VERITAS™

VERITAS Volume Manager 4.1

Administrator's Guide

Solaris x64 Platform Edition

N13103F

November 2005

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

Contents

Preface

The *VERITAS Volume Manager Administrator's Guide* provides information on how to use VERITAS Volume Manager (VxVM) from the command line.

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

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 VERITAS Volume Manager. 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 VERITAS Volume Manager.

This guide assumes that you have a:

- Working knowledge of the Solaris™ operating system (OS).
- Basic understanding of Solaris system administration .
- Basic understanding of storage management.

Details on how to use the VERITAS Enterprise Administrator (VEA) can be found in the *VERITAS Enterprise Administrator Getting Started* manual and VEA online help. 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 301 for a listing of the commonly used commands in VxVM together with references to their descriptions.

Note Some features such as Intelligent Storage Provisioning (ISP) and Cross-platform Data Sharing (CDS) software have their own dedicated administration guides.

Most VERITAS Volume Manager commands require superuser or other appropriate privileges.

How This Guide Is Organized

This guide is organized as follows:

Refer to the *Release Notes* for information about the other documentation that is provided with this product.

Conventions

Convention	Usage	Example
`monospace`	Used for path names, commands, output, directory and file names, functions, and parameters.	Read tunables from the `/etc/vx/tunefstab` file. See the `ls`(1) manual page for more information.
`monospace` (**`bold`**)	Indicates user input.	# **`ls pubs`** C:\> **`dir pubs`**
italic	Identifies book titles, new terms, emphasized text, and variables replaced with a name or value.	See the *User's Guide* for details. The variable *system_name* indicates the system on which to enter the command.
bold	Depicts GUI objects, such as fields, list boxes, menu selections, etc. Also depicts GUI commands.	Enter your password in the **Password** field. Press **Return.**
blue text	Indicates hypertext links.	See "Getting Help" on page xviii.
#	Unix superuser prompt (all shells).	# **`cp /pubs/4.0/user_book /release_mgnt/4.0/archive`**

C:\>	Windows user prompt.	C:\> **`copy \pubs\4.0\user_book`**
		`c:\release_mgnt\4.0\archive`

Getting Help

For technical assistance, visit http://support.veritas.com and select phone or email support. This site also provides access to resources such as TechNotes, product alerts, software downloads, hardware compatibility lists, and the VERITAS customer email notification service. Use the Knowledge Base Search feature to access additional product information, including current and past releases of product documentation.

Diagnostic tools are also available to assist in troubleshooting problems associated with the product. These tools are available on disc or can be downloaded from the VERITAS FTP site. See the `README.VRTSspt` file in the `/support` directory for details.

For license information, software updates and sales contacts, visit https://my.veritas.com/productcenter/ContactVeritas.jsp. For information on purchasing product documentation, visit http://webstore.veritas.com.

Documentation Feedback

Your feedback on product documentation is important to us. Send suggestions for improvements and reports on errors or omissions to foundation_docs@veritas.com. Include the title and part number of the document (located in the lower left corner of the title page), and chapter and section titles of the text on which you are reporting. Our goal is to ensure customer satisfaction by providing effective, quality documentation. For assistance with topics other than documentation, visit http://support.veritas.com.

Understanding VERITAS Volume Manager 1

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)
- SmartSync Recovery Accelerator
- Volume Snapshots
- Hot-Relocation

Further information on administering VERITAS Volume Manager may be found in the following documents:

- *VERITAS Storage Foundation Cross-Platform Data Sharing Administrator's Guide*

  Provides more information on using the Cross-platform Data Sharing (CDS) feature of VERITAS Volume Manager, which allows you to move VxVM disks and objects between machines that are running under different operating systems.

  Note CDS requires a VERITAS Storage Foundation license.

- *VERITAS Storage Foundation Intelligent Storage Provisioning Administrator's Guide*

  Describes the command-line interface to the VERITAS Intelligent Storage Provisioning (ISP) feature, which uses a rule-based engine to create VxVM objects and make optimal usage of the available storage.

- *VERITAS Volume Manager Troubleshooting Guide*

  Describes recovery from hardware failure, disk group configuration and restoration, command and transaction logging, and common error messages together with suggested solutions.

- *VERITAS Enterprise Administrator Getting Started*

  Describes how to use the VERITAS Enterprise Administrator — the graphical user interface to VERITAS Volume Manager. More detailed information is available in the VEA online help.

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 where 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.

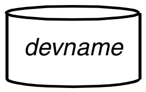
Physical Objects—Physical Disks

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” on page 4, 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

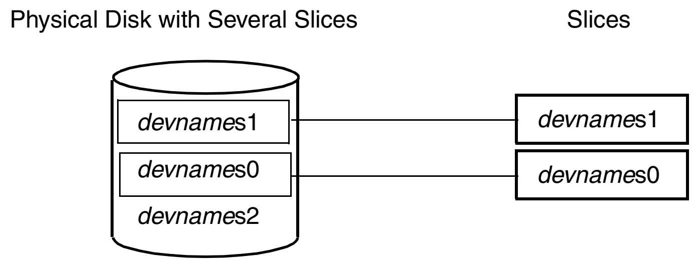


VxVM writes identification information on physical disks under VxVM control (VM disks). VxVM disks can be identified even after physical disk disconnection or system outages. VxVM can then re-form disk groups and logical objects to provide failure detection and to speed system recovery.

Partitions

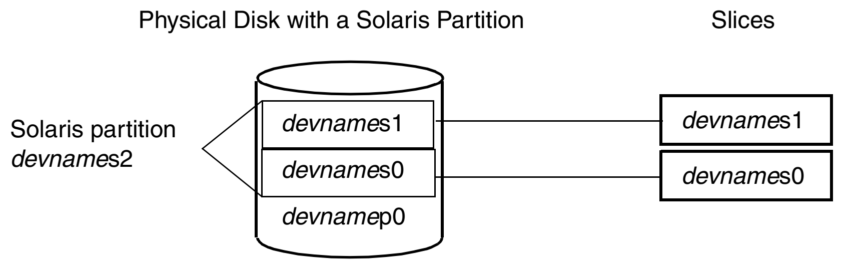
On a SPARC system, a physical disk can be divided into one or more *partitions*, also known as *slices*. The *slice number* is added at the end of the devname, and is denoted by `s#`. Note that the s2 device refers to an entire physical disk for non-EFI disks. See the partition shown in "Partition Example for an x86 System."

Partition Example for a SPARC System



On an x86 system, a physical disk can be divided into one or more *partitions*, and a Solaris partition may be also be divided into *slices*. The *slice number* is added at the end of the device name, and is denoted by `s#`. Unlike the SPARC platform, the special partition `p0` refers to the entire physical disk (for example, `c0t0d0p0`), and the `s2` device indicates only the Solaris partition on the disk (for example, `c0t0d0s2`). An additional device, `s8`, is used to denote the boot slice. See the partition shown in "Partition Example for an x86 System."

Partition Example for an x86 System



Disk Arrays

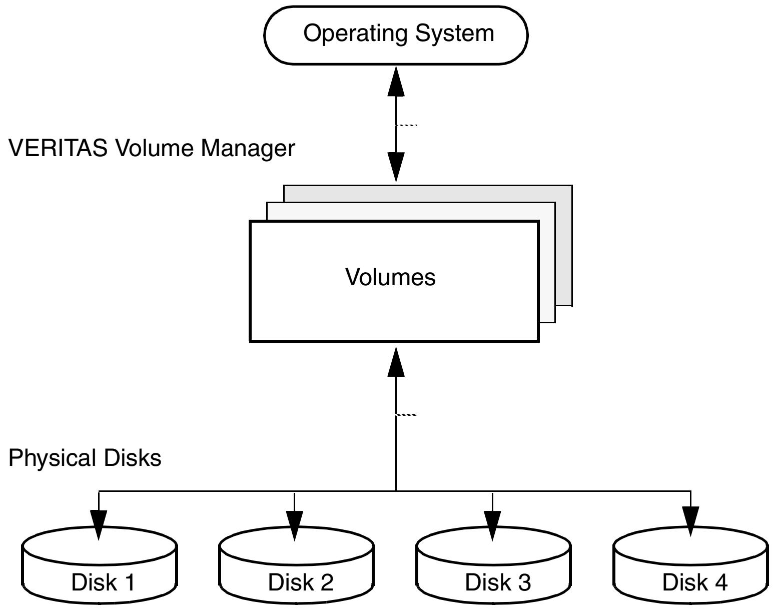
Performing I/O to disks is a relatively slow process because disks are physical devices that require time to move the heads to the correct position on the disk before reading or writing. If all of the read or write operations are done to individual disks, one at a time, the read-write time can become 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" on page 6 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 service enables you to add support dynamically for new disk arrays. This operation, which uses a facility called the Device Discovery Layer (DDL), is achieved without the need for a reboot.

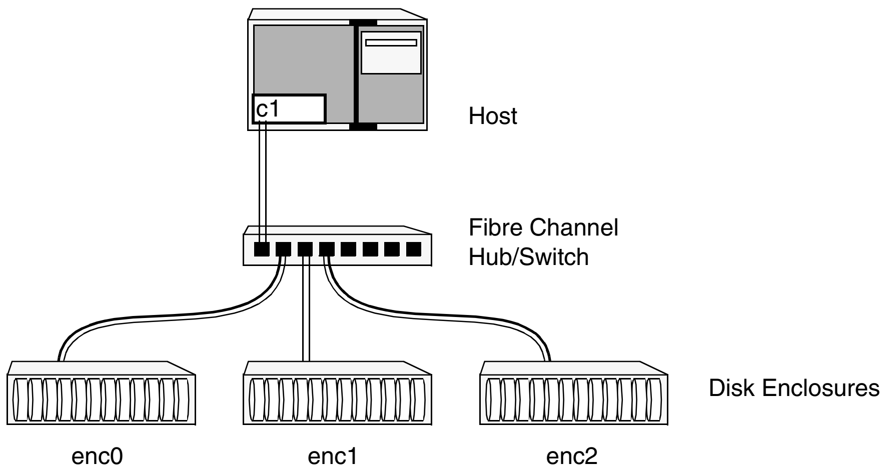
This means that you can dynamically add a new disk array to a host, and run a command which scans the operating system's device tree for all the attached disk devices, and reconfigures DMP with the new device database. For more information, see "Administering the Device Discovery Layer" on page 57.

Enclosure-Based Naming

Enclosure-based naming provides an alternative to the disk device naming described in "Physical Objects—Physical Disks" on page 4. 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.

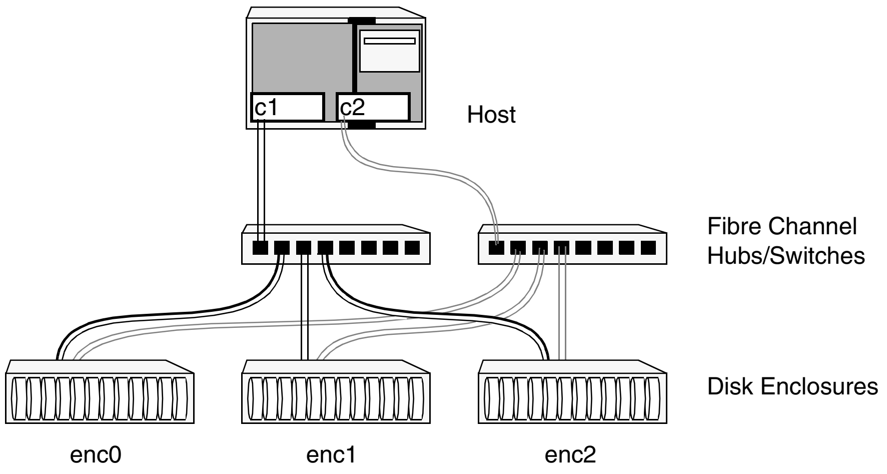
Another important benefit of enclosure-based naming is that it enables VxVM to avoid placing redundant copies of data in the same enclosure. This is a good thing to avoid as each enclosure can be considered to be a separate fault domain. For example, if a mirrored volume were configured only on the disks 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 119 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” on page 9. 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 187.

