larger than the originating disk):

```
You can choose to mirror volumes from disk disk02 onto any
available disk space, or you can choose to mirror onto a
specific disk. To mirror to a specific disk, select the name of
that disk. To mirror to any available disk space, select "any".
Enter destination disk [<disk>,list,q,?] (default: any) disk01
```

4. At the following prompt, press Return to make the mirror:

```
The requested operation is to mirror all volumes on disk disk02
in disk group rootdg onto available disk space on disk disk01.

NOTE: This operation can take a long time to complete.

Continue with operation? [y,n,q,?] (default: y)
The vxdiskadm program displays the status of the mirroring
operation, as follows:
Mirror volume voltest-bk00 ...

Mirroring of disk disk01 is complete.
```

5. At the following 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 a mirror is no longer needed, you can remove it to free up disk space.

Note The last valid plex associated with a volume cannot be removed.

To remove a mirror from a volume, use the following command:

```
# vxassist remove mirror volume
```

Additionally, you can use storage attributes to specify the storage to be removed. For example, to remove a mirror on disk `disk01`, from volume vol01, enter:

```
# vxassist remove mirror vol01 !disk01
```

For more information about storage attributes, see “Creating a Volume on Specific Disks” on page 178.

Alternatively, use the following command to dissociate and remove a mirror from a volume:

```
# vxplex -o rm dis plex
```

For example, to dissociate and remove a mirror named `vol01-02`, use the following command:

```
# vxplex -o rm dis vol01-02
```

This command removes the mirror `vol01-02` and all associated subdisks. This is equivalent to entering the following separate commands:

```
# vxplex dis vol01-02
# vxedit -r rm vol01-02
```

Adding a DCO and DCO Volume

Caution If the existing volume was created before release 3.2 of VxVM, and it has any attached snapshot plexes or it is associated with any snapshot volumes, follow the procedure given in “Enabling Persistent FastResync on Existing Volumes with Associated Snapshots” on page 223. The procedure given in this section is for existing volumes without existing snapshot plexes or associated snapshot volumes.

To put Persistent FastResync into effect for a volume, a Data Change Object (DCO) and DCO volume must first be associated with that volume. When you have added a DCO object and DCO volume to a volume, you can then enable Persistent FastResync on the volume as described in “Enabling FastResync on a Volume” on page 221.

Note You may need an additional license to use the Persistent FastResync feature. Even if you do not have a license, you can configure a DCO object and DCO volume so that snap objects are associated with the original and snapshot volumes. For more information about snap objects, see "How Persistent FastResync Works with Snapshots" on page 43.

To add a DCO object and DCO volume to an existing volume (which may already have dirty region logging (DRL) enabled), use the following procedure:

1. Ensure that the disk group containing the existing volume 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
   ```

   For more information, see "Upgrading a Disk Group" on page 145.

2. Use the following command to turn off Non-Persistent FastResync on the original volume if it is currently enabled:

   ```
   # vxvol [-g diskgroup] set fastresync=off volume
   ```

   If you are uncertain about which volumes have Non-Persistent FastResync enabled, use the following command to obtain a listing of such volumes:

   ```
   # vxprint [-g diskgroup] -F "%name" \
     -e "v_fastresync=on && !v_hasdcolog"
   ```

3. Use the following command to add a DCO and DCO volume to the existing volume:

   ```
   # vxassist [-g diskgroup[ addlog volume logtype=dco \
     [ndcomirror=number] [dcolen=size] [storage_attributes]
   ```

   For non-layered volumes, the default number of plexes in the mirrored DCO volume is equal to the lesser of the number of plexes in the data volume or 2. For layered volumes, the default number of DCO plexes is always 2. If required, use the `ndcomirror` attribute to specify a different *number*. It is recommended that you configure as many DCO plexes as there are existing data and snapshot plexes in the volume. For example, specify `ndcomirror=3` when adding a DCO to a 3-way mirrored volume.

   The default size of each plex is 132 blocks. You can use the `dcolen` attribute to specify a different *size*. If specified, the size of the plex must be a integer multiple of 33 blocks from 33 up to a maximum of 2112 blocks.

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` output for the volume named `zoo` (the TUTIL0 and PUTIL0 columns are omitted for clarity):

```
TY NAME         ASSOC       KSTATE    LENGTH    PLOFFS  STATE   ...
v  zoo          fsgen       ENABLED   1024      -       ACTIVE
pl zoo-01       zoo         ENABLED   1024      -       ACTIVE
sd c1t66d0s2-02 zoo-01      ENABLED   1024      0       -
pl foo-02       zoo         ENABLED   1024      -       ACTIVE
sd c1t67d0s2-02 zoo-02      ENABLED   1024      0       -
dc zoo_dco      zoo         -         -         -       -
v  zoo_dcl      gen         ENABLED   132       -       ACTIVE
pl zoo_dcl-01   zoo_dcl     ENABLED   132       -       ACTIVE
sd c1t66d0s2-01 zoo_dcl-01  ENABLED   132       0       -
pl zoo_dcl-02   zoo_dcl     ENABLED   132       -       ACTIVE
sd c1t67d0s2-01 zoo_dcl-02  ENABLED   132       0       -
```

In this output, the DCO object is shown as `zoo_dco`, and the DCO volume as `zoo_dcl` with 2 plexes, `zoo_dcl-01` and `zoo_dcl-02`.

For more information, see the `vxassist`(1M) manual page.

Attaching a DCO and DCO volume to a RAID-5 Volume

The procedure in the previous section can be used to add a DCO and DCO volume to a RAID-5 volume. This allows you to enable Persistent FastResync on the volume for fast resynchronization of snapshots on snapback (see “Enabling FastResync on a Volume” on page 221). However, the procedure has the side effect of converting the RAID-5 volume into a special type of layered volume. You cannot relayout or resize such a volume unless you convert it back to a pure RAID-5 volume. To do this, remove any snapshot plexes from the volume, and dissociate the DCO and DCO volume from the layered volume using the procedure described in “Removing a DCO and DCO Volume” on page 210. You can then perform relayout and resize operations on the resulting non-layered RAID-5 volume.

To allow Persistent FastResync to be used with the RAID-5 volume again, re-associate the DCO and DCO volume as described in “Reattaching a DCO and DCO Volume” on page 211.

Note Dissociating a DCO and DCO volume disables Persistent FastResync on the volume. A full resynchronization of any remaining snapshots is required when they are snapped back.

Specifying Storage for DCO Plexes

If the disks that contain volumes and their snapshots are to be moved or split into different disk groups, the disks that contain their respective DCO plexes must be able to accompany them. By default, VxVM attempts to place the DCO plexes on the same disks as the data plexes of the parent volume. However, this may be impossible if there is insufficient space available on those disks. In this case, VxVM uses any available space on other disks in the disk group. If the DCO plexes are placed on disks which are used to hold the plexes of other volumes, this may cause problems when you subsequently attempt to move volumes into other disk groups.

You can use storage attributes to specify explicitly which disks to use for the DCO plexes. If possible, specify the same disks as those on which the volume is configured. For example, to add a DCO object and DCO volume with plexes on `disk5` and `disk6`, and a plex size of 264 blocks to the volume, `myvol`, use the following command:

```
# vxassist -g mydg addlog myvol logtype=dco dcolen=264 \
  disk5 disk6
```

If required, you can use the `vxassist move` command to relocate DCO plexes to different disks. For example, the following command moves the plexes of the DCO volume for volume `vol1` from `disk3` and `disk4` to `disk7` and `disk8`:

```
# vxassist -g mydg move vol1_dcl !disk3 !disk4 disk7 disk8
```

For more information, see “Considerations for Placing DCO Plexes” on page 138.

Removing a DCO and DCO Volume

To dissociate a DCO object, DCO volume and any snap objects from a volume, use the following command:

```
# vxassist [-g diskgroup] remove log volume logtype=dco
```

This completely removes the DCO object, DCO volume and any snap objects. It also has the effect of disabling FastResync for the volume.

Alternatively, you can use the `vxdco` command to the same effect:

```
# vxdco [-g diskgroup] [-o rm] dis dco_obj
```

The default name of the DCO object, *dco_obj*, for a volume is usually formed by appending the string `_dco` to the name of the parent volume. To find out the name of the associated DCO object, use the `vxprint` command on the volume.

To dissociate, but not remove, the DCO object, DCO volume and any snap objects from the volume, *myvol*, in the disk group, *mydg*, use the following command:

```
# vxdco -g mydg dis myvol_dco
```

This form of the command dissociates the DCO object from the volume but does not destroy it or the DCO volume. If the `-o rm` option is specified, the DCO object, DCO volume and its plexes, and any snap objects are also removed.

Note Dissociating a DCO and DCO volume disables Persistent FastResync on the volume. A full resynchronization of any remaining snapshots is required when they are snapped back.

For more information, see the `vxassist`(1M) and `vxdco`(1M) manual pages.

Reattaching a DCO and DCO Volume

If the DCO object and DCO volume are not removed by specifying the `-o rm` option to `vxdco`, they can be reattached to the parent volume using the following command:

```
# vxdco [-g diskgroup] att volume dco_obj
```

For example, to reattach the DCO object, *myvol_dco*, to the volume, *myvol*, use the following command:

```
# vxdco -g mydg att myvol myvol_dco
```

For more information, see the `vxdco`(1M) manual page.

Adding DRL Logging to a Mirrored Volume

To put dirty region logging (DRL) into effect for a mirrored volume, a log subdisk must be added 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] addlog volume logtype=drl [nlog=n]
```

Note If specified, the `-b` option makes adding the new logs a background task.

The `nlog` attribute can be used to specify the number of log plexes to add. By default, one log plex is added. For example, to add a single log plex for the volume `vol03`, use the following command:

```
# vxassist addlog vol03 logtype=drl
```

When the `vxassist` command is used to add a log subdisk to a volume, by default a log plex is also created to contain the log subdisk unless you include the keyword `nolog` in the layout specification.

For a volume that will be written to sequentially, such as a database log volume, include the **logtype=drlseq** attribute to specify that sequential DRL is to be used:

```
# vxassist addlog volume logtype=drlseq [nlog=n]
```

Once created, the plex containing a log subdisk can be treated as a regular plex. Data subdisks can be added to the log plex. The log plex and log subdisk can be removed using the same procedures as are used to remove ordinary plexes and subdisks.

Removing a DRL Log

To remove a DRL log, use the `vxassist` command as follows:

```
# vxassist remove log volume [nlog=n]
```

Use the optional attribute `nlog=`*n* to specify the number, *n*, of logs to be removed. By default, the `vxassist` command removes one log.

Adding a RAID-5 Log

Note You may need an additional license to use this feature.

Only one RAID-5 plex can exist per RAID-5 volume. Any additional plexes become RAID-5 log plexes, which are used to log information about data and parity being written to the volume. When a RAID-5 volume is created 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] addlog volume [loglen=length]
```

Note If specified, the `-b` option makes adding the new log a background task.

Note You can specify the log length used when adding the first log to a volume. 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`, use the following command:

```
# vxassist addlog volraid
```

Adding a RAID-5 Log using vxplex

As an alternative to using `vxassist`, you can add a RAID-5 log using the `vxplex` command. For example, to attach a RAID-5 log plex, `r5log`, to a RAID-5 volume, `r5vol`, use the following command:

```
# vxplex att r5vol r5log
```

The attach operation can only proceed if the size of the new log is large enough to hold all of 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. This is because 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 -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 -o rm dis plex
```

For example, to dissociate and remove the log plex `volraid-02` from `volraid`, use the following command:

```
# vxplex -o rm dis volraid-02
```

You can also remove a RAID-5 log with the `vxassist` command, as follows:

```
# vxassist remove log volume [nlog=n]
```

Use the optional attribute `nlog=`*n* to specify the number, *n*, of logs to be removed. By default, the `vxassist` command removes one log.

Note When removing the log leaves the volume with less than two valid logs, a warning is printed and the operation is not allowed to continue. The operation may be forced by additionally specifying the `-f` option to `vxplex` or vxassist.

Resizing a Volume

Resizing a volume changes the volume size. For example, you might need to increase the length of a volume if it is no longer large enough for the amount of data to be stored on it. To resize a volume, use one of the commands: `vxresize` (preferred), `vxassist`, or `vxvol`. Alternatively, you can use the graphical VERITAS Enterprise Administrator (VEA) to resize volumes.

Note You cannot use the procedures in this chapter 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 really need to resize the volumes on the `root` disk, see the section "Recovering a root Disk" in the chapter "Recovery from Boot Disk Failure" in the *VERITAS Volume Manager Troubleshooting Guide*.

If a volume is increased in size, the `vxassist` command automatically locates available disk space. The `vxresize` command requires that you specify the names of the disks to be used 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 automatically free unused space for use by the disk group. For the `vxvol` command, you must do this yourself. To find out by how much you can grow a volume, use the following command:

```
# vxassist 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 no unit is specified, sectors are assumed. The `vxassist` command also allows you to specify an increment by which to change the volume's size.

Caution If you use `vxassist` or `vxvol` to resize a volume, do not shrink it below the size of the file system which is located 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. Other file systems may require you to back up your data so that you can later recreate the file system and restore its data.

Resizing Volumes using vxresize

Use the `vxresize` command to resize a volume containing a file system. Although other commands can be used to resize volumes containing file systems, the `vxresize` command offers the advantage of automatically resizing certain types of file system as well as the volume.

See the following table for details of what operations are permitted and whether you must first unmount the file system to resize the file system:

Permitted Resizing Operations on File Systems

	VxFS	UFS
Mounted File System	Grow and shrink	Grow only
Unmounted File System	Not allowed	Grow only

For example, the following command resizes the 1-gigabyte volume, `homevol`, that contains a VxFS file system to 10 gigabytes using the spare disks `disk10` and `disk11`:

```
# vxresize -b -F vxfs -t homevolresize homevol 10g disk10 disk11
```

The `-b` option specifies that this operation runs in the background. Its progress can be monitored by specifying the task tag `homevolresize` to the `vxtask` command.

Note the following restrictions for using `vxresize`:

- vxresize works with VxFS and UFS file systems only.
- In some situations, when resizing large volumes, `vxresize` may take a long time to complete.
- Resizing a volume with a usage type other than FSGEN or RAID5 can result in loss of data. If such an operation is required, use the `-f` option to forcibly resize such a 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: Volume` *volume* `has different organization in each mirror`

For more information about the `vxresize` command, see the `vxresize`(1M) manual page.

Resizing Volumes using vxassist

The following modifiers are used with the `vxassist` command to resize a volume:

- `growto`—increase volume to a specified length
- `growby`—increase volume by a specified amount
- `shrinkto`—reduce volume to a specified length
- `shrinkby`—reduce volume by a specified amount

Caution You cannot grow or shrink any volume associated with an encapsulated root disk (`rootvol`, `usr`, `var`, `opt`, `swapvol`, and so on) because these map to a physical underlying partition on the disk and must be contiguous. If you attempt to grow `rootvol`, `usrvol`, `varvol`, or `swapvol`, the system could become unbootable if you need to revert back to booting from slices. It can also prevent a successful Solaris upgrade and you might have to do a fresh install. Additionally, the `upgrade_start` script might fail.

Extending to a Given Length

To extend a volume *to* a specific length, use the following command:

```
# vxassist [-b] growto volume length
```

Note If specified, the `-b` option makes growing the volume a background task.

For example, to extend `volcat` to 2000 sectors, use the following command:

```
# vxassist growto volcat 2000
```

Note If you previously performed a relayout on the volume, additionally specify the attribute `layout=nodiskalign` to the `growto` command if you want the subdisks to be grown using contiguous disk space.

Extending by a Given Length

To extend a volume *by* a specific length, use the following command:

```
# vxassist [-b] growby volume length
```

Note If specified, the `-b` option makes growing the volume a background task.

For example, to extend `volcat` by 100 sectors, use the following command:

```
# vxassist growby volcat 100
```

Note If you previously performed a relayout on the volume, additionally specify the attribute `layout=nodiskalign` to the `growby` command if you want the subdisks to be grown using contiguous disk space.

Shrinking to a Given Length

To shrink a volume *to* a specific length, use the following command:

```
# vxassist shrinkto volume length
```

For example, to shrink `volcat` to 1300 sectors, use the following command:

```
# vxassist shrinkto volcat 1300
```

Caution Do not shrink the volume below the current size of the file system or database using the volume. The `vxassist shrinkto` command can be safely used on empty volumes.

Shrinking by a Given Length

To shrink a volume *by* a specific length, use the following command:

```
# vxassist shrinkby volume length
```

For example, to shrink `volcat` by 300 sectors, use the following command:

```
# vxassist shrinkby volcat 300
```

Caution Do not shrink the volume below the current size of the file system or database using the volume. The `vxassist shrinkby` command can be safely used on empty volumes.

Resizing Volumes using vxvol

To change the length of a volume using the `vxvol set` command, use the following command:

```
# vxvol set len=length volume
```

For example, to change the length to 100000 sectors, use the following command:

```
# vxvol set len=100000 vol01
```


Note The `vxvol set len` command cannot increase the size of a volume unless the needed space is available in the plexes of the volume. When the size of a volume is reduced 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 its length is being reduced, the operation must be forced using the `-o force` option to `vxvol`. This prevents accidental removal of space from applications using the volume.

The length of logs can also be changed using the following command:

```
# vxvol set loglen=length log_volume
```

Note Sparse log plexes are not valid. They must map the entire length of the log. If increasing the log length would make 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), the log length cannot be changed. This avoids the loss of any of the log contents (if the log length is decreased), or the introduction of random data into the logs (if the log length is being increased).

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 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.

The read policy can be changed from `round` to `prefer` (or the reverse), or to a different preferred plex. The `vxvol rdpol` command sets the read policy for a volume.

Note You cannot set the read policy on a RAID-5 volume. RAID-5 plexes have their own read policy (RAID).

To set the read policy to `round`, use the following command:

```
# vxvol rdpol round volume
```

For example, to set the read policy for volume `vol01` to round-robin, use the following command:

```
# vxvol rdpol round vol01
```

To set the read policy to `prefer`, use the following command:

```
# vxvol 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 rdpol prefer vol01 vol01-02
```

To set the read policy to `select`, use the following command:

```
# vxvol rdpol select volume
```

For more information about how read policies affect performance, see "Volume Read Policies" on page 298.

Removing a Volume

Once a volume is no longer necessary (it is inactive and its contents have been archived, for example), it is possible to remove the volume and free up the disk space for other uses.

Before removing a volume, use the following procedure to stop all activity on the 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 this command:

   ```
   # umount /dev/vx/dsk/diskgroup/volume
   ```

3. If the volume is listed in the `/etc/vfstab` file, remove its entry by editing this file. Refer to your operating system documentation for more information about the format of this file and how you can modify it.

4. Stop all activity by VxVM on the volume with the command:

   ```
   # vxvol stop volume
   ```

After following these steps, remove the volume with the `vxassist` command:

```
# vxassist remove volume volume
```

Alternatively, you can use the `vxedit` command to remove a volume:

```
# vxedit [-r] [-f] rm volume
```


The `-r` option to `vxedit` indicates recursive removal. This removes all plexes associated with the volume and all subdisks associated with those plexes. The `-f` option to `vxedit` forces removal. This is necessary if the volume is still enabled.

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. To do this, ensure that the target disks have sufficient space, and then use the following procedure:

1. Select menu item `7 (Move volumes from a disk)` from the `vxdiskadm` main menu.

2. At the following prompt, enter the disk name of the disk whose volumes you wish to move, as follows:

```
Move volumes from a disk
Menu: VolumeManager/Disk/Evacuate
Use this menu operation to move any volumes that are using a
disk onto other disks. Use this menu immediately prior to
removing a disk, either permanently or for replacement. You can
specify a list of disks to move volumes onto, or you can move
the volumes to any available disk space in the same disk group.

NOTE: Simply moving volumes off of a disk, without also removing
    the disk, does not prevent volumes from being moved onto
    the disk by future operations. For example, using two
    consecutive move operations may move volumes from the
    second disk to the first.

Enter disk name [<disk>,list,q,?] disk01
```

```
After the following display, you can optionally specify a list of
disks to which the volume(s) should be moved.
  You can now specify a list of disks to move onto.  Specify a
  list of disk media names (e.g., disk01) all on one line
  separated by blanks.  If you do not enter any disk media names,
  then the volumes will be moved to any available space in the
  disk group.

At the following prompt, press Return to move the volumes:
  Requested operation is to move all volumes from disk disk01 in
  group rootdg.

  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:

```
Move volume voltest ...
Move volume voltest-bk00 ...
```

When the volumes have all been moved, the `vxdiskadm` program displays the following success message:

```
Evacuation of disk disk01 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

Note You may need an additional license to use this feature.

FastResync performs quick and efficient resynchronization of stale mirrors. It also increases the efficiency of the VxVM snapshot mechanism when used with operations such as backup and decision support. See "Backing Up Volumes Online Using Snapshots" on page 228 and "FastResync" on page 41 for more information.

From Release 3.2, there are two possible versions of FastResync that can be enabled on a volume:

- Persistent FastResync, introduced in VxVM 3.2, holds copies of the FastResync maps on disk. These can be used for the speedy recovery of mirrored volumes if a system is rebooted. This form of FastResync requires that both a data change object (DCO) and DCO volume first be associated with the volume.

  See "Creating a Volume with a DCO and DCO Volume" on page 184, and "Adding a DCO and DCO Volume" on page 207 for more information.

  If the existing volume was created before release 3.2 of VxVM, and it has any attached snapshot plexes or it is associated with any snapshot volumes, follow the procedure given in "Enabling Persistent FastResync on Existing Volumes with Associated Snapshots" on page 223.

- Non-Persistent FastResync, introduced in VxVM 3.1, holds the FastResync maps in memory. These do not survive on a system that is rebooted.

By default, FastResync is not enabled on newly created volumes. Specify the `fastresync=on` attribute to the `vxassist make` command if you want to enable FastResync on a volume that you are creating.

Note It is not possible to configure both Persistent and Non-Persistent FastResync on a volume. Persistent FastResync is used if a DCO object and a DCO volume are associated with the volume. Otherwise, Non-Persistent FastResync is used.

To turn FastResync on for an existing volume, specify `fastresync=on` to the `vxvol` command as shown here:

```
# vxvol [-g diskgroup] set fastresync=on volume
```

Note To use FastResync with a snapshot, FastResync must be enabled before the snapshot is taken, and 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
```

This command returns `on` if FastResync is enabled; 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
```

This command returns `on` if Persistent FastResync is enabled; otherwise, it returns `off`.

To list all volumes on which Non-Persistent FastResync is enabled, use the following command:

```
# 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 shown here:

```
# vxvol [-g diskgroup] set fastresync=off volume
```

Turning FastResync off releases all tracking maps for the specified volume. All subsequent reattaches will not use the FastResync facility, but perform a full resynchronization of the volume. This occurs even if FastResync is later turned on.

Enabling Persistent FastResync on Existing Volumes with Associated Snapshots

The procedure described in this section describes how to enable Persistent FastResync on a volume created before release 3.2 of VxVM, and which has attached snapshot plexes or is associated with one or more snapshot volumes.

Note If you do not perform the reconfiguration described in this section, full resynchronization is required every time that snapback is used to reattach a snapshot to its original volume.

If a volume was created before release 3.2 of VxVM, but does not have any snapshot plexes or associated snapshot volumes, you do not need to perform this procedure if you perform a snapstart operation on the volume after you have added a data change object (DCO) and DCO volume to it.

Before enabling Persistent FastResync on an existing volume that contains any snapshot plexes, or which has any snapshot volumes, you must create and associate a data change object (DCO) and DCO volume with the volume. The number of plexes that you need to configure in a DCO volume is determined by the number of data and snapshot plexes that must be tracked. (A volume's snapshot plexes are those that the vxprint command displays with their state set to SNAPDONE.)

Because Persistent FastResync performs tracking on the original volume and on its snapshot volumes, you must also configure and associate a DCO and DCO volume with each snapshot volume. It is only necessary to do this before you have enabled Persistent FastResync on a volume. Once you have enabled Persistent FastResync on a volume, the snapstart, snapshot and snapback operations handle the creation and management of DCOs and DCO volumes automatically.

Note The DCO plexes require persistent storage space on disk to be available for the FastResync maps. To make room for the DCO plexes, you may need to add extra disks to the disk group, or reconfigure existing volumes to free up space in the disk group. Another way to add disk space is to use the disk group move feature to bring in spare disks from a different disk group. For more information, see "Reorganizing the Contents of Disk Groups" on page 133.

Perform the following steps to enable Persistent FastResync on an existing volume that has attached snapshot plexes or associated snapshot volumes:

1. Upgrade the disk group containing the existing volume to at least version 90 before performing the remainder of the procedure described in this section. 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***

   For more information, see "Upgrading a Disk Group" on page 145.

2. For a volume that has one or more associated snapshot volumes, it is recommended that you use following command to reattach and resynchronize each snapshot:

   # **vxassist [-g *diskgroup*] snapback *snapvol***

   If Non-Persistent 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 Non-Persistent FastResync was not enabled, a full resynchronization is performed.

   If you choose to reattach all the snapshots, you need only add a DCO and DCO volume to the original volume.

   If you choose not to snapback the snapshot volumes, you must add a DCO and DCO volume to the original volume, and separately to each of its snapshot volumes. This approach requires a full resynchronization on the first subsequent snapback of each snapshot volume after you have enabled Persistent FastResync.

3. Use the following command to turn off Non-Persistent FastResync on the original volume if it is currently enabled:

   # **vxvol [-g *diskgroup*] set fastresync=off *volume***

   If you are uncertain about which volumes have Non-Persistent FastResync enabled, use the following command to obtain a listing of such volumes:

   ```
   # vxprint [-g diskgroup] -F "%name" \
      -e "v_fastresync=on && !v_hasdcolog"
   ```

4. Use the following command on the original volume and on each of its snapshot volumes (if any) to add a DCO and DCO volume.

```
# vxassist [-g diskgroup] addlog volume logtype=dco \
  dcolen=loglen ndcomirror=number [storage_attribute ...]
```

Set the value of `ndcomirror` equal to the number of data and snapshot plexes in the volume.

The `ndcomirror` attribute specifies the number of DCO plexes that are created in the DCO volume. It is recommended that you configure as many DCO plexes as there are data plexes in the volume. For example, specify `ndcomirror=3` when adding a DCO to a 3-way mirrored volume. If a volume has any snapshot plexes, a separate DCO plex must also be reserved for each of these plexes. These 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 `ndcomirror=5` for a volume with 3 data plexes and 2 snapshot plexes. For a snapshot volume, set the value of `ndcomirror` to the number of plexes in the volume.

The value of the `dcolen` attribute specifies the size of a DCO plex, and must be a integer multiple of 33 blocks from 33 to 2112 blocks. The default value is 132 blocks. A larger value requires more disk space, but the finer granularity provided by the FastResync maps provides faster resynchronization.

If a snapshot volume is to be moved to a separate disk group (using the disk group move, split and join feature), you must ensure that the plexes its DCO volume are not set up on the same physical disk as the plexes of any volume that is to remain in the original disk group. To ensure this, specify appropriate storage attributes to define the disks that can and / or cannot be used. For example, the following command would allow the DCO plex for the volume `SNAP-vol` to be set up on disk `disk03`, but not on `disk01` or `disk02`:

```
# vxassist -g egdg addlog SNAP-vol logtype=dco \
  dcolen=264 ndcomirror=1 !disk01 !disk02 disk03
```

Note If the DCO plexes of the snapshot volume are configured on disks that also contain the plexes of other volumes, this prevents the snapshot volume from being moved to a different disk group. See “Considerations for Placing DCO Plexes” on page 138 for more information.

5. Perform this step for each snapshot volume and for each snapshot plex in the original volume. It is optional for the data plexes of the original volume. If the `dco_plex_rid` attribute is not set, or if it is set incorrectly on a plex in a snapshot volume, then Persistent FastResync is configured incorrectly, and a full resynchronization is required on snapback. You can omit this step if you chose to reattach all the snapshot volumes in step 2.

For each plex in each volume, use the following command to set the plex's dco_plex_rid attribute to refer to the corresponding plex in the DCO volume.

```
# vxedit [-g diskgroup] set dco_plex_rid=`vxprint -F"%rid" \
  dcologplex` plex
```

For example, to set the `dco_plex_rid` attribute of the plex `SNAP-vol-01` to point to the DCO plex `SNAP-vol_dcl-01`, use the following command:

```
# vxedit -g egdg set dco_plex_rid=`vxprint -F"%rid" \
  SNAP-vol_dcl-01` SNAP-vol-01
```

Note The choice of which DCO plex to associate with a given plex is arbitrary unless the snapshot plex is to be moved along with a snapshot volume to a different disk group. If such is the case, the DCO plex must not be configured on the same physical disk as the plexes of any volume that is to remain in the original disk group. If any DCO plex of a snapshot volume is configured on a disk that also contains the plexes of other volumes, this prevents the snapshot volume from being moved to a different disk group. For more information, see "Considerations for Placing DCO Plexes" on page 138.

6. Perform this step on any snapshot volumes as well as on the original volume.

   Enable Persistent FastResync on the volume using this command:

   ```
   # vxvol [-g diskgroup] set fastresync=on volume
   ```

Backing up Volumes Online

It is important to make backup copies of your volumes. These provide replicas of the data as it existed at the time of the backup. Backup copies are used to restore volumes lost due to disk failure, or data destroyed due to human error. VxVM allows you to back up volumes online with minimal interruption to users.

Two methods of backing up volumes online are described in the following sections: "Backing Up Volumes Online Using Mirrors" and "Backing Up Volumes Online Using Snapshots" on page 228. For information on implementing off-host online backup, see "Configuring Off-Host Processing" on page 285.

Backing Up Volumes Online Using Mirrors

If a volume is mirrored, it can be backed up by taking one of the data plexes offline for a period of time. This removes the need for extra disk space for the purpose of backup only. However, if the volume only has two data plexes, it also removes redundancy of the volume for the duration of the time needed for the backup to take place.

You can perform backup of a mirrored volume on an active system with these steps:

1. Dissociate one of the volume's data plexes (`vol01-01`, for example) using the following command:

   # **`vxplex [-g diskgroup] dis plex`**

   Optionally, stop user activity during this time to improve the consistency of the backup.

2. Create a temporary volume, *tempvol*, that uses the dissociated plex, using the following command:

   # **`vxmake -g diskgroup -U gen vol tempvol plex=plex`**

3. Start the temporary volume, using the following command:

   # **`vxvol [-g diskgroup] start tempvol`**

4. Use `fsck` (or some utility appropriate for the application running on the volume) to clean the temporary volume's contents. For example, you can use this command:

   # **`fsck -F vxfs /dev/vx/rdsk/diskgroup/tempvol`**

5. Perform appropriate backup procedures, using the temporary volume.

6. Stop the temporary volume, using the following command:

   # **`vxvol [-g diskgroup] stop tempvol`**

7. Dissociate the backup plex from its temporary volume, using the following command:

   # **`vxplex [-g diskgroup] dis plex`**

8. Reassociate the backup plex with its original volume to regain redundancy of the volume, using the following command:

   # **`vxplex [-g diskgroup] att original_volume plex`**

9. Remove the temporary volume, using the following command:

   # **`vxedit [-g diskgroup] rm tempvol`**

Note If the file system is active during the period that the temporary volume is created, its contents may be inconsistent. For information on an alternative online backup method using the VxVM snapshot facility, see the next section "Backing Up Volumes Online Using Snapshots."

Backing Up Volumes Online Using Snapshots

Note You can use the procedure described in this section to create a snapshot of a RAID-5 volume and to back it up.

VxVM provides snapshot images of volume devices using `vxassist` and other commands. If the `fsgen` volume usage type is set on a volume that contains a VERITAS File System (VxFS), the snapshot mechanism ensures the internal consistency of the file system that is backed up. For `ufs` and `s5` file system types, there may be inconsistencies between in-memory data and the data in the snapshot image.