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



See "Disk Device Naming in VxVM" on page 50 and "Changing the Disk-Naming Scheme" on page 63 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:

- Disk Groups
- VM Disks
- 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 196 and the `vxprint`(1M) manual page.

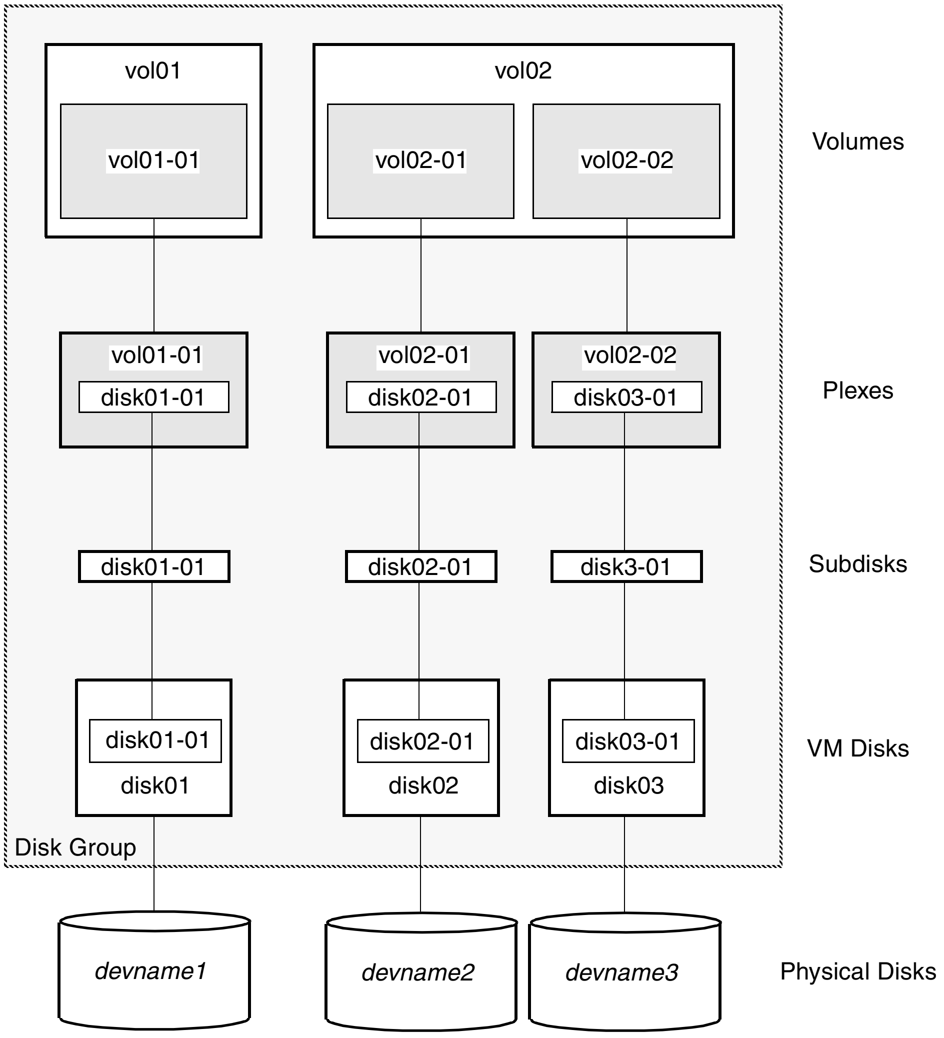
Combining Virtual Objects in VxVM

VxVM virtual objects are combined to build volumes. The virtual objects contained in volumes are VM disks, disk groups, subdisks, and plexes. 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" on page 11, shows the connections between VERITAS Volume Manager virtual objects and how they relate to physical disks. The disk group contains three VM disks which are used to create two volumes. Volume `vol01` is simple and has a single plex. Volume `vol02` is a mirrored volume with two plexes.

Connection Between Objects in VxVM



The various types of virtual objects (disk groups, VM disks, subdisks, plexes and volumes) are described in the following sections.

Disk Groups

A *disk group* is a collection of disks that share a common configuration, and which are managed by VxVM (see "VM Disks" on page 12). A disk group configuration is a set of records with detailed information about related VxVM objects, their attributes, and their connections. A disk group name can be up to 31 characters long.

In releases prior to VxVM 4.0, the default disk group was `rootdg` (the *root disk group*). For VxVM to function, the `rootdg` disk group had to exist and it had to contain at least one disk. This requirement no longer exists, and VxVM can work without any disk groups configured (although you must set up at least one disk group before you can create any volumes of otherVxVM objects). For more information about changes to disk group configuration, see "Creating and Administering Disk Groups" on page 123.

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

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

VM Disks

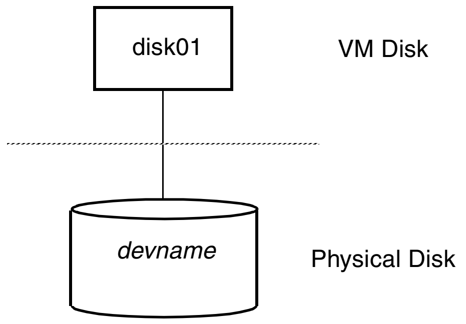
When you place a physical disk under VxVM control, a VM disk is assigned to the physical disk. A VM disk is under VxVM control and is usually in a disk group. Each VM disk corresponds to at least one physical disk or disk partition. VxVM allocates storage from a contiguous area of VxVM disk space.

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

Each VM disk has a unique *disk media name* (a virtual disk name). You can either define a disk name of up to 31 characters, or allow VxVM to assign a default name that takes the form `diskgroup##`, where *diskgroup* is the name of the disk group to which the disk belongs (see "Disk Groups" on page 12).

"VM Disk Example" on page 13 shows a VM disk with a media name of `disk01` that is assigned to the physical disk *devname*.

VM Disk Example



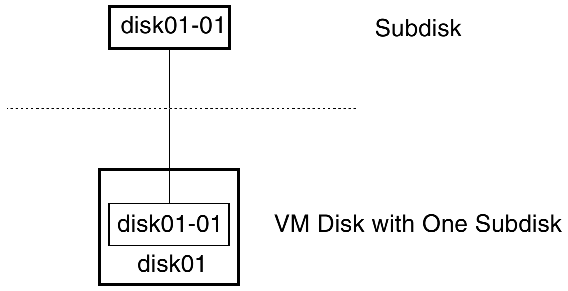
Subdisks

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

The default name for a VM disk is *`diskgroup`*`##` and the default name for a subdisk is *`diskgroup`*`##-##`, where *diskgroup* is the name of the disk group to which the disk belongs (see "Disk Groups" on page 12).

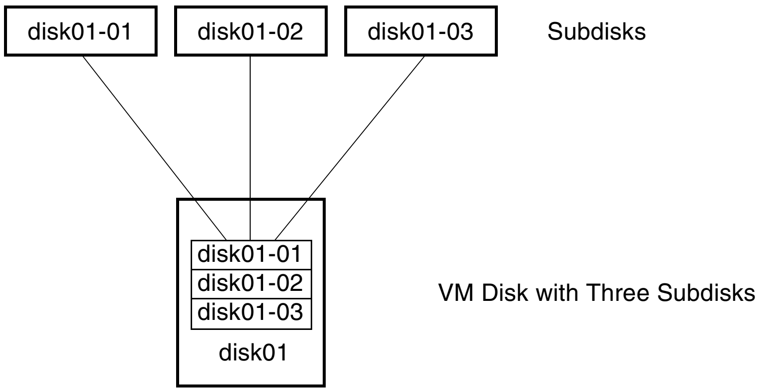
In the figure, "Subdisk Example" on page 13, `disk01-01` is the name of the first subdisk on the VM disk named `disk01`.

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" on page 14 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



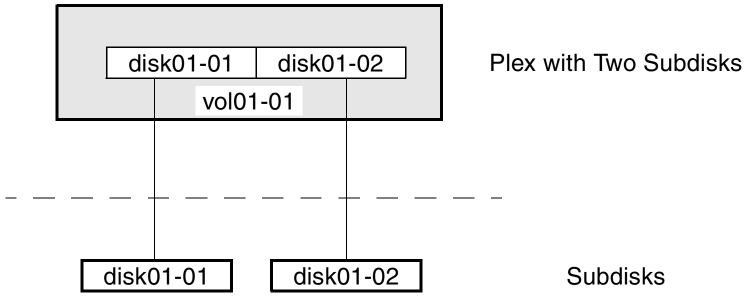
Any VM disk space that is not part of a subdisk is free space. You can use free space to create new subdisks.

VERITAS Volume Manager supports the concept of layered volumes in which subdisks can contain volumes. For more information, see "Layered Volumes" on page 35.

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 18.

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:

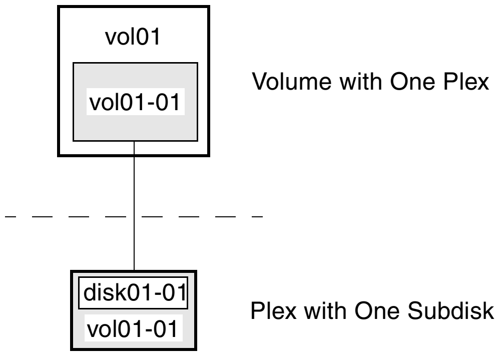
- Its name can contain up to 31 characters.
- It can consist of up to 32 plexes, each of which contains one or more subdisks.
- It must have at least one associated plex that has a complete copy of the data in the volume with at least one associated subdisk.
- All subdisks within a volume must belong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

Note You can use the VERITAS Intelligent Storage Provisioning (ISP) feature to create and administer application volumes. These volumes are very similar to the traditional VxVM volumes that are described in this chapter. However, there are significant differences between the functionality of the two types of volume that prevent them from being used interchangeably. Refer to the *VERITAS Storage Foundation Intelligent Storage Provisioning Administrator's Guide* for more information about creating and administering ISP application volumes.

In "Example of a Volume with One Plex," volume `vol01` has the following characteristics:

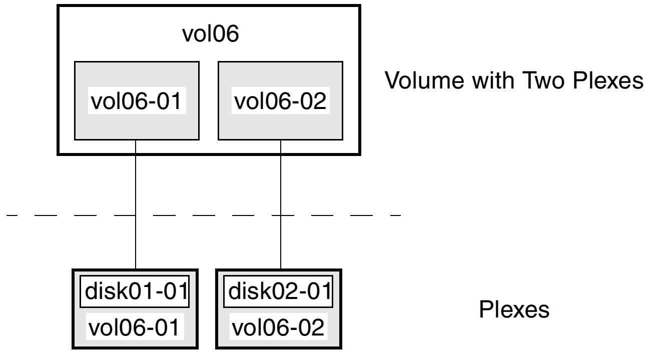
- It contains one plex named `vol01-01`.
- The plex contains one subdisk named `disk01-01`.
- The subdisk `disk01-01` is allocated from VM disk `disk01`.