There are various procedures for doing backups, depending upon the requirements for integrity of the volume contents. The procedures require a plex that is large enough to store the complete contents of the volume. The plex can be larger than necessary, but if a plex that is too small is used, an incomplete copy results.

The recommended approach to performing volume backup from the command line, or from a script, is to use the `vxassist` command. The `vxassist snapstart`, `snapwait`, and `snapshot` tasks allow you to back up volumes online with minimal disruption to users.

The `vxassist snapshot` procedure consists of two steps:

1. Running `vxassist snapstart` to create a snapshot mirror
2. Running `vxassist snapshot` to create a snapshot volume

The `vxassist snapstart` step creates a write-only backup plex which gets attached to and synchronized with the volume. When synchronized with the volume, the backup plex is ready to be used as a `snapshot` mirror. The end of the update procedure is indicated by the new `snapshot` mirror changing its state to SNAPDONE. This change can be tracked by the `vxassist snapwait` task, which waits until at least one of the mirrors changes its state to SNAPDONE. If the attach process fails, the `snapshot` mirror is removed and its space is released.

Once the `snapshot` mirror is synchronized, it continues being updated until it is detached. You can then select a convenient time at which to create a `snapshot` volume as an image of the existing volume. You can also ask users to refrain from using the system during the brief time required to perform the `snapshot` (typically less than a minute). The amount of time involved in creating the `snapshot` mirror is long in contrast to the brief amount of time that it takes to create the `snapshot` volume.

The online backup procedure is completed by running the `vxassist snapshot` command on a volume with a SNAPDONE mirror. This task detaches the finished `snapshot` (which becomes a normal mirror), creates a new normal volume and attaches the `snapshot` mirror to the `snapshot` volume. The `snapshot` then becomes a normal, functioning mirror and the state of the `snapshot` is set to ACTIVE.

If the `snapshot` procedure is interrupted, the `snapshot` mirror is automatically removed when the volume is started.

To back up a volume with the `vxassist` command, use the following procedure:

1. Create a snapshot mirror for a volume using the following command:

   # **`vxassist [-b] [-g`** ***`diskgroup`*****`] snapstart [nmirror=`*****`N`*****`]`** ***`volume`***

   For example, to create a snapshot mirror of a volume called `voldef`, use the following command:

   # **`vxassist [-g`** ***`diskgroup`*****`] snapstart voldef`**

   The `vxassist snapstart` task creates a write-only mirror, which is attached to and synchronized from the volume to be backed up.

Note By default, VxVM attempts to avoid placing a snapshot mirrors on a disk that already holds any plexes of a data volume. However, this may be impossible if insufficient space is available in the disk group. In this case, VxVM uses any available space on other disks in the disk group. If the snapshot plexes are placed on disks which are used to hold the plexes of other volumes, this may cause problems when you subsequently attempt to move a snapshot volume into another disk group as described in "Considerations for Placing DCO Plexes" on page 138. To override the default storage allocation policy, you can use storage attributes to specify explicitly which disks to use for the snapshot plexes. See "Creating a Volume on Specific Disks" on page 178 for more information.

   If you start `vxassist snapstart` in the background using the `-b` option, you can use the `vxassist snapwait` command to wait for the creation of the mirror to complete as shown here:

   # **`vxassist [-g`** ***`diskgroup`*****`] snapwait`** ***`volume`***

   If `vxassist snapstart` is not run in the background, it does not exit until the mirror has been synchronized with the volume. The mirror is then ready to be used as a plex of a snapshot volume. While attached to the original volume, its contents continue to be updated until you take the snapshot.

   Use the `nmirror` attribute to create as many snapshot mirrors as you need for the snapshot volume. For a backup, you should usually only require the default of one.

   It is also possible to make a snapshot plex from an existing plex in a volume. See "Converting a Plex into a Snapshot Plex" on page 231 for details.

2. Choose a suitable time to create a snapshot. If possible, plan to take the snapshot at a time when users are accessing the volume as little as possible.

3. Create a snapshot volume using the following command:

   # **vxassist [-g *diskgroup*] snapshot [nmirror=*N*] *volume snapshot***

   If required, use the `nmirror` attribute to specify the number of mirrors in the snapshot volume.

   For example, to create a snapshot of `voldef`, use the following command:

   # **vxassist [-g *diskgroup*] snapshot voldef snapvol**

   The `vxassist snapshot` task detaches the finished snapshot mirror, creates a new volume, and attaches the snapshot mirror to it. This step should only take a few minutes. The snapshot volume, which reflects the original volume at the time of the snapshot is now available for backing up, while the original volume continues to be available for applications and users.

   If required, you can make snapshot volumes for several volumes in a disk group at the same time. See "Backing Up Multiple Volumes Using Snapshots" on page 232 for more information.

4. Use `fsck` (or some utility appropriate for the application running on the volume) to clean the temporary volume's contents. For example, you can use this command:

   # **fsck -F vxfs /dev/vx/rdsk/*diskgroup/snapshot***

5. Use a backup utility or operating system command to copy the temporary volume to tape, or to some other appropriate backup media.

When the backup is complete, you have three choices for what to do with the snapshot volume:

- Reattach some or all of the plexes of the snapshot volume with the original volume as described in "Merging a Snapshot Volume (snapback)" on page 232. If FastResync was enabled on the volume before the snapshot was taken, this speeds resynchronization of the snapshot plexes before the backup cycle starts again at step 3.
- Dissociate the snapshot volume entirely from the original volume as described in "Dissociating a Snapshot Volume (snapclear)" on page 233. This may be useful if you want to use the copy for other purposes such as testing or report generation.
- Remove the snapshot volume to save space with this command:

  # **vxedit [-g *diskgroup*] -rf rm *snapshot***

Note Dissociating or removing the snapshot volume loses the advantage of fast resynchronization if FastResync was enabled. If there are no further snapshot plexes available, any subsequent snapshots that you take require another complete copy of the original volume to be made.

Converting a Plex into a Snapshot Plex

In some circumstances, you may find it more convenient to convert an existing plex in a volume into a snapshot plex rather than running `vxassist snapstart`. For example, you may want to do this if you are short of disk space for creating the snapshot plex and the volume that you want to snapshot contains more than two plexes.

Note It is advisable to retain at least two plexes in a volume to maintain data redundancy.

To convert an existing plex into a snapshot plex for a volume on which Persistent FastResync is enabled, use the following command:

```
# vxplex [-g diskgroup] dcoplex=dcologplex convert \
  state=SNAPDONE plex
```

dcologplex is the name of an existing DCO plex that is to be associated with the new snapshot plex. You can use the `vxprint` command to find out the name of the DCO volume as described in "Adding a DCO and DCO Volume" on page 207.

For example, to make a snapshot plex from the plex `trivol-03` in the 3-plex volume `trivol`, you would use the following command:

```
# vxplex dcoplex=trivol_dcl-03 convert state=SNAPDONE trivol-03
```

Here the DCO plex `trivol_dco_03` is specified as the DCO plex for the new snapshot plex.

To convert an existing plex into a snapshot plex in the SNAPDONE state for a volume on which Non-Persistent FastResync is enabled, use the following command:

```
# vxplex [-g diskgroup] convert state=SNAPDONE plex
```

A converted plex is in the SNAPDONE state, and can be used immediately to create a snapshot volume.

Note The last complete regular plex in a volume, an incomplete regular plex, or a dirty region logging (DRL) log plex cannot be converted into a snapshot plex.

Backing Up Multiple Volumes Using Snapshots

To make it easier to create snapshots of several volumes at the same time, the snapshot option accepts more than one volume name as its argument, for example:

```
# vxassist [-g diskgroup] snapshot volume1 volume2 ...
```

By default, each replica volume is named SNAP*number*-*volume*, where number is a unique serial number, and *volume* is the name of the volume for which the snapshot is being taken. This default pattern can be overridden by using the option `-o name=`*pattern*, as described on the vxassist(1M) manual page. For example, the pattern SNAP%v-%d reverses the order of the *number* and *volume* components in the name.

To snapshot all the volumes in a single disk group, specify the option `-o allvols` to `vxassist`:

```
# vxassist -g diskgroup -o allvols snapshot
```

This operation requires that all `snapstart` operations are complete on the volumes. It fails if any of the volumes in the disk group do not have a complete snapshot plex in the SNAPDONE state.

Merging a Snapshot Volume (snapback)

Note The information in this section does not apply to RAID-5 volumes unless they have been converted to a special layered volume layout by the addition of a DCO and DCO volume. See “Attaching a DCO and DCO volume to a RAID-5 Volume” on page 209 for details.

Snapback merges a snapshot copy of a volume with the original volume. One or more snapshot plexes are detached from the snapshot volume and re-attached to the original volume. The snapshot volume is removed if all its snapshot plexes are snapped back. This task resynchronizes the data in the volume so that the plexes are consistent.

Note To enhance the efficiency of the snapback operation, enable FastResync on the volume before taking the snapshot, as described in “Enabling FastResync on a Volume” on page 221.

To merge one snapshot plex with the original volume, use the following command:

```
# vxassist snapback snapshot
```

where *snapshot* is the snapshot copy of the volume.

To merge all snapshot plexes in the snapshot volume with the original volume, use the following command:

```
# vxassist -o allplexes snapback snapshot
```

To merge a specified number of plexes from the snapshot volume with the original volume, use the following command:

```
# vxassist snapback nmirror=number snapshot
```

Here the `nmirror` attribute specifies the number of mirrors in the snapshot volume that are to be re-attached.

Once the snapshot plexes have been reattached and their data resynchronized, they are ready to be used in another `snapshot` operation.

By default, the data in the original volume is used to update the snapshot plexes that have been re-attached. To copy the data from the replica volume instead, use the following command:

```
# vxassist -o resyncfromreplica snapback snapshot
```

Caution Unmount the file system corresponding to the primary volume before using the `resyncfromreplica` option.

Dissociating a Snapshot Volume (snapclear)

The link between a snapshot and its original volume can be permanently broken so that the snapshot volume becomes an independent volume.

If Non-Persistent FastResync is enabled on the original volume, use the following command to dissociate the snapshot volume, *snapshot*:

```
# vxassist snapclear snapshot
```

If Persistent FastResync is enabled, and both the snapshot volume and the original volume are still in the same disk group, use either of the following commands to stop FastResync tracking on both volumes with respect to each other:

```
# vxassist snapclear volume snap_object1
# vxassist snapclear snapshot snap_object2
```

Here `snap_object1` is the snap object in the original volume that refers to the snapshot volume, and `snap_object2` is the snap object in the snapshot volume that refers to the original volume. For example, if `myvol` and `SNAP-myvol` are in the same disk group `mydg`, either of the following commands stops tracking for both `myvol` and `SNAP-myvol`:

```
# vxassist -g mydg snapclear SNAP-myvol myvol_snp
# vxassist -g mydg snapclear myvol SNAP-myvol_snp
```

If you have split or moved the snapshot volume and the original volume into different disk groups, you must run `snapclear` on the each volume separately, specifying the snap object in the volume that points to the other volume:

```
# vxassist snapclear volume snap_object
```


For example, if myvol1 and SNAP-myvol1 are in separate disk groups mydg1 and mydg2 respectively, the following commands stop the tracking on SNAP-myvol1 with respect to myvol1 and on myvol1 with respect to SNAP-myvol1:

```
# vxassist -g mydg2 snapclear SNAP-myvol1 myvol1_snp
# vxassist -g mydg1 snapclear myvol1 SNAP-myvol1_snp
```

Displaying Snapshot Information (snapprint)

The vxassist snapprint command displays the associations between the original volumes and their respective replicas (snapshot copies):

```
# vxassist snapprint [volume]
```

Output from this command is shown in the following examples:

```
# vxassist -g mydg snapprint v1
V  NAME         USETYPE      LENGTH
SS SNAPOBJ      NAME         LENGTH     %DIRTY
DP NAME         VOLUME       LENGTH     %DIRTY

v  v1           fsgen        20480
ss SNAP-v1_snp  SNAP-v1      20480      4
dp v1-01        v1           20480      0
dp v1-02        v1           20480      0

v  SNAP-v1      fsgen        20480
ss v1_snp       v1           20480      0
# vxassist -g mydg snapprint v2
V  NAME         USETYPE      LENGTH
SS SNAPOBJ      NAME         LENGTH     %DIRTY
DP NAME         VOLUME       LENGTH     %DIRTY

v  v2           fsgen        20480
ss --           SNAP-v2      20480      0
dp v2-01        v2           20480      0

v  SNAP-v2      fsgen        20480
ss --           v2           20480      0
```

In this example, Persistent FastResync is enabled on volume v1, and Non-Persistent FastResync on volume v2. Lines beginning with v, dp and ss indicate a volume, detached plex and snapshot plex respectively. The %DIRTY field indicates the percentage of a snapshot plex or detached plex that is dirty with respect to the original volume. Notice

that no snap objects are associated with volume `v2` or with its snapshot volume `SNAP-v2`. See "How Persistent FastResync Works with Snapshots" on page 43 for more information about snap objects.

If a volume is specified, the `snapprint` command displays an error message if no FastResync maps are enabled for that volume.

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:

```
# vxassist [-b] [-g diskgroup] relayout volume [layout=layout] \
  [relayout_options]
```

Note If specified, the `-b` option makes relayout of the volume a background task.

The following are valid destination layout configurations as determined by the tables in "Permitted Relayout Transformations" on page 33:

`concat-mirror`—concatenated-mirror

`concat` or `span`, `nostripe`, `nomirror`–concatenated

`raid5`—RAID-5 (not supported for shared disk groups)

`stripe`—striped

`stripe-mirror`—striped-mirror

For example, the following command changes a concatenated volume to a striped volume with the default number of columns, 2, and stripe unit size, 64k:

```
# vxassist relayout vol02 layout=stripe
```

On occasions, it may be necessary to perform a relayout on a plex rather than on a volume. See "Specifying a Plex for Relayout" on page 236 for more information.

Specifying a Non-Default Layout

You can specify one or more relayout options to change the default layout configuration. Examples of these options are:

`ncol=`*number*—specifies the number of columns

`ncol=+`*number*—specifies the number of columns to add

`ncol=-`*number*—specifies the number of colums to remove

`stripeunit=`*size*—specifies the stripe width

See the vxassist(1M) manual page for more information about relayout options.

The following are some examples of using vxassist to change the stripe width and number of columns for a striped volume in the disk group dbaseg:

```
# vxassist -g dbaseg relayout vol03 stripeunit=64k ncol=6
# vxassist -g dbaseg relayout vol03 ncol=+2
# vxassist -g dbaseg relayout vol03 stripeunit=128k
```

The next example changes a concatenated volume to a RAID-5 volume with four columns:

```
# vxassist -g fsgrp relayout vol04 layout=raid5 ncol=4
```

Specifying a Plex for Relayout

Any layout can be changed to RAID-5 if there are sufficient disks and space in the disk group. If you convert a mirrored volume to RAID-5, you must specify which plex is to be converted. All other plexes are removed when the conversion has finished, 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. You can specify the plex to be converted by naming it in place of a volume:

```
# vxassist relayout plex [layout=layout] [relayout_options]
```

Tagging a Relayout Operation

If you want 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, this relayout is performed as a background task and has the tag myconv:

```
# vxassist -b -g fsgrp -t myconv relayout vol04 layout=raid5 ncol=4
```

See the following sections, “Viewing the Status of a Relayout” and “Controlling the Progress of a Relayout,” for more information about tracking and controlling the progress of relayout.

Viewing the Status of a Relayout

Online relayout operations take some time to perform. You can use the `vxrelayout` command to obtain information about the status of a relayout operation. For example, the command:

```
# vxrelayout status vol04
```

might display output similar to this:

```
STRIPED, columns=5, stwidth=128--> STRIPED, columns=6, stwidth=128
Relayout running, 68.58% completed.
```

In this example, the reconfiguration of a striped volume from 5 to 6 columns is in progress, and is just over two-thirds complete.

See the `vxrelayout`(1M) manual page for more information about this command.

If you specified a task tag to `vxassist` when you started the relayout, you can use this tag with the `vxtask` command to monitor the progress of the relayout. For example, to monitor the task tagged as `myconv`, enter:

```
# 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 altogether (`abort`). If you specified a task tag to `vxassist` when you started the relayout, you can use this tag to specify the task to `vxtask`. For example, to pause the relayout operation tagged as `myconv`, enter:

```
# vxtask pause myconv
```

To resume the operation, use the `vxtask` command:

```
# 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 shown here:

```
# vxrelayout -o bg start vol04
```

Note 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 64-kilobyte region:

```
# vxrelayout -o bg,slow=1000,iosize=64 start vol04
```

The default delay and region size values are 250 milliseconds and 32 kilobytes respectively.

To reverse the direction of relayout operation that is currently stopped, specify the `reverse` keyword to `vxrelayout` as shown in this example:

```
# vxrelayout -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 the `vxrelayout`(1M) and vxtask(1M) manual pages for more information about these commands. See "Managing Tasks with vxtask" on page 201 for more information about controlling tasks in VxVM.

Converting Between Layered and Non-Layered Volumes

The `vxassist convert` command transforms volume layouts between layered and non-layered forms:

```
# vxassist [-b] convert volume [layout=layout] [convert_options]
```

Note If specified, the `-b` option makes conversion of the volume 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

Volume conversion can be used before or after performing online relayout to achieve a larger number of transformations than would otherwise be possible. During relayout process, a volume may also be converted into a layout that is intermediate to the one that

is desired. 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:

```
# vxassist relayout vol1 ncol=5
```

When the relayout has completed, use the `vxassist convert` command to change the resulting layered striped-mirror volume to a non-layered mirrored-stripe:

```
# vxassist 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 crash occurred during the first stage of a two-stage relayout and convert operation, only the first stage will be completed. You must run `vxassist convert` manually to complete the operation.

Administering Hot-Relocation 9

Introduction

If a volume has a disk I/O failure (for example, because 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.

Note 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. For more information, see the following section, "How Hot-Relocation works."

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 as described in "Removing and Replacing Disks" on page 84.

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.

Note 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:

1. `vxrelocd` informs the system administrator (and other nominated users, see "Modifying the Behavior of Hot-Relocation" on page 257) by electronic mail of the failure and which VxVM objects are affected. See "Partial Disk Failure Mail Messages" on page 245 and "Complete Disk Failure Mail Messages" on page 246 for more information.

2. `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.

3. 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.

4. 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.

Note Hot-relocation does not guarantee the same layout of data or the same performance after relocation. The system administrator can make configuration changes 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).
- If you use `vxdiskadm` to remove a disk that contains subdisks of a volume.
- 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 log plex.

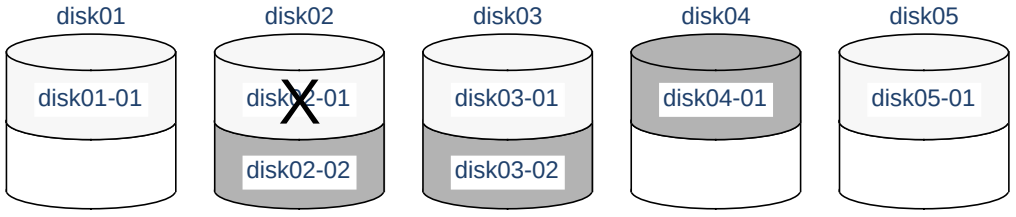
See the `xvrelocd`(1M) manual page for more information about the hot-relocation daemon.

"Example of Hot-Relocation for a Subdisk in a RAID-5 Volume" on page 244 illustrates the hot-relocation process in the case of the failure of a single subdisk of a RAID-5 volume.

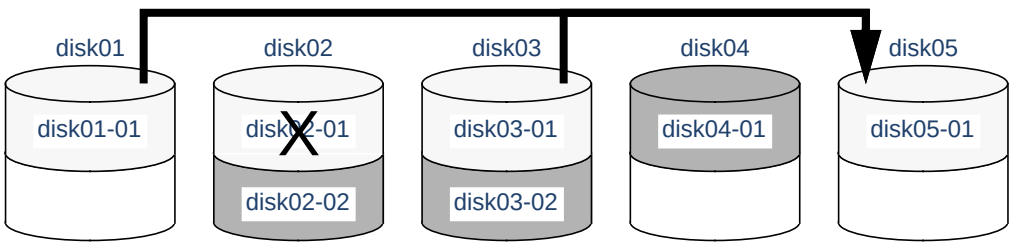
Example of Hot-Relocation for a Subdisk in a RAID-5 Volume

a) Disk group contains five disks. Two RAID-5 volumes are configured across four of the disks. One spare disk is available for hot-relocation.

b) Subdisk disk02-01 in one RAID-5 volume fails. Hot-relocation replaces it with subdisk disk05-01 that it has created on the spare disk, and then initiates recovery of the RAID-5 volume.



c) RAID-5 recovery recreates subdisk disk02-01's data and parity on subdisk disk05-01 from the data and parity information remaining on subdisks disk01-01 and disk03-01.



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
```

See “Modifying the Behavior of Hot-Relocation” on page 257 for information on how to send the mail to users other than `root`.

You can determine which disk is causing the failures in the above example message by using the following command:

```
# vxstat -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 disk01-04               0         0
sd disk01-06               0         0
sd disk02-03               1         0
sd disk02-04               1         0
```

This example shows failures on reading from subdisks `disk02-03` and `disk02-04` of `disk02`.

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 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 84).

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:
  disk02

failed plexes:
  home-02
  src-02
  mkting-01

failing disks:
  disk02
```

This message shows that `disk02` 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).

As described in "Partial Disk Failure Mail Messages" on page 245, the problem can be a cabling error. If the problem is not a cabling error, replace the disk (see "Removing and Replacing Disks" on page 84).

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.

The free space used in hot-relocation must not have been excluded from hot-relocation use. Disks can be excluded from hot-relocation use by using `vxdiskadm`, `vxedit` or the VERITAS Enterprise Administrator (VEA).

You can designate one or more disks as hot-relocation spares within each disk group. Disks can be designated as spares by using `vxdiskadm`, `vxedit`, or the VEA. Disks designated as spares do not participate in the free space model and should not have storage space allocated on them.

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, target, 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. A disk under the same target as the failed disk is closer than one on a different target.

Hot-relocation tries to move all subdisks from a failing drive to the same destination disk, if possible.

If the failing disk is a root disk, hot-relocation only works if it can relocate all of the file systems to the same disk. If none are found, the system administrator is notified through email.

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.

- To find out which disks are spares or are excluded from hot-relocation, see “Displaying Spare Disk Information” on page 248.

You can prepare for hot-relocation by designating one or more disks per disk group as hot-relocation spares.

- To designate a disk as being a hot-relocation spare for a disk group, see “Marking a Disk as a Hot-Relocation Spare” on page 248.
- To remove a disk from use as a hot-relocation spare, see “Removing a Disk from Use as a Hot-Relocation Spare” on page 249.

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.

- To exclude a disk from hot-relocation use, see “Excluding a Disk from Hot-Relocation Use” on page 250.
- To make a disk available for hot-relocation use, see “Making a Disk Available for Hot-Relocation Use” on page 251.

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 252).

After a successful relocation, remove and replace the failed disk as described in “Removing and Replacing Disks” on page 84).

Displaying Spare Disk Information

Use the following command to display information about spare disks that are available for relocation:

```
# vxdg spare
```

The following is example output:

```
GROUP       DISK        DEVICE      TAG         OFFSET  LENGTH    FLAGS
rootdg      disk02      c0t2d0s2    c0t2d0s2    0       658007    s
```

Here `disk02` is the only disk designated as a spare. The `LENGTH` field indicates how much spare space is currently available on `disk02` 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 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 except for the free space on the disks that were previously excluded from hot-relocation use.

To designate a disk as a hot-relocation spare, enter the following command:

```
# vxedit set spare=on diskname
```

For example, to designate `disk01` as a spare, enter the following command:

```
# vxedit set spare=on disk01
```

You can use the `vxdisk list` command to confirm that this disk is now a spare; `disk01` 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.

Alternatively, you can use `vxdiskadm` to designate a disk as a hot-relocation spare:

1. Select menu item `12 (Mark a disk as a spare for a disk group)` from the `vxdiskadm` main menu.

2. At the following prompt, enter a disk name (such as `disk01`):

   ```
   Menu: VolumeManager/Disk/MarkSpareDisk

   Use this operation to mark a disk as a spare for a disk
   group.
   This operation takes, as input, a disk name. This is the same
   name that you gave to the disk when you added the disk to the
   disk group.

   Enter disk name [<disk>,list,q,?] disk01
   ```

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 set spare=off diskname
```


For example, to make `disk01` available for normal use, use the following command:

```
# vxedit set spare=off disk01
```

Alternatively, you can use `vxdiskadm` to remove a disk from the hot-relocation pool:

1. Select menu item `13 (Turn off the spare flag on a disk)` from the `vxdiskadm` main menu.

2. At the following prompt, enter the name of a spare disk (such as `disk01`):

   ```
   Menu: VolumeManager/Disk/UnmarkSpareDisk

   Use this operation to turn off the spare flag on a disk.
   This operation takes, as input, a disk name. This is the same
   name that you gave to the disk when you added the disk to the
   disk group.

   Enter disk name [<disk>,list,q,?] disk01
   ```

   The `vxdiskadm` program displays the following confirmation:

   ```
   Disk disk01 in rootdg 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 disk_group set nohotuse=on diskname
```

Alternatively, using `vxdiskadm`:

1. Select menu item `15 (Exclude a disk from hot-relocation use)` from the `vxdiskadm` main menu.

2. At the following prompt, enter the disk name (such as `disk01`):

   ```
   Exclude a disk from hot-relocation use
   Menu: VolumeManager/Disk/UnmarkSpareDisk

   Use this operation to exclude a disk from hot-relocation use.
   This operation takes, as input, a disk name. This is the same
   name that you gave to the disk when you added the disk to the
   disk group.
   ```

```
Enter disk name [<disk>,list,q,?] disk01
```

The `vxdiskadm` program displays the following confirmation:

```
Excluding disk01 in rootdg 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 disk_group set nohotuse=off diskname
```

Alternatively, using `vxdiskadm`:

1. Select menu item `16 (Make a disk available for hot-relocation use)` from the `vxdiskadm` main menu.

2. At the following prompt, enter the disk name (such as **`disk01`**):

```
Menu: VolumeManager/Disk/UnmarkSpareDisk

Use this operation to make a disk available for hot-relocation
use. This only applies to disks that were previously excluded
from hot-relocation use. This operation takes, as input, a disk
name. This is the same name that you gave to the disk when you
added the disk to the disk group.

Enter disk name [<disk>,list,q,?] disk01
```

The `vxdiskadm` program displays the following confirmation:

```
Making disk01 in rootdg 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 and Unrelocating 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 off 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 disk02-03 from plex home-02.
Dev_offset 0 length 1164 dm_name disk02 da_name c0t5d0s2.
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 disk02-03 relocated to disk05-01,
but not yet recovered.
```

Before you move any relocated subdisks, fix or replace the disk that failed (as described in "Removing and Replacing Disks" on page 84). Once this is done, you can move a relocated subdisk back to the original disk as described in the following sections.

Caution During subdisk move operations, RAID-5 volumes are not redundant.

Moving and Unrelocating Subdisks using vxdiskadm

To move the hot-relocated subdisks back to the disk where they originally resided after the disk has been replaced following a failure, use the following procedure:

1. Select menu item `14 (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.

   ```
   While unrelocating the subdisks, you can choose to move the
   subdisks to a different disk from 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 disk10 back to disk10 of disk group rootdg.
   Continue with operation? [y,n,q,?] (default: y)
   ```

   A status message is displayed at the end of the operation.

   ```
   Unrelocate to disk disk10 is complete.
   ```


As an alternative to this procedure, use either the `vxassist` command or the vxunreloc command directly, as described in "Moving and Unrelocating subdisks using vxassist" and "Moving and Unrelocating Subdisks using vxunreloc" on page 254.

Moving and Unrelocating subdisks using vxassist

You can use the `vxassist` command to move and unrelocate subdisks. For example, to move the relocated subdisks on `disk05` belonging to the volume `home` back to `disk02`, enter the following command:

```
# vxassist -g rootdg move home !disk05 disk02
```

Here, `!disk05` specifies the current location of the subdisks, and `disk02` 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 and Unrelocating 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. Refer to the `vxunreloc`(1M) manual page for more information.

The following examples demonstrate the use of vxunreloc.

Moving hot-relocated subdisks back to their original disk

Assume that disk01 failed and all the subdisks were relocated. After disk01 is replaced, vxunreloc can be used to move all the hot-relocated subdisks back to disk01.

```
# vxunreloc -g newdg disk01
```

Moving hot-relocated subdisks 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 disk01 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, e.g, disk05. If you want to move all the hot-relocated subdisks back to the new disk, the following command can be used:

```
# vxunreloc -g newdg -n disk05 disk01
```

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 disk01 failed and the subdisks were relocated and that you want to move the hot-relocated subdisks to disk05 where some subdisks already reside. You can use the force option to move the hot-relocated subdisks to disk05, but not to the exact offsets:

```
# vxunreloc -g newdg -f -n disk05 disk01
```


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 `disk01` failed and a subdisk named `disk01-01` was moved to `disk02`, and then `disk02` experienced disk failure, all of the subdisks residing on it, including the one which was hot-relocated to it, will be moved again. When `disk02` was replaced, a `vxunreloc` operation for `disk02` will do nothing to the hot-relocated subdisk `disk01-01`. However, a replacement of `disk01` followed by a `vxunreloc` operation, moves `disk01-01` back to `disk01` 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 `disk01` in the `rootdg` disk group, use the following command:

```
# vxprint -g rootdg -se 'sd_orig_dmname="disk01"'
```

Restarting vxunreloc After Errors

`vxunreloc` moves subdisks in three phases:

1. `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.

2. `vxunreloc` moves the data from each subdisk to the corresponding newly created subdisk on the destination disk.

3. 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.

Caution 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 `vxrelocd` is running. You leave hot-relocation turned on so that you can take advantage of this feature if a failure occurs. However, if you choose to disable this feature (perhaps because you do not want the free space on some of your disks to be used for relocation), prevent `vxrelocd` from starting at system startup time.

You can stop hot-relocation at any time by killing the `vxrelocd` process (this should not be done while a hot-relocation attempt is in progress).

You can make some minor changes to the way `vxrelocd` behaves by either editing the `vxrelocd` line in the startup file that invokes `vxrelocd` (`/etc/rc2.d/S95vxvm-recover`), or by killing the existing `vxrelocd` process and restarting it with different options. After making changes to the way `vxrelocd` is invoked in the startup file, you need to reboot the system so that the changes go into effect. If you choose to kill and restart the daemon instead, make sure that hot-relocation is not in progress when you kill the `vxrelocd` process. You should also restart the daemon immediately so that hot-relocation can take effect if a failure occurs.

You can alter `vxrelocd` behavior as follows:

- To prevent vxrelocd starting, comment out the entry that invokes it in the startup file:

  ```
  # nohup vxrelocd root &
  ```

- 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 &
  ```

- 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.

When executing `vxrelocd` manually, either include `/etc/vx/bin` in your PATH or specify `vxrelocd`'s absolute pathname, for example:

```
# PATH=/etc/vx/bin:$PATH
# export PATH
# nohup vxrelocd root &
```

or

```
# nohup /etc/vx/bin/vxrelocd root user1 user2 &
```

See the `vxrelocd`(1M) manual page for more information.

Administering Cluster Functionality 10

Introduction

A cluster consists of a number of hosts or *nodes* that share a set of disks. The main benefits of cluster configurations are:

- *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.

- *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 functionality of VERITAS Volume Manager (VxVM) allows up to 16 nodes in a cluster to simultaneously access and manage a set of disks under VxVM control (VM disks). The same logical view of disk configuration and any changes to this is available on all the nodes. When the cluster functionality is enabled, all the nodes in the cluster can share VxVM objects. This chapter discusses the cluster functionality that is provided with VxVM.

Note You need an additional license to use this feature.