Example of a Volume with One Plex



In "Example of a Volume with Two Plexes," a volume, `vol06`, with two data plexes is *mirrored*. Each plex of the mirror contains a complete copy of the volume data.

Example of a Volume with Two Plexes



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`).

For more information, see "Mirroring (RAID-1)" on page 26.

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 35 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

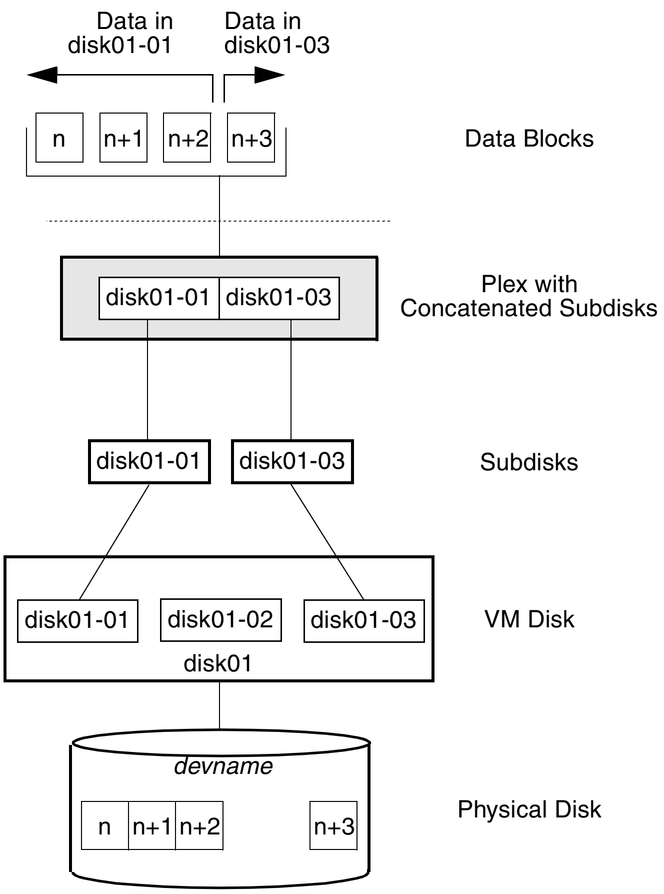
Concatenation maps data in a linear manner onto one or more subdisks in a plex. To access all of the data in a concatenated plex sequentially, data is first accessed in the first subdisk from beginning to end. Data is then accessed in the remaining subdisks sequentially from beginning to end, until the 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 the concatenation of two subdisks from the same VM disk. The blocks `n`, `n+1`, `n+2` and `n+3` (numbered relative to the start of the plex) are contiguous on the plex, but actually come from two distinct subdisks on the same physical disk.

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

Example of Concatenation

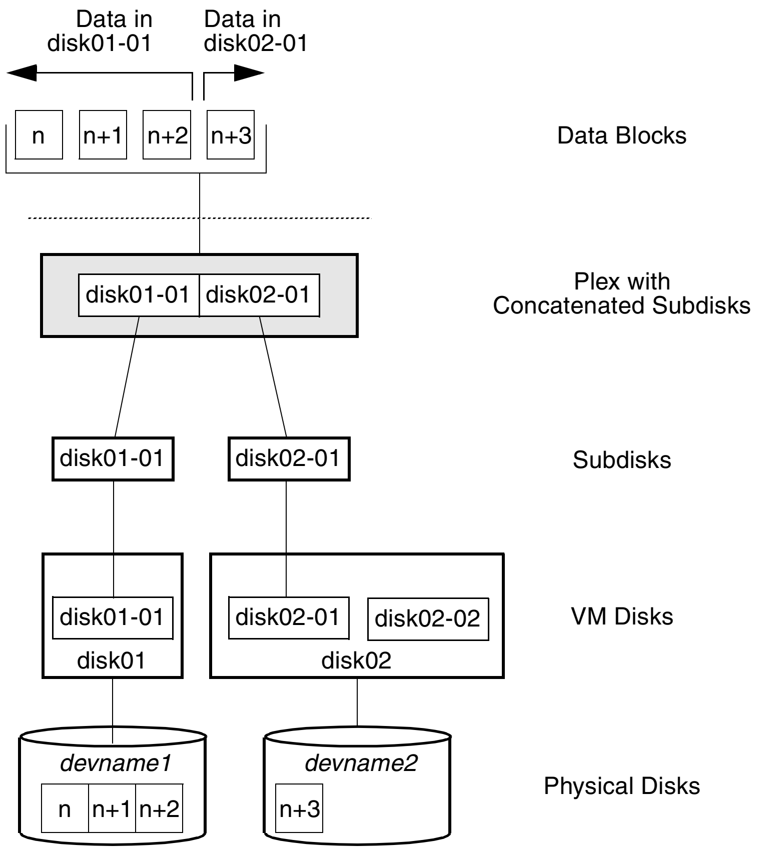


You can use concatenation with multiple subdisks when there is insufficient contiguous space for the plex on any one disk. This form of concatenation can be used for load balancing between disks, and for head movement optimization on a particular disk.

The figure, "Example of Spanning" on page 21, shows data spread over two subdisks in a spanned plex. The blocks `n`, `n+1`, `n+2` and `n+3` (numbered relative to the start of the plex) are contiguous on the plex, but actually come from two distinct subdisks from two distinct physical disks.

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

Example of Spanning



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.

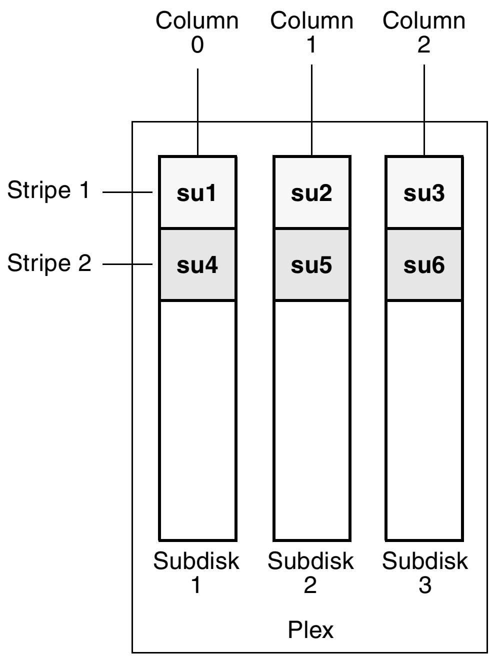
Caution Striping a volume, or splitting a volume across multiple disks, increases the chance that a disk failure will result in failure of that volume.

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" on page 23.

Striping Across Three Columns



SU = Stripe Unit

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

Viewed in sequence, the first stripe consists of:

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

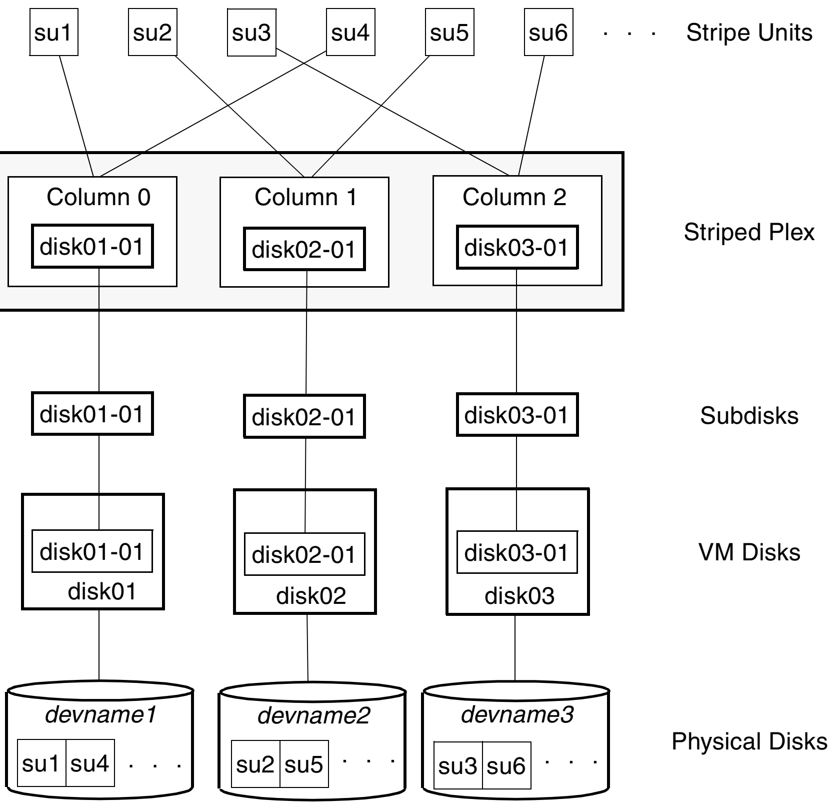
The second stripe consists of:

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

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

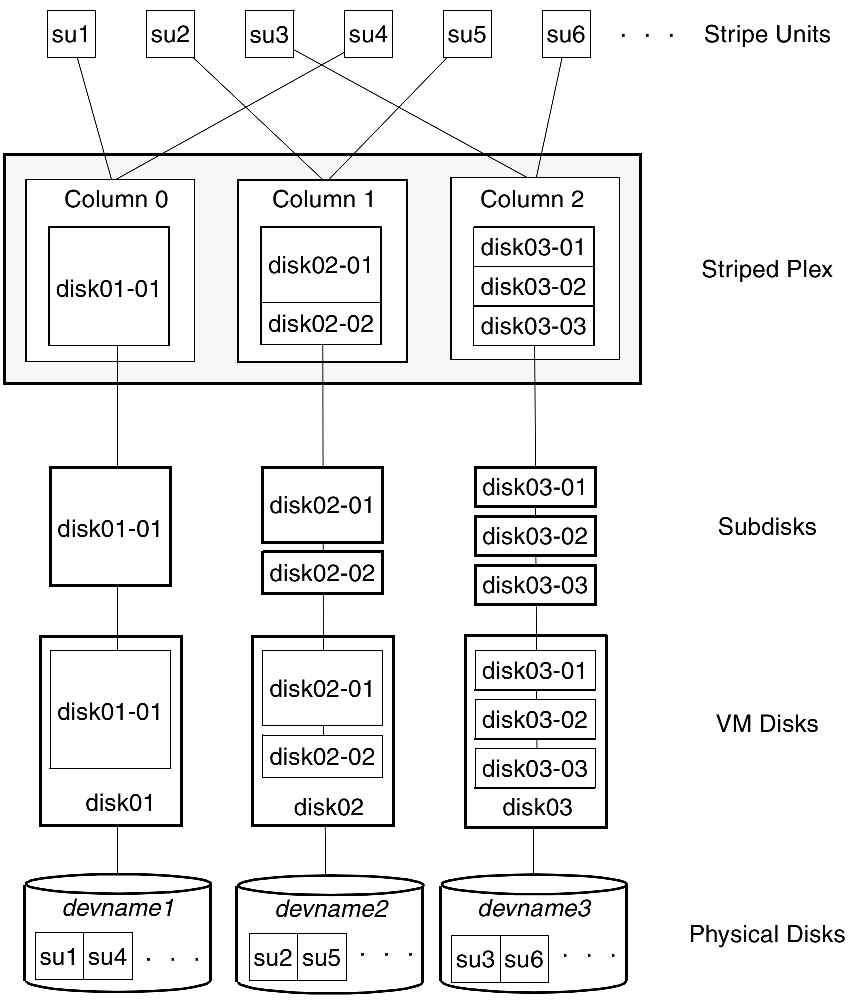
"Example of a Striped Plex with One Subdisk per Column" on page 24 shows a striped plex with three equal sized, single-subdisk columns. There is one column per physical disk. This example shows three subdisks that occupy all of the space on the VM disks. It is also possible for each subdisk in a striped 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" on page 25 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 185 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.