This chapter does not discuss VERITAS Cluster File System™ nor cluster management software such as VERITAS Cluster Server. See the documentation provided with these products for more information about them.

Note VERITAS Cluster File System and VERITAS Cluster Server are separately licensed products. They are not included with VERITAS Volume Manager.

Overview of Cluster Volume Management

In recent years, tightly coupled cluster systems have become increasingly popular 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*.

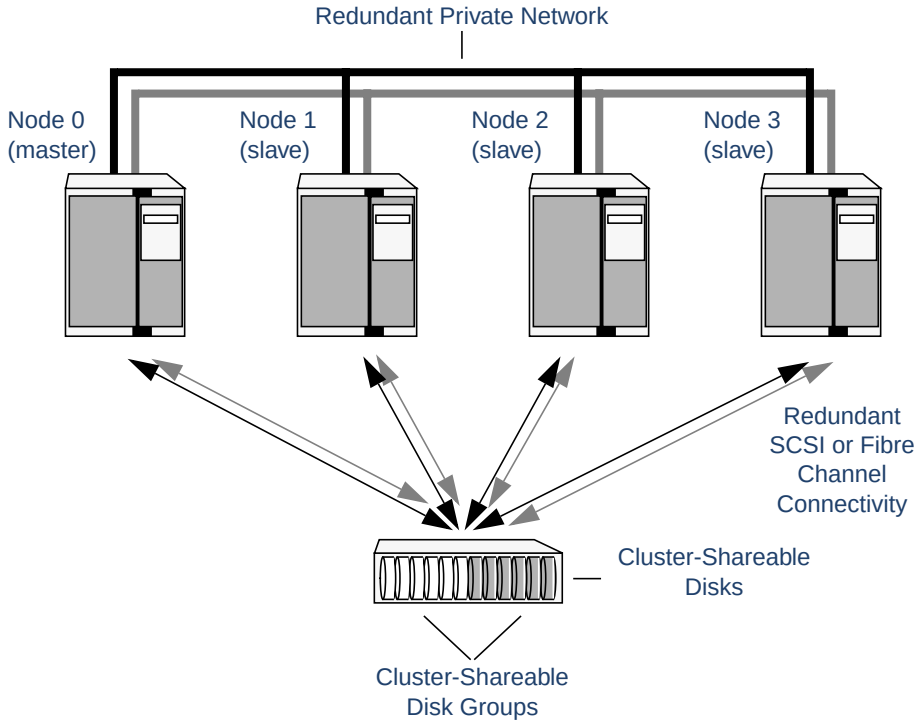
Another major advantage of clustered systems is their ability to reduce contention for system resources caused by activities such as backup, decision support and report generation. Businesses can derive enhanced value from their investment in cluster systems by performing such operations on lightly loaded 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*.

The cluster functionality of VxVM works together with the *cluster monitor* daemon that is provided by the host operating system or by VCS. 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 with support for cluster functionality. When a node *joins* a cluster, it gains access to shared disks. When a node *leaves* a cluster, it no longer has access to shared disks. A node joins a cluster when the cluster monitor is started on that node.

“Example of a 4-Node Cluster” on page 261illustrates 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). 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. Each node has two independent paths to these disks, which are configured in one or more cluster-shareable disk groups.

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*.

Example of a 4-Node 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 cluster 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.

Note You must run commands that configure or reconfigure VxVM objects *on the master node*. Tasks that must be initiated from the master node include setting up shared disk groups, creating and reconfiguring volumes, and performing snapshot operations.

VxVM determines that 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. In "Example of a 4-Node Cluster," node 0 is the master node and nodes 1, 2 and 3 are slave nodes.

Private and Shared Disk Groups

Two types of disk groups are defined:

- *Private disk groups*—belong to only one node. A private disk group is only imported by one system. Disks in a private disk group may be physically accessible from one or more systems, but access is restricted to one system only. The root disk group (`rootdg`) is always a private disk group.
- *Shared disk groups*—shared by all nodes. A shared (or *cluster-shareable*) disk group is imported by all cluster nodes. Disks in a shared disk group must be physically accessible from all systems that may join the cluster.

In a cluster, most disk groups are shared. Disks in a shared disk group are accessible from all nodes in a cluster, allowing applications on multiple cluster nodes to simultaneously access the same disk. A volume in a shared disk group can be simultaneously accessed by more than one node in the cluster, subject to licensing and disk group activation mode restrictions.

You can use the `vxdg` command to designate a disk group as cluster-shareable as described in "Importing Disk Groups as Shared" on page 279. 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. As system administrator, you can also import or deport a shared disk group at any time; the operation takes places in a distributed fashion on all nodes.

Each physical disk 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 have the `noautoimport` 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 imports the same shared disk groups as the master and joins the cluster. When a node leaves the cluster, 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 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 as discussed in "Activation Modes of Shared Disk Groups." 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.

Note 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 `exclusive-write`, `read-only`, `shared-read`, `shared-write`, and `off` (inactive). These activation modes are described in detail in the table "Activation Modes for Shared Disk Groups" on page 263.

Note Disk group activation was a new feature in VERITAS Volume Manager 3.0. To maintain compatibility with previous releases, the default activation mode for shared disk groups is `shared-write`.

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.

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.

Activation Modes for Shared Disk Groups

Activation Mode	Description
`exclusive-write (ew)`	The node has exclusive write access to the disk group. No other node can activate the disk group for write access.
`read-only (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.
`shared-read (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.
`shared-write (sw)`	The node has write access to the disk group.
`off`	The node has neither read nor write access to the disk group. Query operations on the disk group are permitted.

The table "Allowed and Conflicting Activation Modes" summarizes the allowed and conflicting activation modes for shared disk groups:

Allowed and Conflicting Activation Modes

Disk group activated in cluster as...	Attempt to activate disk group on another node as...			
	exclusive-write	read-only	shared-read	shared-write
exclusive-write	Fails	Fails	Succeeds	Fails
read-only	Fails	Succeeds	Succeeds	Fails
shared-read	Succeeds	Succeeds	Succeeds	Succeeds
shared-write	Fails	Fails	Succeeds	Succeeds

To place activation modes under user control, 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 `exclusive-write`, `read-only`, `shared-read`, `shared-write`, or `off`.

Note When enabling activation using the defaults file, it is recommended that the defaults file be identical on all nodes in the cluster. Otherwise, the results of activation are unpredictable.

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.

If the defaults file is edited while the `vxconfigd` daemon is already running, the `vxconfigd` process must be restarted for the changes in the defaults file to take effect.

Note If the default activation node 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 as described in "Listing Shared Disk Groups" on page 277.

You can also use the `vxdg` command to change the activation mode on a shared disk group as described in "Changing the Activation Mode on a Shared Disk Group" on page 281.

For a description of how 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 281 and “Setting Exclusive Open Access to a Volume by a Node” on page 282.

Connectivity Policy of Shared Disk Groups

The nodes in a cluster must always agree on the status of a disk. In particular, if one node cannot write to a given disk, all nodes must stop accessing that disk before the results of the write operation are returned to the caller. Therefore, if a node cannot contact a disk, it should contact another node to check on the disk’s status. If the disk fails, no node can access it and the nodes can agree to detach the disk. If the disk does not fail, but rather the access paths from some of the nodes fail, the nodes cannot agree on the status of the disk. Either of the following policies for resolving this type of discrepancy may be applied:

- Under the *global* connectivity policy, the detach occurs cluster-wide (globally) if any node in the cluster reports a disk failure. This is the default policy.
- Under the *local* connectivity policy, in the event of disks failing, the failures are confined to the particular nodes that saw the failure. Note that an attempt is made to communicate with all nodes in the cluster to ascertain the disks’ usability. If all nodes report a problem with the disks, a cluster-wide detach occurs.

See “Setting the Connectivity Policy on a Shared Disk Group” on page 281 for information on how to use the `vxedit` command to set the connectivity policy on a shared disk group.

Limitations of Shared Disk Groups

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. Online relayout is supported provided that it does not involve RAID-5 volumes.

The root disk group (`rootdg`) cannot be made cluster-shareable. It must be private.

Only raw device access may be performed via the cluster functionality of VxVM. 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.

If a shared disk group contains unsupported objects, deport it and then re-import 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.

Cluster 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 cluster functionality of VxVM requires that a cluster monitor (such as provided by SunCluster™ or by GAB (Group Membership and Atomic Broadcast) in VCS) has been configured. For a VCS environment, use the `vxcvmconfig` command on any node to configure the cluster to use the cluster functionality of VxVM. The `vxcvmconfig` command is provided in the `VRTScavf` package, which is included with VERITAS SANPoint Foundation Suite™ HA, but not 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. Once the join for a given node is complete, that node has access to the shared disks.

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 have completed. Volume reconfigurations (see "Volume Reconfiguration" on page 268) 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.

For more information on cluster reconfiguration, see "vxclust Utility" on page 267 and "vxclustadm Utility" on page 268.

vxclust Utility

Note `vxclust` is used when SunCluster™ acts as the cluster monitor.

Every time there is a cluster reconfiguration, every node currently in the cluster runs the `vxclust` utility at each of several well-orchestrated steps. cluster monitor facilities ensure that the same step is executed on all nodes at the same time. A given step only starts when the previous one has completed on all nodes. At each step in the reconfiguration, the `vxclust` utility determines what the cluster functionality of VxVM should do next. After informing VxVM of its next action, the `vxclust` utility waits for the outcome (success, failure, or retry) and communicates that to the cluster monitor.

If a node does not respond to a the `vxclust` utility request within a specific timeout period, that node aborts. The `vxclust` utility then decides whether to restart the reconfiguration or give up, depending on the circumstances. If the cause of the reconfiguration is a local, uncorrectable error, `vxclust` gives up. If a node cannot complete an operation because another node has left, the surviving node times out. In this case, the `vxclust` utility requests a reconfiguration with the expectation that another node will leave. If no other node leaves, the `vxclust` utility causes the local node to leave.

If a reconfiguration step fails, the `vxclust` utility returns an error to the cluster monitor. The cluster monitor may decide to abort the node, causing its immediate departure from the cluster. Any I/O in progress to the shared disk fails and access to the shared disks is stopped.

vxclust decides what actions to take when it is informed of changes in the cluster. If a new master node is required (due to failure of the previous master), vxclust determines which node becomes the new master.

vxclustadm Utility

Note vxclustadm is only used with VERITAS Cluster Server (VCS).

The vxclustadm command provides an interface to the cluster functionality of VxVM when VCS is used as the cluster monitor. It is also called by VCS during cluster startup and shutdown. In the absence of a cluster monitor, vxclustadm can also be used to activate or deactivate the cluster functionality of VxVM on any node in a cluster.

The startnode keyword to vxclustadm starts cluster 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 cluster functionality on a node. It waits for all outstanding I/O to complete and for all applications to close shared volumes.

The abortnode keyword terminates cluster activity on a node. It does not wait for outstanding I/O to complete nor for applications to close shared volumes.

The nodestate keyword reports the state of a cluster node: out of cluster, joining, or cluster member.

See the vxclustadm(1M) manual page for full information about vxclustadm and for examples of its usage.

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 (see “vxconfigd Daemon” on page 269) play an active role in volume reconfiguration. For reconfiguration to succeed, a vxconfigd daemon must be running on each of the nodes.

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. 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 vxclust utility under SunCluster or from the kernel when running VCS. 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 vxclust utility (SunCluster) or the kernel (for VCS) 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 vxconfigd daemon on each node

On the master node, the vxconfigd daemon sets up the shared configuration by importing shared disk groups, and informs the vxclust utility (for SunCluster) or the kernel (for VCS) 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 vxconfigd daemon notifies the kernel on all the other nodes. The master node then performs any necessary cleanup. If the master node leaves the cluster, the kernels choose a new master node and the vxconfigd daemons on all nodes are notified of the choice.

The vxconfigd daemon also participates in volume reconfiguration as described in "Volume Reconfiguration" on page 268.

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 and other nodes cannot join the cluster until it is restarted. If a cluster monitor is enabled, it may try to fail over VxVM to another node in the cluster. It is therefore inadvisable to stop the vxconfigd daemon on any cluster node.

Different actions are taken depending on which node the vxconfigd daemon is stopped:

- 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 `vxclust` utility (for SunCluster) or the kernel (for VCS) determines that the `vxconfigd` daemon has stopped on a node, `vxconfigd` is restarted automatically. If it necessary to restart vxconfigd manually in a VCS controlled cluster to resolve a VxVM issue, use this procedure:

1. Use the following command to disable failover on any service groups that contain VxVM objects:

   # **`hagrp -freeze group`**

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 group`**

Note 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.

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 cluster 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.

Note 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.

Node Abort

If a node does not leave a cluster cleanly, this is because it crashed or because some cluster component made the node leave on an emergency basis. The ensuing cluster reconfiguration calls the VxVM abort function. This procedure immediately attempts to halt all access to shared volumes, although it does wait until pending I/O from or to the disk completes.

I/O operations that have not yet been started are failed, and the shared volumes are removed. Applications that were accessing the shared volumes therefore fail with errors.

After a node abort or crash, shared volumes must be recovered, either by a surviving node or by a subsequent cluster restart, because it is very likely that there are unsynchronized mirrors.

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.

Upgrading Cluster Functionality

The rolling upgrade feature allows you to upgrade the version of VxVM running in a cluster without shutting down the entire cluster. To install the new version of VxVM running on a cluster, make one node leave the cluster, upgrade it, and then join it back into the cluster. This operation is repeated for each node in the cluster.

Each VERITAS Volume Manager release starting with Release 3.1 has a *cluster protocol version number* associated with it. The cluster protocol version is not the same as the release number or the disk group version number. The cluster protocol version is stored in the `/etc/vx/volboot` file. During a new installation of VxVM, the `vxdctl init` command creates the `volboot` file and sets the cluster protocol version to the highest supported version.

Each new VERITAS Volume Manager release supports two cluster protocol versions. The lower version number corresponds to a previous VERITAS Volume Manager release. This has a fixed set of features and communication protocols. The higher version number corresponds to the new release of VxVM which has a new set of these features. If the new release of VxVM does not have any functional or protocol changes, but only bug fixes or minor changes, the cluster protocol version remains unchanged. In this case, the cluster protocol version does not need to be upgraded.

During a rolling upgrade, each node must be shut down and the VERITAS Volume Manager release with the latest cluster protocol version must be installed. All nodes that have the new release of VxVM continue to use the lower level version. A slave node that has the new cluster protocol version installed tries to join the cluster. If the new cluster protocol version is not in use on the master node, it rejects the join and provides the current cluster protocol version to the slave node. The slave retries the join with the cluster protocol version provided by the master node. If the join fails at this point, the cluster protocol version on the master node is out of range of the protocol versions supported by the joining slave. In such a situation, you must upgrade the remainder of the cluster through each intermediate release of VxVM to reach the latest supported cluster protocol version.

Once you have installed the new release on all nodes, run the `vxdctl upgrade` command on the master node to switch the cluster to the higher cluster protocol version. See “Upgrading the Cluster Protocol Version” on page 283 for more information.

Dirty Region Logging (DRL) 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 DRL and describes how DRL behaves in a cluster environment. For more information on DRL, see "Dirty Region Logging (DRL)" on page 38.

In a cluster environment, the VxVM implementation of DRL differs slightly from the normal implementation. The following sections outline some of the differences and discuss some aspects of the cluster environment implementation.

Header Compatibility

Except for the addition of a cluster-specific magic number, DRL headers in a cluster environment are the same as their non-clustered counterparts.

Dirty Region Log Format and Size Requirements

As in the non-clustered case, the dirty region log in clusters exists on a log subdisk in a mirrored volume.

A dirty region log on a system without cluster support has a recovery map and a single active map. A dirty region log in a cluster, however, has one recovery map and one active map for each node in the cluster). The cluster functionality of VxVM places the recovery map at the beginning of the log.

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.

The log size depends on the volume size and the number of nodes. The log must be large enough to accommodate all maps (one map per node plus a recovery map). Each map must be one block long for each 2 gigabytes of volume size. For a 2-gigabyte volume in a 2-node cluster, a log size of 2 blocks (one block per map) is sufficient; this is the minimum log size. A 4-gigabyte volume in a 4-node cluster requires a log size of 10 blocks, and so on.

It is possible to re-import 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 by 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 are responsible for allocating a new log of sufficient size.

To increase the size of an existing DRL log so that it can accommodate maps for extra nodes, use the `vxplex -o rm dis` command to detach and remove the log plex, and then use the `vxassist addlog` command to recreate the log.

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 before the node can 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, their respective recoveries and recovery map updates can compete with each other. 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 procedures for administering the cluster functionality of VxVM.

Note Most VxVM commands require superuser or equivalent privileges.

Requesting the Status of a Cluster Node

The `vxdctl` utility controls the operation of the `vxconfigd` volume configuration daemon. The `-c` option can be used to request cluster information. To determine whether the `vxconfigd` daemon is enabled and/or running, use the following command:

```
# vxdctl -c mode
```

This produces one of the following output messages depending on the current status of the cluster node:

```
mode: enabled: cluster active - MASTER
mode: enabled: cluster active - SLAVE
mode: enabled: cluster active - role not set
mode: enabled: cluster inactive
```

Note If the `vxconfigd` daemon is disabled, no cluster information is displayed.

See the `vxdctl`(1M) manual page for more information.

Determining if a Disk is Shareable

The `vxdisk` utility manages VxVM disks. To use the `vxdisk` utility to determine whether a disk 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).

A portion of the output from this command (for the device `c4t1d0`) is shown here:

```
Device:    c4t1d0
devicetag: c4t1d0
type:      sliced
clusterid: cvm2
disk:      name=disk01 id=963616090.1034.cvm2
timeout:   30
group:     name=rootdg 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`. When a node is not joined to the cluster, the `flags` field contains the `autoimport` flag instead of `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
rootdg       enabled          774215886.1025.teal
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
```

where *diskgroup* is the disk group name.

For example, the output for the command vxdg list group1 on the master is as follows:

```
Group:     group1
dgid:      774222028.1090.teal
import-id: 32768.1749
flags:     shared
version:   70
local-activation: exclusive-write
cluster-actv-modes: node0=ew node1=off
detach-policy: local
copies:    nconfig=default nlog=default
config:    seqno=0.1976 permlen=1456 free=1448 templen=6 loglen=220
config disk c1t0d0s2 copy 1 len=1456 state=clean online
config disk c1t1d0s2 copy 1 len=1456 state=clean online
log disk c1t0d0s2 copy 1 len=220
log disk c1t1d0s2 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. Also note the local-activation and cluster-actv-modes fields. These display the activation mode for this node and for each node in the cluster respectively.

Creating a Shared Disk Group

Note Shared disk groups can only be created on the master node.

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=]devicename
```

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).

Caution 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 on two nodes at the same time, the results are undefined. Perform all configuration on one node only, and preferably on the master node.

Forcibly Adding a Disk to a Disk Group

Note Disks can only be forcibly added to a shared disk group on the master node.

If VxVM does not add a disk to an existing disk group because that disk is not attached to the same nodes as the other disks in the disk group, you can forcibly add the disk using the following command:

```
# vxdg -f adddisk -g diskgroup [diskname=]devicename
```

Caution Only use the force option(`-f`) if you are fully aware of the consequences such as possible data corruption.

Importing Disk Groups as Shared

Note Shared disk groups can only be imported on the master node.

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.

Caution 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
  ```

- Some of the nodes to which disks in the disk group are attached are not currently in the cluster, 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.

Converting a Disk Group from Shared to Private

Note Shared disk groups can only be deported on the master node.

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 Disk Groups

As described in "Moving Objects Between Disk Groups" on page 140, 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.

Note You can only move objects between shared disk groups on the master node. You cannot move objects between private and shared disk groups.

Splitting Disk Groups

As described in "Splitting Disk Groups" on page 142, 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.

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 only split a shared disk group or create a shared target disk group on the master node.

For a description of the other options, see "Moving Objects Between Disk Groups" on page 140.

Joining Disk Groups

As described in "Joining Disk Groups" on page 143, 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 must perform the join on the master node.

Note You cannot join a private disk group and a shared disk group.

Changing the Activation Mode on a Shared Disk Group

Note The activation mode for access by a cluster node to a shared disk group is set 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 `exclusive-write` or `ew`, `read-only` or `ro`, `shared-read` or `sr`, `shared-write` or `sw`, or `off`. See "Activation Modes of Shared Disk Groups" on page 263 for more information.

Setting the Connectivity Policy on a Shared Disk Group

Note The connectivity policy for a shared disk group can only be set on the master node.

The `vxedit` command may be used to set either the `global` or `local` connectivity policy for a shared disk group:

```
# vxedit -g diskgroup set diskdetpolicy=global|local diskgroup
```

See "Connectivity Policy of Shared Disk Groups" on page 265 for more information.

Creating Volumes with Exclusive Open Access by a Node

Note Volumes with exclusive open access can only be created on the master 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.

Setting Exclusive Open Access to a Volume by a Node

Note Exclusive open access on a volume can only be set on the master node. 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.

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: 40
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 40
```

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 out put similar to the following:

```
Support information:
  vold_vrsn: 11
  dg_minimum: 60
  dg_maximum: 70
  kernel: 10
  protocol_minimum: 30
  protocol_maximum: 40
  protocol_current: 40
```

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: 30, maxprotoversion: 40
```

Upgrading the Cluster Protocol Version

Note The cluster protocol version can only be updated on the master node.

After all the nodes in the cluster have been updated with a new cluster protocol, you can upgrade the entire cluster using the following command on the master node:

```
# vxdctl upgrade
```


Recovering Volumes in Shared Disk Groups

Note Volumes can only be recovered on the master node.

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.

Note 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 an integer. 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 group1 -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.

Configuring Off-Host Processing 11

Introduction

Off-host processing allows you to 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, a replica of the database can be set up using the snapshots. Operations such as decision support analysis and business reporting do not require access to up-to-the-minute information. This means that 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 provides developers, system testers and QA groups with a realistic basis for testing the robustness, integrity and performance of new applications.
- *Database Error Recovery*—Logic errors caused by an administrator or an application program can compromise the integrity of a database. 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.

Off-host processing is made possible by using the FastResync and disk group move, split and join features of VxVM. These features are described in the following sections.

You can also use such solutions in conjunction with the cluster functionality of VxVM.

For implementation guidelines, see “Implementing Off-Host Processing Solutions” on page 287.

FastResync of Volume Snapshots

Note You may need an additional license to use this feature.

VxVM allows you to take multiple snapshots of your data at the level of a volume. A snapshot volume contains a stable copy of a volume's data at a given moment in time that you can use for online backup or decision support. If FastResync is enabled on a volume, VxVM uses a *FastResync map* to keep track of which blocks are updated in the volume and in the snapshot. If the data in one mirror is not updated for some reason, it becomes out-of-date, or *stale*, with respect to the other mirrors in the volume. The presence of the FastResync map means that only those updates that the mirror has missed need be reapplied to resynchronize it with the volume. A full, and thereby much slower, resynchronization of the mirror from the volume is unnecessary.

Two forms or FastResync may be configured on a volume: Persistent FastResync and Non-Persistent FastResync. Persistent FastResync uses disk storage to ensure that FastResync maps survive both system and cluster crashes. Non-Persistent FastResync maps are held in memory. Non-Persistent FastResync maps for volumes in shared disk groups can survive individual system crashes in a cluster but cannot survive cluster crashes. Non-Persistent FastResync maps for volumes in private disk groups do not survive if the system crashes that is accessing them.

When snapshot volumes are reattached to their original volumes, FastResync allows the snapshot data to be quickly refreshed and re-used. If Persistent FastResync is enabled on a volume in a private disk group, such incremental resynchronization can happen even if the host is rebooted.

Persistent FastResync can track the association between volumes and their snapshot volumes after they are moved into different disk groups. When the disk groups are rejoined, this allows the snapshot plexes to be quickly resynchronized. Non-Persistent FastResync cannot be used for this purpose.

Note If you move or split an original volume into a separate disk group from its snapshot volume, and then move or join the volumes into the same disk group, you must use the `vxplex snapback` command with the `-f` option to resynchronize the snapshot plexes. You cannot use `vxassist snapback` for this purpose. This restriction does not apply if you split a snapshot volume into a separate disk group from its original volume, and subsequently return the snapshot volume to the original disk group.

For more information, see "Volume Snapshots" on page 40 and "FastResync" on page 41.

Disk Group Split and Join

Note You may need an additional license to use this feature.

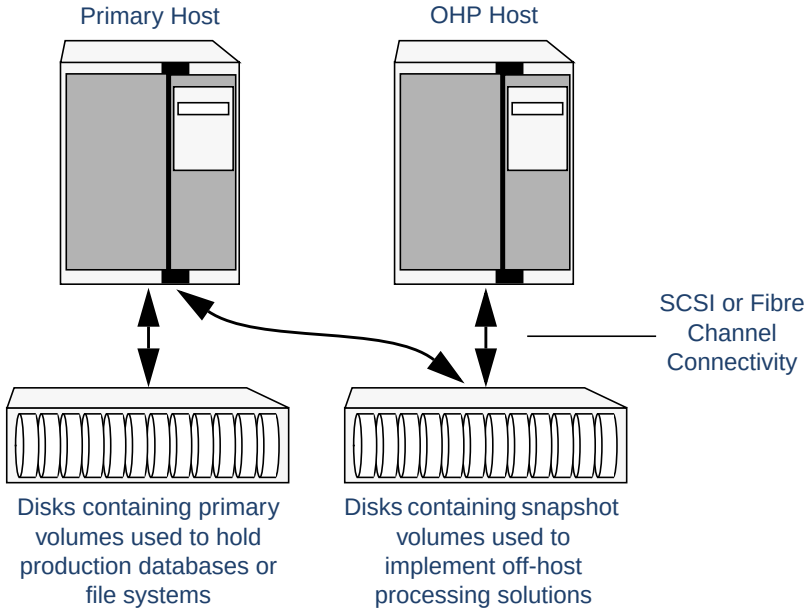
A volume, such as a snapshot volume, can be split off into a separate disk group and deported. It is then ready for importing on another host that is dedicated to off-host processing. This host need not be a member of a cluster but must have access to the disks. At a later stage, the disk group can be deported, re-imported, and joined with the original disk group or with a different disk group.

For more information, see "Reorganizing the Contents of Disk Groups" on page 133.

Implementing Off-Host Processing Solutions

As shown in "Example Implementation of Off-Host Processing" on page 287, by accessing snapshot volumes from a lightly loaded host (shown here as the *OHP host*), CPU- and I/O-intensive operations for online backup and decision support do not degrade the performance of the primary host that is performing the main production activity (such as running a database). Also, if you 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.

Example Implementation of Off-Host Processing



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. Two applications are outlined in the following sections:

- Implementing Online Backup
- Implementing Decision Support

These applications use the Persistent FastResync and disk group move, split and join features of VxVM in conjunction with volume snapshots.

Implementing Online Backup

This section describes a procedure for implementing off-host online backup for a volume in a private disk group. The intention is to present an outline of how to set up a regular backup cycle by combining the Persistent FastResync and disk group split and join features of VxVM. 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, use the following procedure.

1. Use the following command on the primary host to see if the volume is associated with a data change object (DCO) and DCO volume that allow Persistent FastResync to be used with the volume:

   # **vxprint -g *volumedg* -F%hasdcolog *volume***

   This command returns `on` if there is a DCO and DCO volume; otherwise, it returns `off`.

   If the volume is not associated with a DCO object and DCO volume, follow the procedure described in "Adding a DCO and DCO Volume" on page 207.

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 Persistent FastResync is disabled, enable it using the following command on the primary host:

   # **vxvol -g *volumedg* set fastresync=on *volume***

Note If the volume was created before release 3.2 of VxVM, and it has any attached snapshot plexes or it is associated with any snapshot volumes, follow the procedure given in "Enabling Persistent FastResync on Existing Volumes with Associated Snapshots" on page 223.

3. If the volume does not already contain a snapshot plex, create a snapshot mirror for a volume using the following command on the primary host:

   # **vxassist -g *volumedg* [-b] snapstart [nmirror=*N*] *volume***

   The `vxassist snapstart` task creates a write-only mirror, which is attached to and synchronized from the volume to be backed up.

Note By default, VxVM attempts to avoid placing a snapshot mirrors on a disk that already holds any plexes of a data volume. However, this may be impossible if insufficient space is available in the disk group. In this case, VxVM uses any available space on other disks in the disk group. If the snapshot plexes are placed on disks which are used to hold the plexes of other volumes, this may cause problems when you subsequently attempt to move a snapshot volume into another disk group as described in "Considerations for Placing DCO Plexes" on page 138. To override the default storage allocation policy, you can use storage attributes to specify explicitly which disks to use for the snapshot plexes. See "Creating a Volume on Specific Disks" on page 178 for more information.

   If you start `vxassist snapstart` in the background using the `-b` option, you can use the `vxassist snapwait` command to wait for the creation of the mirror to complete as shown here:

   # **vxassist -g *volumedg* snapwait *volume***

   If `vxassist snapstart` is not run in the background, it does not exit until the mirror has been synchronized with the volume. The mirror is then ready to be used as a plex of a snapshot volume. While attached to the original volume, its contents continue to be updated until you take the snapshot.

   Use the `nmirror` attribute to create as many snapshot mirrors as you need for the snapshot volume. For a backup, you should usually only require the default of one.

4. If the volume to be backed up contains database tables in a file system, suspend updates to the volume. The database may have a hot backup mode that allows you to do this by temporarily suspending writes to its tables.

5. On the primary host, make a snapshot volume, *snapvol*, using the following command:

   # **vxassist -g *volumedg* snapshot [nmirrors=*N*] *volume snapvol***

   If required, use the `nmirrors` attribute to specify the number of mirrors in the snapshot volume.

   If a database spans more than one volume, specify all the volumes and their snapshot volumes on the same line, for example:

   ```
   # vxassist -g dbasedg snapshot vol1 snapvol1 vol2 snapvol2 \
     vol3 snapvol3
   ```

6. If you temporarily suspended updates to the volume by a database in step 4, release all the tables from hot backup mode.

7. On the primary host, use the following command to split the snapshot volume into a separate disk group, *snapvoldg*, from the original disk group, *volumedg*:

   # **vxdg split *volumedg snapvoldg snapvol***

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 split. 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
    ```

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 -F vxfs /dev/vx/rdsk/snapvoldg/snapvol
    # mount -F vxfs /dev/vx/dsk/snapvoldg/snapvol mount_point
    ```

    Back up the file system at this point, and then 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. On the primary host, use the following command to rejoin the snapshot volume's disk group with the original volume's disk group:

    # **vxdg join *snapvoldg volumedg***

15. The snapshot volume is initially disabled following the join. Use the following commands on the primary host to recover and restart the snapshot volume:

    # **vxrecover -g *volumedg* -m *snapvol***
    # **vxvol -g *volumedg* start *snapvol***

16. On the primary host, reattach the plexes of the snapshot volume to the original volume, and resynchronize their contents using the following command:

    # **vxassist -g *volumedg* -o allplexes snapback *snapvol***

Repeat steps 4 through 16 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 by combining the Persistent FastResync and disk group split and join features of VxVM. 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, use the following procedure.

1. Use the following command on the primary host to see if the volume is associated with a data change object (DCO) and DCO volume that allow Persistent FastResync to be used with the volume:

    # **vxprint -g *volumedg* -F%hasdcolog *volume***

    This command returns `on` if there is a DCO and DCO volume; otherwise, it returns `off`.

If the volume is not associated with a DCO object and DCO volume, follow the procedure described in "Adding a DCO and DCO Volume" on page 207.

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 Persistent FastResync is disabled, enable it using the following command on the master host:

   # **vxvol -g *volumedg* set fastresync=on *volume***

3. If the volume does not already contain a snapshot plex, create one using the following command on the primary host:

   # **vxassist -g *volumedg* [-b] snapstart [nmirror=*N*] *volume***

   The `vxassist snapstart` task creates a write-only mirror, which is attached to and synchronized from the volume to be backed up.

Note By default, VxVM attempts to avoid placing a snapshot mirrors on a disk that already holds any plexes of a data volume. However, this may be impossible if insufficient space is available in the disk group. In this case, VxVM uses any available space on other disks in the disk group. If the snapshot plexes are placed on disks which are used to hold the plexes of other volumes, this may cause problems when you subsequently attempt to move a snapshot volume into another disk group as described in "Considerations for Placing DCO Plexes" on page 138. To override the default storage allocation policy, you can use storage attributes to specify explicitly which disks to use for the snapshot plexes. See "Creating a Volume on Specific Disks" on page 178 for more information.

If you start `vxassist snapstart` in the background using the `-b` option, you can use the `vxassist snapwait` command to wait for the creation of the mirror to complete as shown here:

vxassist -g *volumedg* snapwait *volume*

If `vxassist snapstart` is not run in the background, it does not exit until the mirror has been synchronized with the volume. The mirror is then ready to be used as a plex of a snapshot volume. While attached to the original volume, its contents continue to be updated until you take the snapshot.

Use the `nmirror` attribute to create as many snapshot mirrors as you need for the snapshot volume. For applications where data redundancy is required for the volume that contains the replica database, specify a number greater than one.

4. 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.

5. On the primary host, suspend updates to the volume that contains the database tables. The database may have a hot backup mode that allows you to do this by temporarily suspending writes to its tables.

6. On the primary host, make a snapshot volume, *snapvol*, using the following command:

   # **vxassist -g *volumedg* snapshot [nmirrors=*N*] *volume snapvol***

   If required, use the `nmirrors` attribute to specify the number of mirrors in the snapshot volume.

   If a database spans more than one volume, specify all the volumes and their snapshot volumes on the same line, for example:

   ```
   # vxassist -g dbasedg snapshot vol1 snapvol1 vol2 snapvol2 \
     vol3 snapvol3
   ```

7. On the primary host, release the tables from hot backup mode.

8. On the primary host, use the following command to split the snapshot volume into a separate disk group, *snapvoldg*, from the original disk group, *volumedg*:

   # **vxdg split *volumedg snapvoldg snapvol***

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 split. 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 -F vxfs /dev/vx/rdsk/snapvoldg/snapvol
    # mount -F 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.

When you no longer need the replica database, or you want to resynchronize its data with the primary database, you can reattach the snapshot plexes with the original volume as described below:

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. On the primary host, use the following command to rejoin the snapshot volume's disk group with the original volume's disk group:

   # **vxdg join *snapvoldg volumedg***

5. The snapshot volume is initially disabled following the join. Use the following commands on the primary host to recover and restart the snapshot volume:

   # **vxrecover -g *volumedg* -m *snapvol***
   # **vxvol -g *volumedg* start *snapvol***

6. On the primary host, reattach the plexes of the snapshot volume to the original volume, and resynchronize their contents using the following command:

   # **vxassist -g *volumedg* -o allplexes snapback *snapvol***

Performance Monitoring and Tuning 12

Introduction

VERITAS Volume Manager (VxVM) can improve overall system performance by optimizing the layout of data storage on the available hardware. This chapter contains guidelines establishing performance priorities, for monitoring performance, and for configuring your system appropriately.

Performance Guidelines

VxVM allows you to optimize data storage performance using the following two 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) that allows continuous access to data in the event of disk failure.

Data Assignment

When deciding where to locate file systems, you, as a system administrator, typically attempt to balance I/O load among available disk drives. The effectiveness of this approach is limited by the difficulty of anticipating future usage patterns, as well as the inability to split file systems across drives. For example, if a single file system receives most disk accesses, moving the file system to another drive also moves the bottleneck to that drive.

VxVM can split volumes across multiple drives. This permits you a finer level of granularity when locating data. After measuring actual access patterns, you can adjust your previous decisions on the placement of file systems. You can reconfigure volumes online without adversely impacting their availability.

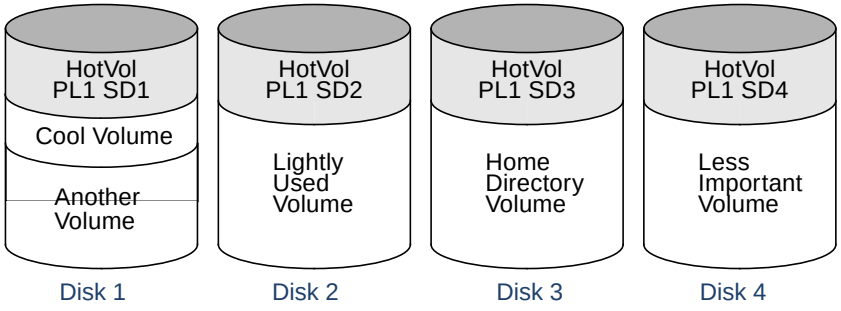
Striping

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.

Having identified 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.

The figure "Use of Striping for Optimal Data Access" shows an example of a single volume (`HotVol`) that has been identified as a data-access bottleneck. This volume is striped across four disks, leaving the remaining space on these disks free for use by less-heavily used volumes.

Use of Striping for Optimal Data Access



Mirroring

Mirroring stores multiple copies of data on a system. When properly applied, mirroring provides continuous availability of data and protection against data loss due to physical media failure. Mirroring improves the chance of data recovery in the event of a system crash or the failure of a disk or other hardware.

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 result in reduced performance.

Combining Mirroring and Striping

Mirroring and striping can be used together to achieve a significant improvement in performance when there are multiple I/O streams.

Striping provides better throughput because parallel I/O streams can operate concurrently on separate devices. Serial access is optimized when I/O exactly fits across all stripe units in one stripe.

Because mirroring is generally used to protect against loss of data due to disk failures, it is often applied to write-intensive workloads which degrades throughput. In such cases, combining mirroring with striping delivers both high availability and increased throughput.

A mirrored-stripe volume may be created by striping half of the available disks to form one striped data plex, and striping the remaining disks to form the other striped data plex in the mirror. This is often the best way to configure a set of disks for optimal performance with reasonable reliability. However, the failure of a single disk in one of the plexes makes the entire plex unavailable.

Alternatively, you can arrange equal numbers of disks into separate mirror volumes, and then 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 22). The failure of a single disk in a mirror does not take the disks in the other mirrors out of use. A striped-mirror layout is preferred over a mirrored-stripe layout for large volumes or large numbers of disks.

RAID-5

RAID-5 offers many of the advantages of combined mirroring and striping, but requires less disk space. RAID-5 read performance is similar to that of striping and RAID-5 parity offers redundancy similar to mirroring. Disadvantages of RAID-5 include relatively slow write performance.

RAID-5 is not usually seen as a way of improving throughput performance except in cases where 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 supports the following read policies on data plexes:

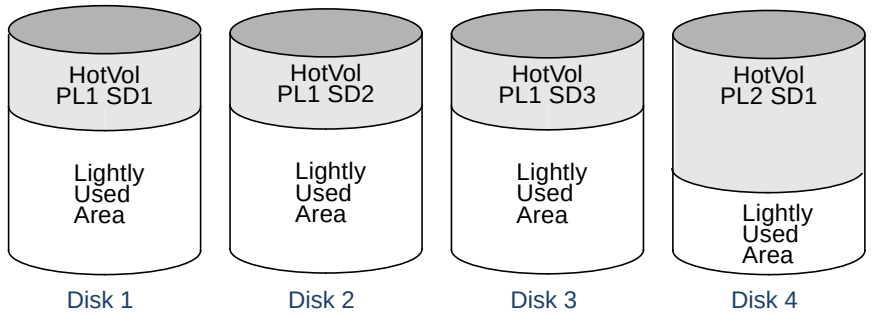
- `round`—a *round-robin* read policy, where all plexes in the volume take turns satisfying read requests to the volume.
- `prefer`—a *preferred-plex* read policy, where the plex with the highest performance usually satisfies read requests. If that plex fails, another plex is accessed.
- `select`—default read policy, where the appropriate read policy for the configuration is selected automatically. For example, `prefer` is selected when there is only one striped plex associated with the volume, and `round` is selected in most other cases.

Note You cannot set the read policy on a RAID-5 data plex. RAID-5 plexes have their own read policy (RAID).

For instructions on how to configure the read policy for a volume's data plexes, see "Changing the Read Policy for Mirrored Volumes" on page 218.

In the configuration example shown in the figure "Use of Mirroring and Striping for Improved Performance," the read policy of the mirrored-stripe volume labeled `Hot Vol` is set to `prefer` for the striped plex `PL1`. This 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.)

Use of Mirroring and Striping for Improved Performance



Note To improve performance for read-intensive workloads, you can attach up to 32 data plexes to the same volume. However, this would usually be 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. 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.

For detailed information about how to use `vxtrace`, refer to 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.

For detailed information about how to use `vxstat`, refer to 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      32.5     24.0
vol  home       2980    5287       6504     10550      37.7    221.1
vol  local     49477   49230     507892    204975      28.5     33.5
vol  rootvol  102906  342664    1085520   1962946      28.1     25.6
vol  src       79174   23603     425472    139302      22.4     30.9
vol  swapvol   22751   32364     182001    258905      25.3    323.2
```