Disk duplexing, in which each mirror exists on a separate controller, is also supported. See "Mirroring across Targets, Controllers or Enclosures" on page 187 for details.

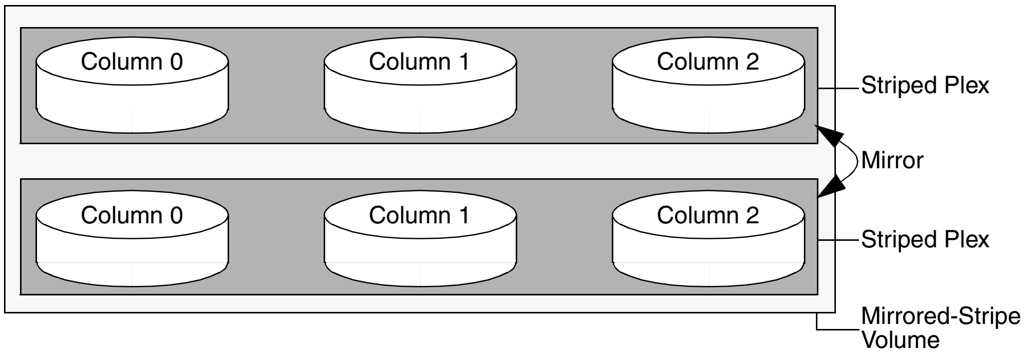
Striping Plus Mirroring (Mirrored-Stripe or RAID-0+1)

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" on page 27 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 186 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)

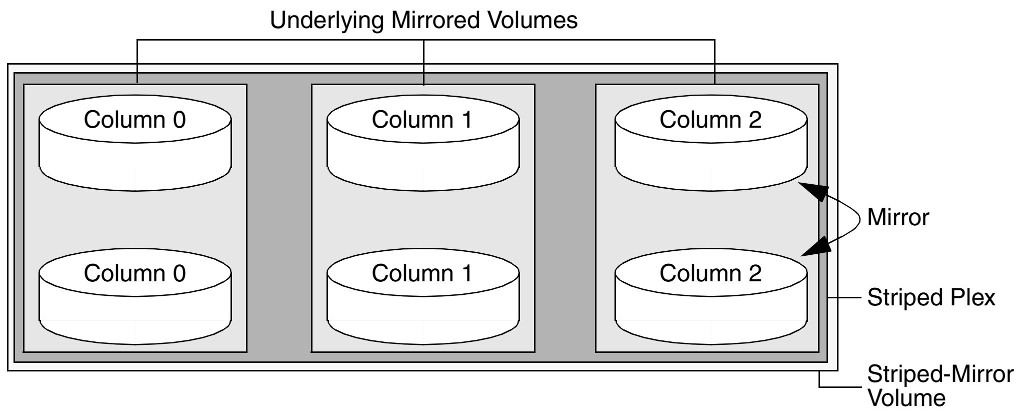
VxVM supports the combination of striping above mirroring. This combined layout is called a *striped-mirror* layout. Putting mirroring below striping mirrors each column of the stripe. If there are multiple subdisks per column, each subdisk can be mirrored individually instead of each column.

Note A striped-mirror volume is an example of a layered volume. See "Layered Volumes" on page 35 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" on page 28 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.

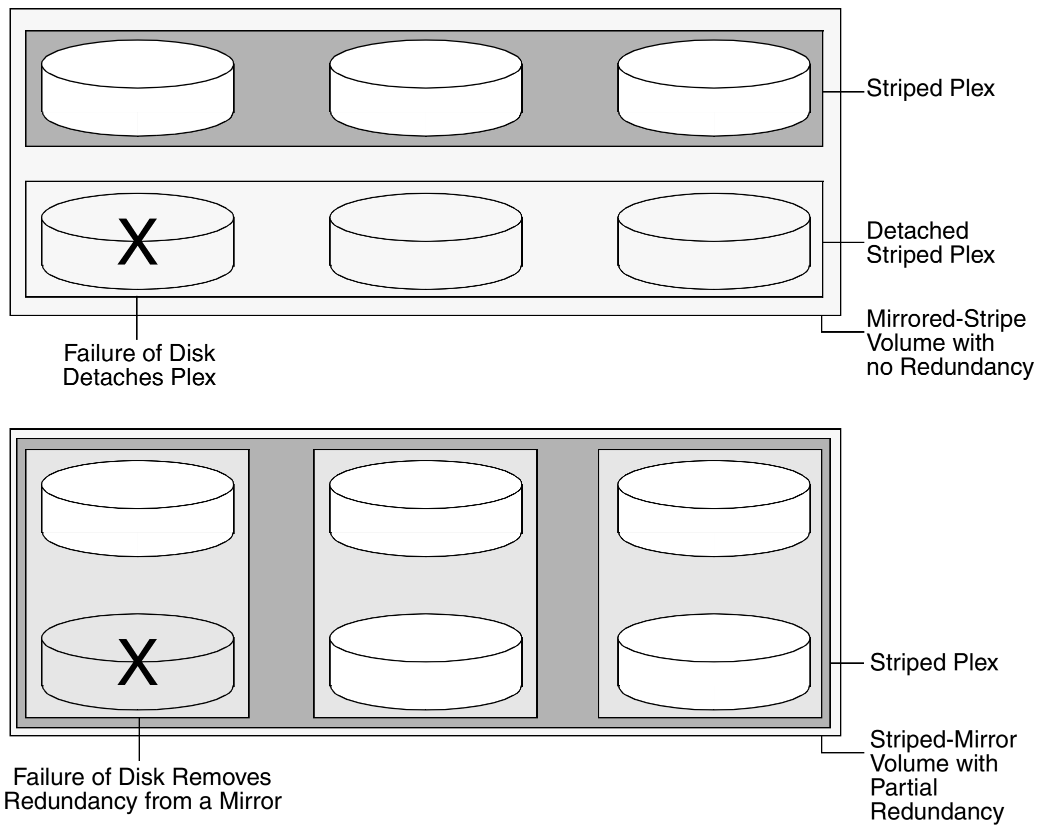
Striped-Mirror Volume Laid out on Six Disks



See "Creating a Striped-Mirror Volume" on page 186 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" on page 29, 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


Striped Plex
X
Detached Striped Plex
Mirrored-Stripe Volume with no Redundancy
Failure of Disk Detaches Plex
X
Striped Plex
Striped-Mirror Volume with Partial Redundancy
Failure of Disk Removes Redundancy from a Mirror

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

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

RAID-5 (Striping with Parity)

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

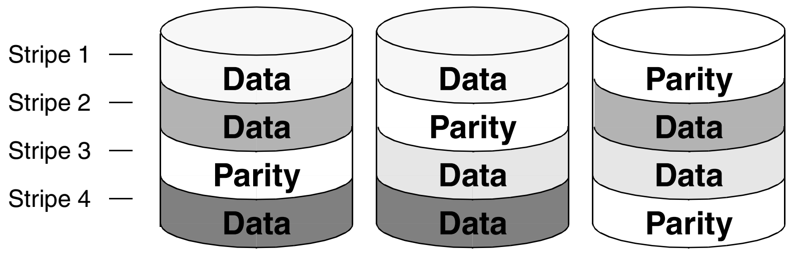
Although both mirroring (RAID-1) and RAID-5 provide redundancy of data, they use different methods. Mirroring provides data redundancy by maintaining multiple complete copies of the data in a volume. Data being written to a mirrored volume is reflected in all copies. If a portion of a mirrored volume 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 Mode" on page 30, 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 Mode

l



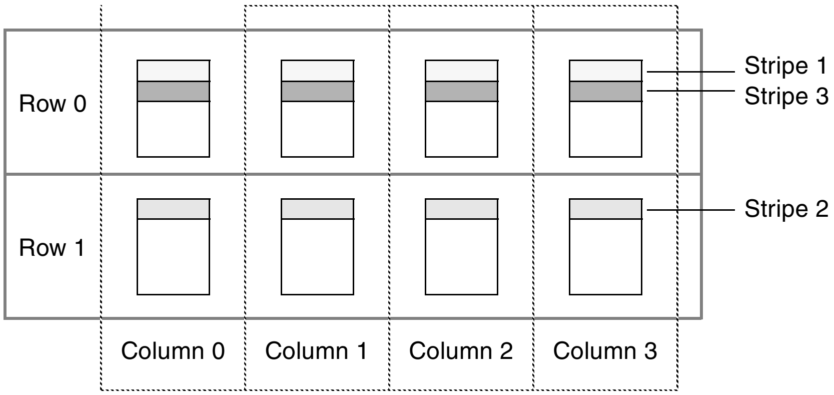
RAID-5 volumes can additionally perform logging to minimize recovery time. RAID-5 volumes use RAID-5 logs to keep a copy of the data and parity currently being written. RAID-5 logging is optional and can be created along with RAID-5 volumes or added later.

The implementation of RAID-5 in VxVM is described in "VERITAS Volume Manager RAID-5 Arrays" on page 31.

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



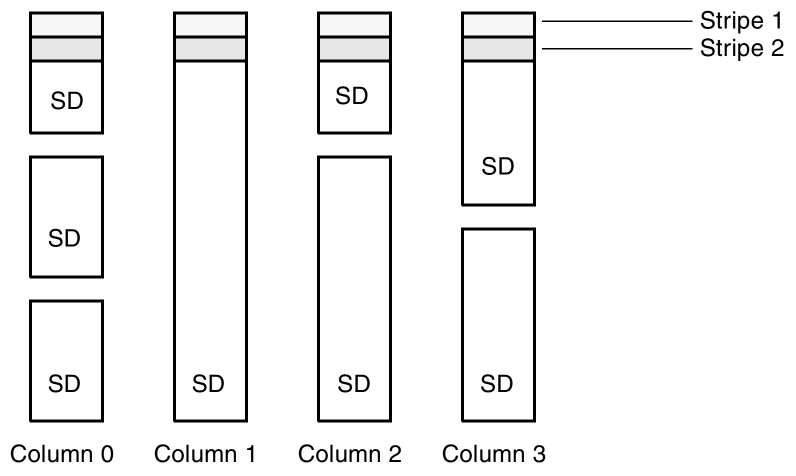
This traditional array structure supports growth by adding more rows per column. Striping is accomplished by applying the first stripe across the disks in Row 0, then the second stripe across the disks in Row 1, then the third stripe across the Row 0 disks, and so on. This type of array requires all 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 Manager RAID-5 Array" on page 32). 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 22 for further information about stripe units.

VERITAS Volume Manager RAID-5 Array



SD = Subdisk

Note Mirroring of RAID-5 volumes is not supported.

See "Creating a RAID-5 Volume" on page 188 for information on how to create a RAID-5 volume.

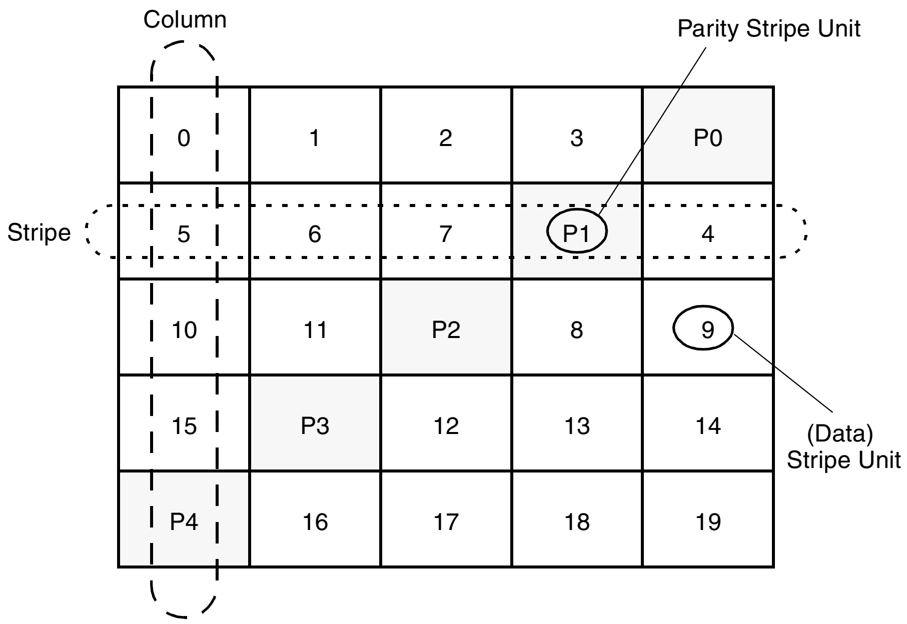
Left-Symmetric Layout

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.

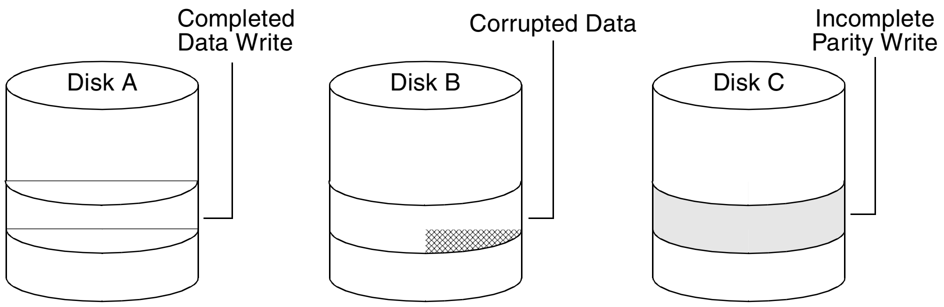
RAID-5 Logging

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.

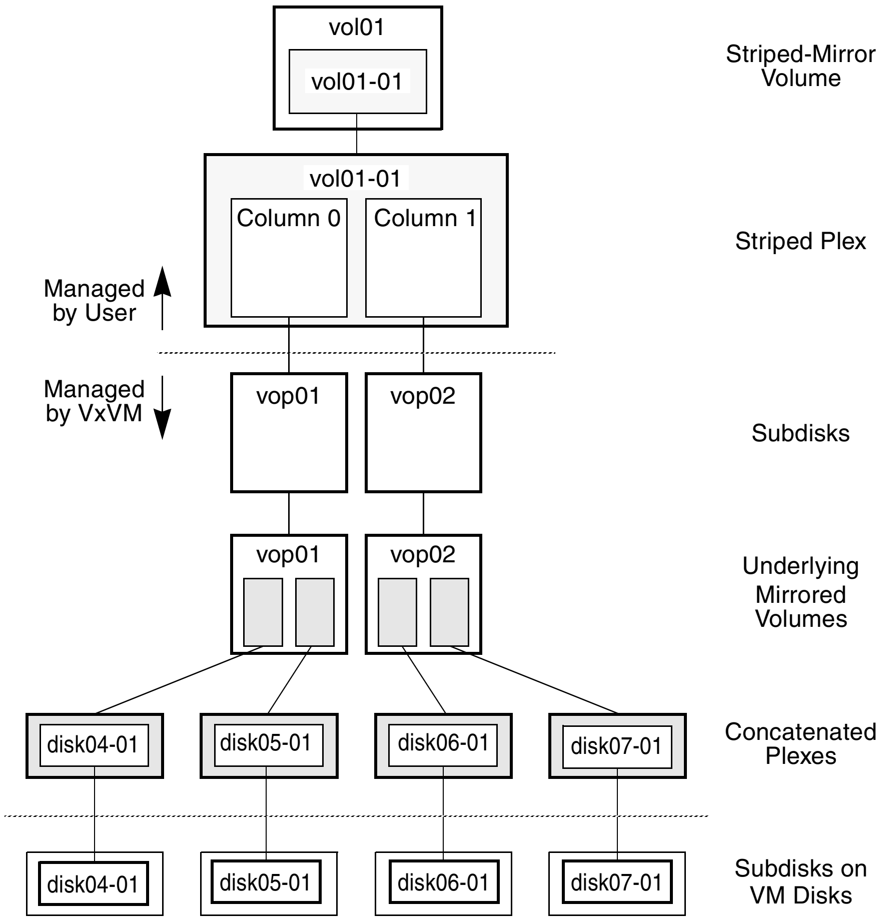
Logs are associated with a RAID-5 volume by being attached as log plexes. More than one log plex can exist for each RAID-5 volume, in which case the log areas are mirrored.

See “Adding a RAID-5 Log” on page 210 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.

Example of a Striped-Mirror Layered Volume



The figure, "Example of a Striped-Mirror Layered Volume" on page 35, 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).

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 186, and the `vxassist`(1M) manual page.)
- Creating concatenated-mirrors. (See "Creating a Concatenated-Mirror Volume" on page 183, and the `vxassist`(1M) manual page.)
- Online Relayout. (See "Online Relayout" on page 37, and the `vxrelayout`(1M) and `vxassist`(1M) manual pages.)
- RAID-5 subdisk moves. (See the `vxsd`(1M) manual page.)
- Snapshots. (See "Administering Volume Snapshots" on page 227, and the `vxsnap`(1M) and `vxassist`(1M) manual pages.)

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 219 for details of how to perform online relayout of volumes in VxVM. Also see "Converting Between Layered and Non-Layered Volumes" on page 225 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:

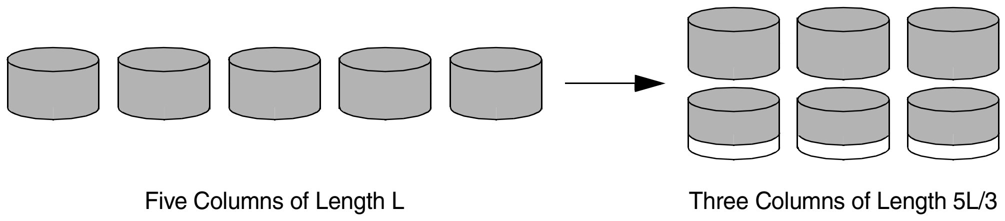
```
tmpsize too small to perform this relayout (nblks minimum required)
```

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.

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

Additional permanent disk space may be required for the destination volumes, depending on the type of relayout that you are performing. This may happen, for example, if you change the number of columns in a striped volume. The figure, "Example of Decreasing the Number of Columns in a Volume" on page 38, 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.

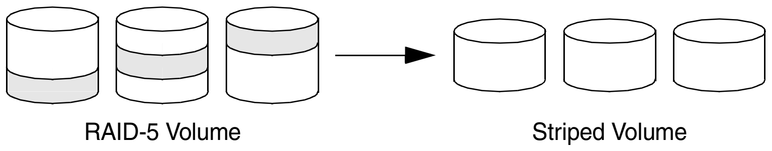
Example of Decreasing the Number of Columns in a Volume



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

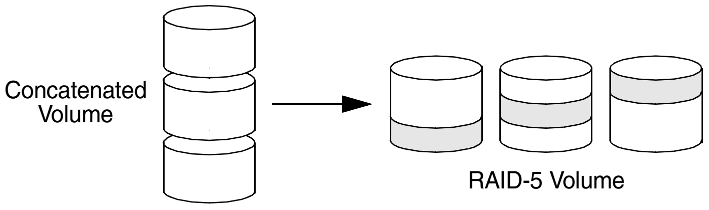
- Change a RAID-5 volume to a concatenated, striped, or layered volume (remove parity). See "Example of Relayout of a RAID-5 Volume to a Striped Volume" on page 39. Note that removing parity (shown by the shaded area) decreases the overall storage space that the volume requires.

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



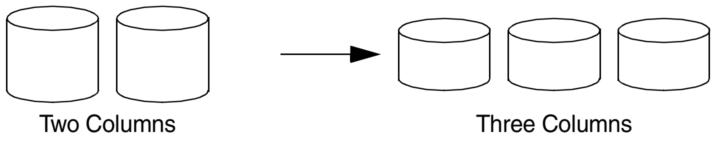
- Change a volume to a RAID-5 volume (add parity). See "Example of Relayout of a Concatenated Volume to a RAID-5 Volume" on page 39. Note that adding parity (shown by the shaded area) increases the overall storage space that the volume requires.

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



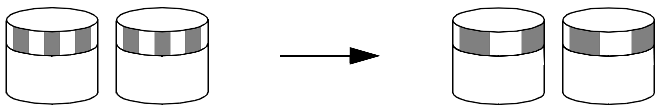
- Change the number of columns in a volume. See "Example of Increasing the Number of Columns in a Volume" on page 39. Note that the length of the columns is reduced to conserve the size of the volume.

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" on page 40.

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 219. For information about the relayout transformations that are possible, see "Permitted Relayout Transformations" on page 220.

Limitations of Online Relayout

Note the following limitations of online relayout:

- Log plexes cannot be transformed.
- Volume snapshots cannot be taken when there is an online relayout operation running on the volume.
- Online relayout cannot create a non-layered mirrored volume in a single step. It always creates a layered mirrored volume even if you specify a non-layered mirrored layout, such as `mirror-stripe` or `mirror-concat`. Use the `vxassist convert` command to turn the layered mirrored volume that results from a relayout into a non-layered volume. See "Converting Between Layered and Non-Layered Volumes" on page 225 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. Relayout is not possible unless you make the layouts of the individual plexes identical.
- 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 211.

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 the disk group that is automatically imported at boot time (usually aliased as the reserved system-wide disk group, `bootdg`), 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 "Dirty Region Logging (DRL)" on page 43 for more information.

For raw volumes used by database applications, the SmartSync Recovery Accelerator can be used if this is supported by the database vendor (see "SmartSync Recovery Accelerator" on page 45).

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.

Note Configuration of a DRL log within a version 20 Data Change Object (DCO) is not supported in this release.

Dirty Region Logs

DRL logically divides a volume into a set of consecutive regions, and maintains a log on disk where each region is represented by a status bit. This log records regions of a volume for which writes are pending. Before data is written to a region, DRL synchronously marks the corresponding status bit in the log as *dirty*. To enhance performance, the log bit remains set to dirty until the region becomes the least recently accessed for writes. This allows writes to the same region to be written immediately to disk if the region's log bit is set to dirty.