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   disk01     40473   174045     455898   951379     29.5     35.4
dm   disk02     32668    16873     470337   351351     35.2    102.9
dm   disk03     55249    60043     780779   731979     35.3     61.2
dm   disk04     11909    13745     114508   128605     25.0    30.7
```

If the you need to move the volume named `archive` onto another disk, use the following command to identify on which disks it lies:

```
# vxprint -tvh archive
```

The following is a typical display:

```
V  NAME     SETYPE     STATE     STATE    LENGTH  READPOL REFPLEX
PL NAME     VOLUMEK    STATE     STATE    LENGTH  LAYOUT  NCOL/WDTH MODE
SD NAME     PLEX       PLOFFS    DISKOFFSLENGTH  [COL/]OFF FLAGS

v  archive fsgen       ENABLED   ACTIVE  20480   0       SELECT    -
pl archive-01archive   ENABLED   ACTIVE  20480   0       CONCAT    RW
sd disk03-03archive-010          409600  20480   0/0     c1t2d0
```

Note Your system may use a *device name* that differs from the examples. For more information on device names, see “Administering Disks” on page 53.

The subdisks line (beginning `sd`) indicates that the `archive` volume is on disk `disk03`. To move the volume off `disk03`, use the following command:

```
# vxassist move archive !disk03 dest_disk
```

where *dest_disk* is the disk to which you want to move the volume. It is not necessary to specify a *dest_disk*. If you do not specify a *dest_disk*, the volume is moved to an available disk with enough space to contain the volume.

For example, to move the volume from `disk03` to `disk04`, use the following command:

```
# vxassist move archive !disk03 disk04
```

This command indicates that the volume is to be reorganized so that no part remains on `disk03`.

Note The graphical user interface (GUI) provides an easy way to move pieces of volumes between disks and may be preferable to using the command line.

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 `disk02`, `disk03`, and `disk04`, and then remove the original plex `archive-01`:

```
# vxassist mirror archive layout=stripe disk02 disk03 disk04
# vxplex -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.

Caution 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.

Tuning VxVM

This section describes how to adjust the tunable parameters that control the system resources 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. In cases where tuning can be used 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. Any tuning changes must be performed with care, as they may adversely affect overall system performance or may even leave VxVM unusable.

Various mechanisms exist for tuning VxVM. Many parameters can be tuned by editing the file `/kernel/drv/vxio.conf` to override the default values set by the `vxio` driver. Other values can only be tuned using the command line interface to VxVM.

Tuning Guidelines for Large Systems

On smaller systems (with less than a hundred disk drives), tuning is unnecessary and VxVM is capable of adopting 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`(1M) command to select from, and also allow for the creation of larger arrays. 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 with the result 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. This involves re-initializing each disk in the disk group (and can involve reconfiguring everything and restoring from backup).

A general recommendation for users of disk array subsystems is to create a single disk group for each array so the disk group can be physically moved as a unit between systems.

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 configuration copies in a disk group speeds up initial access of the disk group, initial startup of the `vxconfigd` daemon, and transactions 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 minimum 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 (see the `vxedit`(1M) manual page). For example, to configure five configuration copies for the disk group, `bigdg`, use the following command:

```
# vxedit set nconfig=5 bigdg
```

Changing Values of Tunables

Tunables can be modified by editing the file `/kernel/drv/vxio.conf` for most VxVM tunables, or by editing the file `/kernel/drv/vxdmp.conf` for DMP tunables. The system must be shut down and rebooted for the change to take effect.

Caution If you modify `/kernel/drv/vxio.conf` or `/kernel/drv/vxdmp.conf`, make a backup copy of the file.

For example, a single entry has been added to the end of the following `/kernel/drv/vxio.conf` file to change the value of *vol_tunable* to 5000:

```
name="vxio" parent="pseudo" instance=0
vol_tunable=5000;
```

Caution Do not edit the configuration file for the `vxspec` driver, `/kernel/drv/vxspec.conf`.

You can use the `prtconf -vP` command to display the current values of the tunables. All VxVM tunables that you specify in `/kernel/drv/vxio.conf` and `/kernel/drv/vxdmp.conf` are listed in the output under the “System properties” heading for the `vxio` and `vxdmp` drivers. All unchanged tunables are listed with their default values under the “Driver properties” heading. The following sample output shows the new value for *vol_tunable* in hexadecimal:

```
# prtconf -vP
...
vxio, instance #0
     System properties:
        name <vol_tunable> length <4>
           value <0x00001388>
     Driver properties:
        name <voldrl_max_seq_dirty> length <4>
           value <0x00000003>
        name <vol_kmsg_trace_count> length <4>
           value <0x000007d0>
        name <vol_kmsg_resend_period> length <4>
           value <0x00000006>
...
```

For more information, see the `prtconf`(1M) and `driver.conf`(4) manual pages.

Tunable Parameters

The following sections describe specific tunable parameters.

Note Except where noted, the values of tunables are modified by entries in the `/kernel/drv/vxio.conf` file.

dmp_pathswitch_blks_shift

The number of contiguous I/O blocks (expressed as an integer power of 2) that are sent along a DMP path to an Active/Active array before switching to the next available path.

The default value of this parameter is set to 11 so that 2048 blocks (1MB) 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.

The DMP path that is used for a block is calculated as follows:

path = (*block#* >> dmp_pathswitch_blks_shift) % *number of paths*

Note The value of this tunable is changed by an entry in the `/kernel/drv/vxdmp.conf` file.

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 of the checkpoint is 20480 sectors (10MB).

Increasing this size reduces the overhead of checkpoints on recovery operations at the expense of additional recovery following a system failure during a recovery.

vol_default_iodelay

The count in clock ticks for which utilities pause if they have been directed to reduce the frequency of issuing I/O requests, but have not been given a specific delay time. This tunable is used by utilities performing operations such as resynchronizing mirrors or rebuilding RAID-5 columns.

The default for this tunable is 50 ticks.

Increasing this value results in slower recovery operations and consequently lower system impact while recoveries are being performed.

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 `vol_fmr_logsz` value of 4 yields a map contains 32,768 bits, each bit representing one region of 64 blocks.

The larger is 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-pagable 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 the `vol_fmr_logsz` 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 `vol_fmr_logsz` is retained for the life of the volume or until FastResync is turned on for 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 of this tunable is 4 KB. The maximum and minimum permitted values are 1 and 32 KB.

Note The value of this tunable does not have any effect on Persistent FastResync.

vol_max_vol

The maximum number of volumes that can be created on the system. This value can be set to between 1 and the maximum number of minor numbers representable in the system.

The default value for this tunable is 131071.

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 for this tunable is 2048 sectors (1MB).

Note The value of `voliomem_maxpool_sz` must be at least 10 times greater than 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 for this tunable is 32768 bytes (32KB).

vol_maxkiocount

The maximum number of I/O operations that can be performed by VxVM in parallel. Additional I/O requests that attempt to use a volume device are queued until the current activity count drops below this value.

The default value for this tunable is 4096.