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 and Plexes

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

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

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

Sequential DRL

Some volumes, such as those that are used for database replay logs, are written sequentially and do not benefit from delayed cleaning of the DRL bits. For these volumes, *sequential DRL* can be used to limit the number of dirty regions. This allows for faster recovery 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 a tunable as detailed in the description of `voldrl_max_seq_dirty` in “Tunable Parameters” on page 292.

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 Dirty Region Logging to a Mirrored Volume” on page 206.

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. If supported by the database vendor, 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`. For a description of `volcvm_smartsync`, see "Tunable Parameters" on page 292.

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 additional information, see “Sequential DRL” on page 44.

Volume Snapshots

Note The Data Change Object (DCO) and FastResync features, which allow fast resynchronization of snapshots, and instant snapshot functionality are not supported in this release.

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

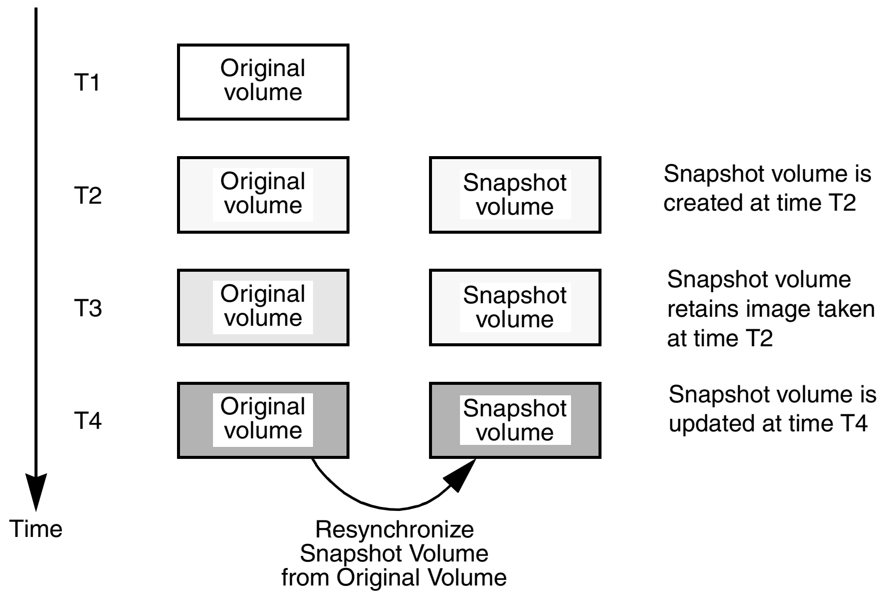
The figure, “Volume Snapshot as a Point-In-Time Image of a Volume” on page 47, illustrates how a snapshot volume represents a copy of an original volume at a given point in time. Even though the contents of the original volume can change, the snapshot volume can be used to preserve the contents of the original volume as they existed at an earlier time.

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

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

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

Volume Snapshot as a Point-In-Time Image of a Volume


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

The type of volume snapshot in VxVM is of the *third-mirror break-off* type. This name comes from its implementation where a snapshot plex (or third mirror) is added to a mirrored volume. The contents of the snapshot plex are then synchronized from the original plexes of the volume. When this synchronization is complete, the snapshot plex can be detached as a snapshot volume for use in backup or decision support applications. At a later time, the snapshot plex can be reattached to the original volume, requiring a full resynchronization of the snapshot plex's contents. For more information about this type of snapshot, see "Third-Mirror Break-Off Snapshots" on page 228.

For more information about taking snapshots of a volume, see "Administering Volume Snapshots" on page 227, and the `vxassist`(1M) manual page.

Hot-Relocation

Note You need an full 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.

Volume Sets

Note You need a full license to use this feature.

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

For more information about creating and administering volume sets, see "Creating and Administering Volume Sets" on page 237.

Administering Disks 2

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, and removing and replacing disks.

Note Most VxVM commands require superuser or equivalent privileges.

EFI disks are not currently supported by VxVM on the Solaris x64 platform.

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 `diskgroup`*##* where *diskgroup* is the name of the disk group to which a disk is being added and *##* is a sequence number. Your system may use device names that differ from those given in the examples.

The *device name* (sometimes referred to as *devname* or *disk access name*) defines the name of a disk device as it is known to the operating system. Such devices are usually, but not always, located in the `/dev/[r]dsk` directories. Devices that are specific to hardware from certain vendors may have different path names.

VxVM recreates disk devices, including those from the `/dev/[r]dsk` directories, as *metadevices* in the `/dev/vx/[r]dmp` directories. The dynamic multipathing (DMP) feature of VxVM uses these metadevices (or *DMP nodes*) to represent disks that can be accessed by more than one physical path, usually via different controllers. The number of access paths that are available depends on whether the disk is a single disk, or is part of a multiported disk array that is connected to a system.

You can use the `vxdisk` utility to display the paths 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.

Device names may also be remapped as enclosure-based names as described in the following section.

Disk Device Naming in VxVM

There are two different methods of naming disk devices:

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

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

c#t#d#s# Based Naming

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 slice on the disk.

Note On the SPARC platform, for non-EFI disks, the slice `s2` represents the entire disk. For both EFI and non-EFI disks, the entire disk is implied if the slice is omitted from the device name.

On the x86 platform, unlike the SPARC platform, the special partition `p0` refers to the entire physical disk (for example, `c0t0d0p0`), and the `s2` partition indicates only the Solaris partition on the disk (for example, `c0t0d0s2`). An additional device, `s8`, is used to denote the boot slice. The entire disk is 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`.

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 (JBOD disks) are named using the Disk_# format.
- Disks in the OTHER_DISKS category (disks that are not multipathed by DMP) are named using the fabric_# format

Enclosure Based Naming

Enclosure-based naming operates as follows:

- Devices with very long device names (for example, Fibre Channel devices that include worldwide name (WWN) identifiers) are always represented by enclosure-based names.
- 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 104 and the vxdmpadm(1M) manual page for more information.)
- Disks in the DISKS category (JBOD disks) are named using the Disk_# format.
- Disks in the OTHER_DISKS category (disks that are not multipathed by DMP) 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 63 for details of how to switch between the two naming schemes.

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

```
# vxdisk path | egrep diskname
```

For information on how to rename an enclosure, see "Renaming an Enclosure" on page 119.

For a description of disk categories, see "Disk Categories" on page 55.

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. If required, the value for the private region size may be overridden when you add or replace a disk using the `vxdiskadm` command. Each disk that has a private region holds an entire copy of the configuration database for the disk group. The size of the configuration database for a disk group is limited by the size of the *smallest* copy of the configuration database on any of its member disks.
public region	An area that covers the remainder of the disk, and which is used for the allocation of storage space to subdisks.

A disk's type identifies how VxVM accesses a disk, and how it manages the disk's private and public regions. The following disk access types are used by VxVM:

`sliced`	The public and private regions are on different disk partitions.
`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. Such disks are most useful for defining special devices (such as RAM disks, if supported) on which private region data would not persist between reboots. They can also be used to encapsulate disks where there is insufficient room for a private region. The disks cannot store configuration and log copies, and they do not support the use of the `vxdisk addregion` command to define reserved regions. VxVM cannot track the movement of `nopriv` disks on a SCSI chain or between controllers.
`auto`	When the `vxconfigd` daemon is started, VxVM obtains a list of known disk device addresses from the operating system and configures disk access records for them automatically.

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

`cdsdisk`	The disk is formatted as a Cross-platform Data Sharing (CDS) disk that is suitable for moving between different operating systems. This is the default format for disks that are not used to boot the system.Typically, most disks on a system are configured as this disk type.
`simple`	The disk is formatted as a simple disk that can be converted to a CDS disk.
`sliced`	The disk is formatted as a sliced disk.

See the `vxcdsconvert`(1M) manual page for information about the utility that you can use to convert disks to the `cdsdisk` format.

By default, auto-configured disks are formatted as `cdsdisk` disks when they are initialized for use with VxVM. You can change the default format by using the `vxdiskadm`(1M) command to update the `/etc/default/vxdisk` defaults file as described in "Displaying and Changing Default Disk Layout Attributes" on page 68. See the `vxdisk`(1M) manual page for details of the usage of this file, and for more information about disk types and their configuration.

VxVM initializes each new disk with the smallest possible number of partitions. For non-EFI 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.

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.

Partial Device Discovery

The Dynamic Multipathing (DMP) feature of VxVM supports partial device discovery where you can include or exclude sets of disks or disks attached to controllers from the discovery process.

The `vxdisk scandisks` command re-scans the devices in the OS device tree and triggers a DMP reconfiguration. You can specify parameters to `vxdisk scandisks` to implement partial device discovery. For example, this command makes VxVM discover newly added devices that were unknown to it earlier:

```
# vxdisk scandisks new
```

The next example discovers fabric devices (that is, devices with the characteristic `DDI_NT_FABRIC` property set on them):

```
# vxdisk scandisks fabric
```

The following command scans for the devices `c1t1d0` and `c2t2d0`:

```
# vxdisk scandisks device=c1t1d0,c2t2d0
```

Alternatively, you can specify a ! prefix character to indicate that you want to scan for all devices *except* those that are listed:

```
# vxdisk scandisks !device=c1t1d0,c2t2d0
```

You can also scan for devices that are connected (or not connected) to a list of logical or physical controllers. For example, this command discovers and configures all devices except those that are connected to the specified logical controllers:

```
# vxdisk scandisks !ctlr=c1,c2
```

The next command discovers devices that are connected to the specified physical controller:

```
# vxdisk scandisks pctlr=/pci@1f,4000/scsi@3/
```

Note The items in a list of physical controllers are separated by + characters.

You can use the command `vxdmpadm getctlr all` to obtain a list of physical controllers.

You can specify only one selection argument to the `vxdisk scandisks` command. Specifying multiple options results in an error.

For more information, see the `vxdisk`(1M) manual page.

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.

Disk Categories

Disk arrays that have been certified for use with VERITAS Volume Manager are supported by an array support library (ASL), and are categorized by the vendor ID string that is returned by the disks (for example, `HITACHI` and `DGC`).

Disks in JBODs for which DMP (see "Administering Dynamic Multipathing (DMP)" on page 95) can be supported in Active/Active mode, and which are capable of being multipathed, are placed in the `DISKS` category. Disks in unsupported arrays can be placed in this category by following the steps given in "Adding Unsupported Disk Arrays to the DISKS Category" on page 59.

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

Adding Support for a New Disk Array

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.

Enabling Discovery of New Devices

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.

Note This command ensures that dynamic multipathing is set up correctly on the array. Otherwise, VxVM treats the independent paths to the disks as separate devices, which can result in data corruption.

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.

Third-Party Driver Coexistence

The third-party driver (TPD) coexistence feature of VxVM 4.1 allows I/O that is controlled by third-party multipathing drivers to bypass DMP while retaining the monitoring capabilities of DMP. Provided that a suitable ASL is available, devices that use TPDs can be discovered without requiring you to set up a specification file, or to run a special command. In previous releases, VxVM only supported TPD coexistence if the code of the third-party driver was intrusively modified. The new TPD coexistence feature maintains backward compatibility with such methods, but it also permits coexistence without require any change in a third-party multipathing driver.

See “Changing Device Naming for TPD-Controlled Enclosures” on page 63 for information on how to change the form of TPD device names that are displayed by VxVM.

See “Displaying Information About TPD-Controlled Devices” on page 107 for details of how to find out the TPD configuration information that is known to DMP.

Autodiscovery of EMC Symmetrix Arrays

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

- ◆ With EMC PowerPath installed, such devices could be configured as foreign devices as described in “Adding Foreign Devices” on page 61.
- ◆ Without EMC PowerPath installed, DMP could be used to perform multipathing.

On upgrading a system to VxVM 4.1, an ASL provided by VERITAS makes information about any existing EMC PowerPath devices available to VxVM. Such devices can be discovered by DDL, and configured into DMP as autoconfigured disks with DMP nodes, even if PowerPath is being used to perform multipathing. There is no longer any need to configure such arrays as foreign devices.

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

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. The DDL is administered using the `vxddladm` utility, which can be used 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 disks that are supported in the `DISKS` (JBOD) category.
- Add disks from different vendors to the `DISKS` category.
- Remove disks from the `DISKS` category.
- Add disks as foreign devices.

The following sections explain these tasks in more detail. 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 all
```

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

To display more detailed information about a particular array library, use this form of the command:

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

This command displays the vendor ID (`VID`), product IDs (`PID`s) for the arrays, array types (for example, A/A or A/P), and array names. The following is sample output.

```
# vxddladm listsupport libname=libvxfujitsu.so
ATTR_NAME            ATTR_VALUE
==================================================
LIBNAME              libvxfujitsu.so
VID                  vendor
PID                  GR710, GR720, GR730
                     GR740, GR820, GR840
ARRAY_TYPE           A/A, A/P
ARRAY_NAME           FJ_GR710, FJ_GR720, FJ_GR730
                     FJ_GR740, FJ_GR820, FJ_GR840
```

Excluding Support for a Disk Array Library

To exclude all arrays that depend on a particular array library from participating in device discovery, use the following command:

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

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

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

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

Re-including Support for an Excluded Disk Array Library

If you have excluded support for all arrays that depend on a particular disk array library, you can use the `includearray` keyword to remove the entry from the exclude list, as shown in the following example:

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

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 arrays and add their 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 DISKS Category

To list disks that are supported in the `DISKS` (JBOD) category, use the following command:

```
# vxddladm listjbod
```

Adding Unsupported Disk Arrays to the DISKS Category

Caution The procedure in this section ensures that Dynamic Multipathing (DMP) is set up correctly on an array that is not supported by VERITAS Volume Manager. Otherwise, VERITAS Volume Manager treats the independent paths to the disks as separate devices, which can result in data corruption.

To add an unsupported disk array:

1. Use the following command to identify the vendor ID and product ID of the disks in the array:

   ```
   # /etc/vx/diag.d/vxdmpinq device_name
   ```

   where *device_name* is the device name of one of the disks in the array (for example, `/dev/rdsk/c1t20d0s2`). Note the values of the vendor ID (VID) and product ID (PID) in the output from this command. For Fujitsu disks, also note the number of characters in the serial number that is displayed. The following is sample output:

   ```
   # /etc/vx/diag.d/vxdmpinq /dev/rdsk/c1t20d0s2
   Vendor id (VID) : SEAGATE
   Product id (PID): ST318404LSUN18G
   Revision        : 8507
   Serial Number   : 0025T0LA3H
   ```

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

2. Enter the following command to add a new JBOD category:

   ```
   # vxddladm addjbod vid=vendorid pid=productid [length=serialno_length]
   ```

   where *vendorid* and *productid* are the VID and PID values that you found from the previous step. For example, *vendorid* might be `FUJITSU`, `IBM`, or `SEAGATE`. For Fujitsu devices, you must also specify the number of characters in the serial number as the argument to the `length` argument (for example, 10).

   Note In VxVM 4.0 and later releases, a `SEAGATE` disk is added as a JBOD device by default.

Continuing the previous example, the command to define an array of disks of this type as a JBOD would be:

```
# vxddladm addjbod vid=SEAGATE pid=ST318404LSUN18G
```

3. Use the `vxdctl enable` command to bring the array under VxVM control as described in "Enabling Discovery of New Devices" on page 55:

```
# vxdctl enable
```

4. To verify that the array is now supported, enter the following command:

```
# vxddladm listjbod
```

The following is sample output from this command for the example array:

```
VID       PID       Opcode  Page Code  Page Offset  SNO length
=================================================================
SEAGATE   ALL PIDs  18      -1         36           12
```

5. To verify that the array is recognized, use the `vxdmpadm listenclosure` command as shown in the following sample output for the example array:

```
# vxdmpadm listenclosure all
ENCLR_NAME     ENCLR_TYPE     ENCLR_SNO      STATUS
=========================================================
OTHER_DISKS    OTHER_DISKS    OTHER_DISKS    CONNECTED
Disk           Disk           DISKS          CONNECTED
```

The enclosure name and type for the array are both shown as being set to `Disk`. You can use the `vxdisk list` command to display the disks in the array:

```
# vxdisk list
DEVICE    TYPE       DISK      GROUP      STATUS
Disk_0    auto:none  -         -          online invalid
Disk_1    auto:none  -         -          online invalid
...
```

6. To verify that the DMP paths are recognized, use the `vxdmpadm getdmpnode` command as shown in the following sample output for the example array:

```
# vxdmpadm getdmpnode enclosure=Disk
NAME    STATE    ENCLR-TYPE   PATHS  ENBL    DSBL   ENCLR-NAME
==================================================================
Disk_0  ENABLED  Disk         2      2       0      Disk
Disk_1  ENABLED  Disk         2      2       0      Disk
...
```

This shows that there are two paths to the disks in the array.

For more information, enter the command `vxddladm help addjbod`, or see the `vxddladm`(1M) and `vxdmpadm`(1M) manual pages.

Removing Disks from the DISKS Category

To remove disks from the `DISKS` (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
```

Adding Foreign Devices

DDL may not be able to discover some devices that are controlled by third-party drivers, such as those that provide multipathing or RAM disk capabilities. For these devices it may be preferable to use the multipathing capability that is provided by the third-party drivers for some arrays rather than using the Dynamic Multipathing (DMP) feature. Such *foreign devices* can be made available as simple disks to VxVM by using the `vxddladm addforeign` command. This also has the effect of bypassing DMP for handling I/O. The following example shows how to add entries for block and character devices in the specified directories:

```
# vxddladm addforeign blockdir=/dev/foo/dsk chardir=/dev/foo/rdsk
```

By default, this command suppresses any entries for matching devices in the OS-maintained device tree that are found by the autodiscovery mechanism. You can override this behavior by using the `-f` and `-n` options as described on the `vxddladm`(1M) manual page.

After adding entries for the foreign devices, use either the `vxdisk scandisks` or the `vxdctl enable` command to discover the devices as simple disks. These disks then behave in the same way as autoconfigured disks.

The foreign device mechanism was introduced in VxVM 4.0 to support non-standard devices such as ramdisks, some solid state disks, and pseudo-devices such as EMC PowerPath. This mechanism has a number of limitations:

- A foreign device is always considered as simple disk with a single path. Unlike an autodiscovered disk, it does not have a DMP node.

- It is not supported for shared disk groups in a clustered environment. Only standalone host systems are supported.
- It is not supported for Persistent Group Reservation (PGR) operations.
- It is not under the control of DMP, so enabling of a failed disk cannot be automatic, and DMP administrative commands are not applicable.
- Enclosure information is not available to VxVM. This can reduce the availability of any disk groups that are created using such devices.

If a suitable ASL is available for an array, these limitations are removed, as described in "Third-Party Driver Coexistence" on page 56.

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 you specify a disk group, VxVM places new disks in a default disk group according to the rules given in "Rules for Determining the Default Disk Group" on page 125.

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.

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

To exclude a device from the view of VxVM, select item 16 (`Prevent multipathing/Suppress devices from VxVM's view`) from the `vxdiskadm` main menu. See "Disabling and Enabling Multipathing for Specific Devices" on page 99 for details.

Changing the Disk-Naming Scheme

Note Devices with very long device names (for example, Fibre Channel devices that include worldwide name (WWN) identifiers) are always represented by enclosure-based names. The operation in this section has no effect on such devices.

You can either use enclosure-based naming for disks or the operating system's 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. When prompted, enter **`y`** to change the naming scheme. This restarts the `vxconfig` daemon to bring the new disk naming scheme into effect.

Alternatively, you can change the naming scheme from the command line. The following commands select enclosure-based and operating system-based naming respectively:

```
# vxddladm set namingscheme=ebn
# vxddladm set namingscheme=osn
```

The change is immediate whichever method you use.

Changing Device Naming for TPD-Controlled Enclosures

Note This feature is available only if the default disk-naming scheme is set to use operating system-based naming, and the TPD-controlled enclosure does not contain fabric disks.

For disk enclosures that are controlled by third-party drivers (TPD) whose coexistence is supported by an appropriate ASL, the default behavior is to assign device names that are based on the TPD-assigned node names. You can use the `vxdmpadm` command to switch between these names and the device names that are known to the operating system:

```
# vxdmpadm setattr enclosure enclosure tpdmode=native|pseudo
```

The argument to the `tpdmode` attribute selects names that are based on those used by the operating system (`native`), or TPD-assigned node names (`pseudo`).

The use of this command to change between TPD and operating system-based naming is illustrated in the following example for the enclosure named `EMC0`:

```
# vxdisk list
DEVICE         TYPE           DISK      GROUP     STATUS
emcpower10s2   auto:sliced    disk1     mydg      online
emcpower11s2   auto:sliced    disk2     mydg      online
emcpower12s2   auto:sliced    disk3     mydg      online
emcpower13s2   auto:sliced    disk4     mydg      online
emcpower14s2   auto:sliced    disk5     mydg      online
emcpower15s2   auto:sliced    disk6     mydg      online
emcpower16s2   auto:sliced    disk7     mydg      online
emcpower17s2   auto:sliced    disk8     mydg      online
emcpower18s2   auto:sliced    disk9     mydg      online
emcpower19s2   auto:sliced    disk10    mydg      online

# vxdmpadm setattr enclosure EMC0 tpdmode=native

# vxdisk list
DEVICE         TYPE           DISK      GROUP     STATUS
c6t0d10s2      auto:sliced    disk1     mydg      online
c6t0d11s2      auto:sliced    disk2     mydg      online
c6t0d12s2      auto:sliced    disk3     mydg      online
c6t0d13s2      auto:sliced    disk4     mydg      online
c6t0d14s2      auto:sliced    disk5     mydg      online
c6t0d15s2      auto:sliced    disk6     mydg      online
c6t0d16s2      auto:sliced    disk7     mydg      online
c6t0d17s2      auto:sliced    disk8     mydg      online
c6t0d18s2      auto:sliced    disk9     mydg      online
c6t0d19s2      auto:sliced    disk10    mydg      online
```

If `tpdmode` is set to `native`, the path with the smallest device number is displayed.

Discovering the Association between Enclosure-Based Disk Names and OS-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#s#` names. To discover the `c#t#d#s#` 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=enclosure-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/vx/dmp` or `/dev/vx/rdmp`.

Regenerating the Persistent Device Name Database

The persistent device naming feature, introduced in VxVM 4.1, makes the names of disk devices persistent across system reboots. If operating system-based naming is selected, each disk name is usually set to the name of one of the paths to the disk. After hardware reconfiguration and a subsequent reboot, the operating system may generate different names for the paths to the disks. As DDL assigns persistent disk names using the persistent device name database that was generated during a previous boot session, the disk names may no longer correspond to the actual paths. This does not prevent the disks from being used, but the association between the disk name and one of its paths is lost.