Because most process threads can only issue a single I/O request at a time, reaching the limit of active I/O requests in the kernel requires 4096 I/O operations to be performed in parallel. Raising this limit is unlikely to provide much benefit except on the largest of systems.

vol_maxparallelio

The number of I/O operations that the `vxconfigd`(1M) daemon is permitted to request from the kernel in a single `VOL_VOLDIO_READ` per `VOL_VOLDIO_WRITE` `ioctl` call.

The default value for this tunable is 256. It is not desirable to change this value.

vol_maxspecialio

The maximum size of an I/O request that can be issued by an `ioctl` call. Although the `ioctl` request itself can be small, it can request a large I/O request be performed. This tunable limits the size of these I/O requests. If necessary, a request that exceeds this value can be failed, or the request can be broken up and performed synchronously.

The default value for this tunable is 2048 sectors (1MB).

Raising this limit can cause difficulties if the size of an I/O request causes the process to take more memory or kernel virtual mapping space than exists and thus deadlock. The maximum limit for `vol_maxspecialio` is 20% of the smaller of physical memory or kernel virtual memory. It is inadvisable to go over this limit, because deadlock is likely to occur.

If stripes are larger than `vol_maxspecialio`, full stripe I/O requests are broken up, which prevents full-stripe read/writes. This throttles the volume I/O throughput for sequential I/O or larger I/O requests.

This tunable limits the size of an I/O request at a higher level in VxVM than the level of an individual disk. For example, for an 8 by 64KB stripe, a value of 256KB only allows I/O requests that use half the disks in the stripe; thus, it cuts potential throughput in half. If you have more columns or you have used a larger interleave factor, then your relative performance is worse.

This tunable must be set, as a minimum, to the size of your largest stripe (RAID-0 or RAID-5).

vol_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.

volcvm_smartsync

If set to `0`, `volcvm_smartsync` disables SmartSync on shared disk groups. If set to `1`, this parameter enables the use of SmartSync with shared disk groups. See“SmartSync Recovery Accelerator” on page 47 for more information.

voldrl_max_drtregs

The maximum number of dirty regions that can exist for non-sequential DRL on a volume. 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 for this tunable is 2048 sectors (1MB).

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 for this tunable 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 VxVM kernel currently sets the default value for this tunable to 1024 sectors.

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 size for this tunable is 64KB.

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 `voliomem_maxpool_sz`, one for RAID-5 and one for mirrored volumes.

A write request to a RAID-5 volume that is greater than `voliomem_maxpool_sz/10` is broken up and performed in chunks of size `voliomem_maxpool_sz/10`.

A write request to a mirrored volume that is greater than `voliomem_maxpool_sz/2` is broken up and performed in chunks of size `voliomem_maxpool_sz/2`.

The default value for this tunable is 5% of memory up to a maximum of 128MB.

Note The value of `voliomem_maxpool_sz` must be greater than the value of `volraid_minpool_size`, and be at least 10 times greater than the value of `vol_maxio`.

voliot_errbuf_default

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 size for this buffer 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_dflt

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 size of this tunable 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.

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 for this tunable is 4194304 bytes (4MB).

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 size for this buffer is 1048576 bytes (1MB).

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 not in use.

volraid_minpool_sz

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 for this tunable is 2048 sectors (1MB).

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 number of transient reconstruct operations that can be performed in parallel 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.

Commands Summary A

This appendix summarizes the usage and purpose of important commands in VERITAS Volume Manager (VxVM). References are included to longer descriptions in the remainder of this book. For detailed information about an individual command, refer to the appropriate manual page in the 1M section.

Obtaining Information About Objects in VxVM

Command	Description
`vxdisk list [`*diskname*`]`	Lists disks under control of VxVM. See "Displaying Disk Information" on page 92.
`vxdg list [`*diskgroup*`]`	Lists information about disk groups. See "Displaying Disk Group Information" on page 121.
`vxdg -s list`	Lists information about shared disk groups in a cluster. See "Creating Volumes with Exclusive Open Access by a Node" on page 281.
`vxinfo [-g` *diskgroup*`] [`*volume* `...]`	Displays information about the accessibility and usability of volumes. See "Stopping a Volume" on page 203.
`vxprint -hrt [`*object*`]`	Prints single-line information about objects in VxVM. See "Displaying Volume Information" on page 197.
`vxprint -st [`*subdisk*`]`	Displays information about subdisks. See "Displaying Subdisk Information" on page 152.
`vxprint -pt [`*plex*`]`	Displays information about plexes. See "Displaying Plex Information" on page 160.

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 Place a Disk Under Control of VxVM" on page 70.
`vxedit rename olddisk newdisk`	Renames a disk under control of VxVM.See "Renaming a Disk" on page 90.
`vxedit set reserve=on|off diskname`	Sets aside / does not set aside a disk from use in a disk group. See "Reserving Disks" on page 91.
`vxedit 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 250 and "Making a Disk Available for Hot-Relocation Use" on page 251.
`vxedit 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 248 and "Removing a Disk from Use as a Hot-Relocation Spare" on page 249.
`vxdisk offline devicename`	Takes a disk offline. See "Taking a Disk Offline" on page 90.
`vxdg -g diskgroup rmdisk diskname`	Removes a disk from its disk group. See "Removing a Disk with No Subdisks" on page 84.
`vxdisk rm diskname`	Removes a disk from control of VxVM. See "Removing a Disk with No Subdisks" on page 84.

Command	Description
`vxdg init` *diskgroup* `[`*diskname*`=]`*devicename*	Creates a disk group using a pre-initialized disk. See “Creating a Disk Group” on page 122.
`vxdg -s init` *diskgroup* `\` `[`*diskname*`=]`*devicename*	Creates a shared disk group in a cluster using a pre-initialized disk. See “Creating a Shared Disk Group” on page 278.
`vxdg [-n` *newname*`] deport` *diskgroup*	Deports a disk group and optionally renames it. See “Deporting a Disk Group” on page 125.
`vxdg [-n` *newname*`] import` *diskgroup*	Imports a disk group and optionally renames it. See “Importing a Disk Group” on page 126.
`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 279.
`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 138.
`vxdg [-o expand] move` *sourcedg* `\` *targetdg object ...*	Moves objects between disk groups. See “Moving Objects Between Disk Groups” on page 140.
`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 142.
`vxdg [-o expand] join` *sourcedg targetdg*	Joins two disk groups and removes the source disk group. See “Joining Disk Groups” on page 143.
`vxdg -g` *diskgroup* `set \ activation=ew|ro|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 281.
`vxrecover -g` *diskgroup* `-sb`	Starts all volumes in an imported disk group. See “Moving Disk Groups Between Systems” on page 130.
`vxdg destroy` *diskgroup*	Destroys a disk group and releases its disks. See “Destroying a Disk Group” on page 145.

Command	Description
`vxmake sd` *subdisk diskname,offset,length*	Creates a subdisk. See "Creating Subdisks" on page 151.
`vxsd assoc` *plex subdisk* `...`	Associates subdisks with an existing plex. See "Associating Subdisks with Plexes" on page 154.
`vxsd 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 154.
`vxsd aslog` *plex subdisk*	Associates a log subdisk with an exiting plex. See "Associating Subdisks with Plexes" on page 154.
`vxsd mv` *oldsubdisk newsubdisk*	Replaces a subdisk. See "Moving Subdisks" on page 152.
`vxsd -s` *size* `split` *subdisk sd1 sd2*	Splits a subdisk in two. See "Splitting Subdisks" on page 153.
`vxsd join` *sd1 sd2 subdisk*	Joins two subdisks. See "Joining Subdisks" on page 153.
`vxassist [-g` *diskgroup*`] move \` *volume* `!`*olddisk newdisk*	Relocates subdisks in a volume between disks. See "Moving and Unrelocating subdisks using vxassist" on page 254.
`vxunreloc [-g diskgroup]` *original_disk*	Relocates subdisks to their original disks. See "Moving and Unrelocating Subdisks using vxunreloc" on page 254
`vxsd dis` *subdisk*	Dissociates a subdisk from a plex. See "Dissociating Subdisks from Plexes" on page 156.
`vxedit rm` *subdisk*	Removes a subdisk. See "Removing Subdisks" on page 157.
`vxsd -o rm dis` *subdisk*	Dissociates and removes a subdisk from a plex. See "Dissociating Subdisks from Plexes" on page 156.

Command	Description
`vxmake plex` *plex* `\` `sd=`*subdisk1*`[,`*subdisk2*`,...]`	Creates a concatenated plex. See "Creating Plexes" on page 159.
`vxmake 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 160.
`vxplex att` *volume plex*	Attaches a plex to an existing volume. See "Attaching and Associating Plexes" on page 165 and "Reattaching Plexes" on page 167.
`vxplex det` *plex*	Detaches a plex. See "Detaching Plexes" on page 166.
`vxplex off` *plex*	Takes a plex offline for maintenance. See "Taking Plexes Offline" on page 166.
`vxmend on` *plex*	Re-enables a plex for use. See "Reattaching Plexes" on page 167.
`vxplex mv` *oldplex newplex*	Replaces a plex. See "Moving Plexes" on page 168.
`vxplex cp` *volume newplex*	Copies a volume onto a plex. See "Copying Plexes" on page 168.
`vxplex fix clean` *plex*	Sets the state of a plex in an unstartable volume to CLEAN. See "Reattaching Plexes" on page 167.
`vxplex -o rm dis` *plex*	Dissociates and removes a plex from a volume. See "Dissociating and Removing Plexes" on page 169.

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 177.
`vxassist make volume length \ [layout=layout ] [attributes]`	Creates a volume. See "Creating a Volume on Any Disk" on page 177 and "Creating a Volume on Specific Disks" on page 178
`vxassist make volume length \ layout=mirror [nmirror=N] [attributes]`	Creates a mirrored volume. See "Creating a Mirrored Volume" on page 183.
`vxassist 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 281.
`vxassist 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 186 and "Creating a RAID-5 Volume" on page 189.
`vxassist make volume length \ layout=layout mirror=ctlr [attributes]`	Creates a volume with mirrored data plexes on separate controllers. See "Mirroring across Targets, Controllers or Enclosures" on page 188.
`vxmake -Uusage_type vol volume \ [len=length] plex=plex,...`	Creates a volume from existing plexes. See "Creating a Volume Using vxmake" on page 191.
`vxvol start volume`	Initializes and starts a volume for use. See "Initializing and Starting a Volume" on page 194 and "Starting a Volume" on page 204.
`vxvol init zero volume`	Initializes and zeros out a volume for use. See "Initializing and Starting a Volume" on page 194.

Command	Description
`vxassist mirror` *volume* [*attributes*]	Adds a mirror to a volume. See "Adding a Mirror to a Volume" on page 205.
`vxassist remove mirror` *volume* [*attributes*]	Removes a mirror from a volume. See "Removing a Mirror" on page 207.
`vxassist addlog` *volume* [*attributes*]	Adds a log to a volume. See "Adding a DCO and DCO Volume" on page 207, "Adding DRL Logging to a Mirrored Volume" on page 211 and "Adding a RAID-5 Log" on page 212.
`vxassist remove log` *volume* [*attributes*]	Removes a log from a volume. See "Removing a DCO and DCO Volume" on page 210, "Removing a DRL Log" on page 212 and "Removing a RAID-5 Log" on page 213.
`vxvol set fastresync=on|off` *volume*	Turns FastResync on or off for a volume. See "Adding a RAID-5 Log" on page 212.
`vxassist growto` *volume length*	Grows a volume to a specified size. See "Resizing Volumes using vxassist" on page 216.
`vxassist growby` *volume length*	Grows a volume by a specified size. See "Resizing Volumes using vxassist" on page 216.
`vxassist shrinkto` *volume length*	Shrinks a volume to a specified size. See "Resizing Volumes using vxassist" on page 216.
`vxassist shrinkby` *volume length*	Shrinks a volume by a specified size. See "Resizing Volumes using vxassist" on page 216.
`vxresize -b -F xvfs` *volume length* \ *diskname* `...`	Resizes a volume and the underlying VERITAS File System. See "Resizing Volumes using vxresize" on page 215.

Command	Description
`vxassist snapstart` *volume*	Prepares a snapshot mirror of a volume. See "Backing Up Volumes Online Using Snapshots" on page 228.
`vxassist snapshot` *volume snapshot*	Takes a snapshot of a volume. See "Backing Up Volumes Online Using Snapshots" on page 228.
`vxassist snapback` *volume snapshot*	Merges a snapshot with its original volume. See "Merging a Snapshot Volume (snapback)" on page 232.
`vxassist snapclear` *snapshot*	Makes the snapshot volume independent. See "Dissociating a Snapshot Volume (snapclear)" on page 233.
`vxassist [-g` *diskgroup*`] relayout` *volume* `\ [layout=`*layout*`] [`*relayout_options*`]`	Performs online relayout of a volume. See "Performing Online Relayout" on page 235.
`vxassist 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 235.
`vxrelayout -o bg reverse` *volume*	Reverses the direction of a paused volume relayout. See "Controlling the Progress of a Relayout" on page 237.
`vxassist 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 238.
`vxassist convert` *volume* `\ layout=mirror-stripe`	Converts a striped-mirror volume to a mirrored-stripe volume. See "Converting Between Layered and Non-Layered Volumes" on page 238.
`vxvol stop` *volume*	Stops a volume. See "Stopping a Volume" on page 203.
`vxassist remove volume` *volume*	Removes a volume. See "Removing a Volume" on page 219.

Monitoring and Controlling Tasks

Command	Description
`vx`*`command`* `-t` *`tasktag`* `[`*`options`*`] [`*`arguments`*`]`	Specifies a task tag to a command. See “Specifying Task Tags” on page 201.
`vxtask [-h] list`	Lists tasks running on a system. See “`vxtask` Usage” on page 203.
`vxtask monitor` *`task`*	Monitors the progress of a task. See “`vxtask` Usage” on page 203.
`vxtask pause` *`task`*	Suspends operation of a task. See “`vxtask` Usage” on page 203.
`vxtask -p list`	Lists all paused tasks. See “Moving Disk Groups Between Systems” on page 130.
`vxtask resume` *`task`*	Resumes a paused task. See “`vxtask` Usage” on page 203.
`vxtask abort` *`task`*	Cancels a task and attempts to reverse its effects. See “`vxtask` Usage” on page 203.

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. Also see path, primary path, and secondary path.

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.

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). Also see private 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).

data change object (DCO)

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.

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 `c0t0d0s2`. The `c#t#d#s#` syntax identifies the controller, target address, disk, and slice (or partition). 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 procedure by which the VxVM monitors and logs modifications to a plex. A bitmap of changed regions is kept in 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. Also see disk enclosure and JBOD.

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. For example, if a disk is represented by the device name `/dev/sbus@1f,0/QLGC,isp@2,10000/sd@8,0:c` then the path component `QLGC,isp@2,10000` represents the disk controller that is connected to the host for disk `sd@8,0:c`.

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 root disk group (`rootdg`) is a special private disk group that always exists.

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/vfstab` 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. Also see Persistent FastResync and Non-Persistent FastResync.

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 *hostid* 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

The common name for an unintelligent disk array which may, or may not, support the hot-swapping of disks. The name is derived from “just a bunch 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. Also see active/passive disk arrays, primary path and secondary path.

pathgroup

In 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. Also see data change object (DCO).

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. Also see path and secondary path.

private disk group

A disk group in which the disks are accessed by only one specific host in a cluster. Also see shared disk group.

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

A Redundant Array of Independent Disks (RAID) is 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 disk group

A special private disk group that always exists on the system. The root disk group is named `rootdg`. Even though `rootdg` is the default disk group, it does not necessarily contain the root disk unless this is 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. Also see path and primary path.

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). Also see private 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.

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).

Storage Area Network (SAN)

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 32 sectors (16K). A *stripe unit size* has also historically been referred to as a *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.

volboot file

A small file that is used to locate copies of the `rootdg` (root 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. A `rootdg` that contains only simple disks requires entries for at least some of those disks in the `volboot` file so that vxconfigd can configure the disk group. The `volboot` file is stored in a system-dependent location.

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 or simply disks.

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`.

volume event log

The device interface (`/dev/vx/event`) through which volume driver events are reported to utilities.

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

S