To find the relationship between a disk and its paths, run one of the following commands:

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

If you want to update the disk names so that they correspond to the new path names, perform the following steps:

1. Remove the file that contains the existing persistent device name database:

   ```
   # rm /etc/vx/disk.info
   ```

2. Restart the VxVM configuration demon:

   ```
   # vxconfigd -k
   ```

   This regenerates the persistent name database.

Issues Regarding Simple/Nopriv Disks with Enclosure-Based Naming

If you change from *c#t#d#s#* based naming to enclosure-based naming, 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 Boot Disk Group
- Simple/Nopriv Disks in Non-Boot 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 c#t#d#s# naming 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 Boot Disk Group

If the boot disk group (usually aliased as `bootdg`) 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 c#t#d#s# naming.

2. Enter the following command to restart the VxVM configuration daemon:

   ```
   # vxconfigd -kr reset
   ```

3. If you want to use enclosure-based naming, use `vxdiskadm` to add a sliced disk to the `bootdg` disk group, change back to the enclosure-based naming scheme, and then run the following command:

   ```
   # /etc/vx/bin/vxdarestore
   ```

Simple/Nopriv Disks in Non-Boot Disk Groups

If an imported disk group, other than `bootdg`, 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:

   ```
   # /etc/vx/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 default disk layout on the x64 platform differs from that on the SPARC platform as follows:

- On a SPARC system, the start of the Solaris partition (which may contain a primary boot executable and boot block in addition to the volume table of contents (VTOC) and any disk slices) is located in cylinder 0. The whole disk is accessed using the device `c#t#d#s2`.
- On an x64 system, an FDISK partition (which may contain a master boot record (MBR)) is located in cylinder 0, and the start of the Solaris partition is located in cylinder 1. The device `c#t#d#s2` references the entire Solaris partition, but not the FDISK partition. The whole disk may be accessed using the device `c#t#d#p0`.

Before a disk with a sun partition label from a Solaris SPARC system can be used on a Solaris x86 system, it is necessary to use the `fdisk` command to rewrite its partition layout and VTOC, so destroying any data on the disk. However, a CDS disk group can be imported on a Solaris x86 system without needing to run the `fdisk` command. The layout of the partition table for CDS disks is the same on all supported platforms, and does not include an FDISK partition, or a Solaris partition and VTOC.

As on the Solaris SPARC platform, you can use the VERITAS Enterprise Administrator (VEA) GUI or the `vxdiskadm`, `vxdiskadd` or `vxdisk` commands to initialize a new disk with one of the following formats: `auto:cdsdisk`, `auto:simple`, `auto:sliced`, `nopriv`, `simple` or `sliced`.

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

Displaying and Changing Default Disk Layout Attributes

To display or change the default values for initializing or encapsulating disks, select menu item `22 (Change/display the default disk layouts)` from the `vxdiskadm` main menu. For disk initialization, you can change the default format and the default length of the private region. For disk encapsulation, you can additionally change the offset values for both the private and public regions.

The attribute settings for initializing disks are stored in the file, `/etc/default/vxdisk`, and those for encapsulating disks in `/etc/default/vxencap`.

See the `vxdisk`(1M) and `vxencap`(1M) manual pages for more information.

Adding Disks 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 names of the disks 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       mydg01       mydg         online
c0t1d0       mydg02       mydg         online
c0t2d0       mydg03       mydg         online
c0t3d0       -            -            online
c1t0d0       mydg10       mydg         online
c1t0d1       -            -            online invalid
.
.
.
c3t0d0       -            -             online invalid
sena0_0       mydg33      mydg         online
sena0_1       mydg34      mydg         online
sena0_2       mydg35      mydg         online

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

The phrase `online invalid` in the `STATUS` line indicates that a disk has yet to be added or initialized for VxVM control. Disks that are listed as `online` with a disk name and disk group are already under VxVM control.

Enter the device name or pattern of the disks that you want to initialize at the prompt and press Return.

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

   ```
   Here are the disks selected.  Output format: [Device]
   ```

   list of device names

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

4. At the following prompt, specify the disk group to which the disks should be added, or **none** to reserve the disks for future use:

   ```
   You can choose to add these disks to an existing disk group,
   a new disk group, or you can leave these disks available for use
   by future add or replacement operations. To create a new disk
   group, select a disk group name that does not yet exist. To leave
   the disks available for future use, specify a disk group name of
   "none".

   Which disk group [<group>,none,list,q,?]
   ```

5. If you specified the name of a disk group that does not already exist, `vxdiskadm` prompts for confirmation that you really want to create this new disk group:

   ```
   There is no active disk group named disk group name.

   Create a new group named disk group name? [y,n,q,?] (default: y) y
   ```

   You are then prompted to confirm whether the disk group should support the Cross-platform Data Sharing (CDS) feature:

   ```
   Create the disk group as a CDS disk group? [y,n,q,?] (default: n)
   ```

   If the new disk group may be moved between different operating system platforms, enter **y**. Otherwise, enter **n**.

6. At the following prompt, either press Return to accept the default disk names 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. If you see the following prompt, it 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,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,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 not to encapsulate the disks, `vxdiskadm` asks if you want to initialize the disks instead of encapsulating them. Enter **`y`** to confirm this:

```
Instead of encapsulating, initialize? [y,n,q,?] (default: n) y
```

`vxdiskadm` now confirms those disks that are being initialized and added to VxVM control with messages similar to the following. In addition, you may be prompted to perform surface analysis.

```
VxVM INFO V-5-2-205 Initializing device device name.
```

❖ If you choose to encapsulate the disks, `vxdiskadm` confirms their device names and prompts you for permission to proceed. Enter **`y`** (or press Return) to continue encapsulation:

```
VxVM NOTICE V-5-2-311 The following disk device has been
selected for encapsulation.
Output format: [Device]
device name

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

`vxdiskadm` now displays an encapsulation status, and informs you that you must perform a shutdown and reboot as soon as possible:

```
The disk device device name will be encapsulated and added to the
disk group dgname with the disk name disk name.
```

12. You can now choose whether the disk is to be formatted as a CDS disk that is portable between different operating systems, or as a non-portable sliced or simple disk:

```
Enter desired format [cdsdisk,sliced,simple,q,?]
(default: cdsdisk)
```

Enter the format that is appropriate for your needs. In most cases, this is the default format, `cdsdisk`.

13. At the following prompt, `vxdiskadm` asks if you want to use the default private region size of 2048 blocks. Press Return to confirm that you want to use the default value, or enter a different value. (The maximum value that you can specify is 524288 blocks.)

```
Enter desired private region length [<privlen>,q,?]
(default: 2048)
```

❖ If you are initializing one or more disks, `vxdiskadm` proceeds to add the disks.

```
VxVM INFO V-5-2-120 Adding disk device device name to disk group
disk group name with disk name disk name.
...
```

❖ If you are encapsulating one or more disks, and you entered `cdsdisk` as the format in step 12 on page 72, you are prompted for the action to be taken if the disks cannot be converted this format:

```
Do you want to use 'sliced' as the format should 'cdsdisk' fail?
[y,n,q,?] (default: y)
```

If you enter **`y`**, and it is not possible to encapsulate a disk as a CDS disk, it is encapsulated as a sliced disk. Otherwise, the encapsulation fails.

`vxdiskadm` then proceeds to encapsulate the disks.

```
VxVM INFO V-5-2-340 The first stage of encapsulation has
completed successfully. You should now reboot your system at the
earliest possible opportunity.
```

The encapsulation requires two or three reboots which will take place automatically after the next reboot. To shut down and reboot you system, use the command:

```
# shutdown -g0 -y -i6
```

Note The `/etc/vfstab` file is updated so that the file systems on the disk devices are mounted using the new volume devices that are created for the encapsulated file systems in `/dev/vx/dsk/`*`diskgroup`* and `/dev/vx/rdsk/`*`diskgroup`*. You may need to update any other references to the old devices in backup scripts, databases or manually created swap devices so that they refer to the new volumes.

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

14. At the following prompt, indicate whether you want to continue to add more disks:

```
Add or initialize other disks? [y,n,q,?] (default: n)
```

See "Displaying and Changing Default Disk Layout Attributes" on page 68 for details of how to change the default layouts that are used to initialize or encapsulate disks.

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 Disks to VxVM" on page 68 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 78), 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 Disks to VxVM" on page 68.

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.

Note Encapsulation of the root disk is not supported in this release.

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

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 a disk group, enter the name of the disk group (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,?]
```

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 dgname
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 *diskgroup* with the disk name *diskgroup*01.
```

7. You can now choose whether the disk is to be formatted as a CDS disk that is portable between different operating systems, or as a non-portable sliced disk:

   ```
   Enter the desired format [cdsdisk,sliced,q,?] (default: cdsdisk)
   ```

   Enter the format that is appropriate for your needs. In most cases, this is the default format, `cdsdisk`.

8. At the following prompt, `vxdiskadm` asks if you want to use the default private region size of 2048 blocks. Press Return to confirm that you want to use the default value, or enter a different value. (The maximum value that you can specify is 524288 blocks.)

   ```
   Enter desired private region length [<privlen>,q,?]
   (default: 2048)
   ```

9. If you entered `cdsdisk` as the format in step 7, you are prompted for the action to be taken if the disk cannot be converted this format:

   ```
   Do you want to use 'sliced' as the format should 'cdsdisk' fail?
   [y,n,q,?] (default: y)
   ```

   If you enter **y**, and it is not possible to encapsulate the disk as a CDS disk, it is encapsulated as a sliced disk. Otherwise, the encapsulation fails.

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

    ```
    VxVM NOTICE V-5-2-311 The *device name* disk has been configured for
    encapsulation.

    VxVM INFO V-5-2-340 The first stage of encapsulation has
    completed successfully. You should now reboot your system at the
    earliest possible opportunity.
    ```

    The encapsulation requires two or three reboots which will take place automatically after the next reboot. To shut down and reboot you system, use the command:

    ```
    # shutdown -g0 -y -i6
    ```

Note The `/etc/vfstab` file is updated so that the file systems on the disk devices are mounted using the new volume devices that are created for the encapsulated file systems in `/dev/vx/dsk/`*`diskgroup`* and `/dev/vx/rdsk/`*`diskgroup`*. You may need to update any other references to the old devices in backup scripts, databases or manually created swap devices so that they refer to the new volumes.

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

11. 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
```

See "Displaying and Changing Default Disk Layout Attributes" on page 68 for details of how to change the default layout that is used to encapsulate disks.

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:

```
VxVM ERROR V-5-2-338 The encapsulation operation failed with the
following error:
It is not possible to encapsulate device, for the following reason:
<VxVM vxslicer ERROR V-5-1-1108 Unsupported disk layout.>
```

See the section, "Using nopriv Disks for Encapsulation" on page 78, 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 `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 length.

Note By default, `vxassist` re-initializes the data area of a volume that it creates. If there is data to be preserved on the partition, do not use `vxassist`. Instead, create the volume with `vxmake` and start the volume with the command `vxvol init active`.

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

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

Note You must disable a disk group as described in "Disabling a Disk Group" on page 140 before you can remove the last disk in that group. Alternatively, you can destroy the disk group as described in "Destroying a Disk Group" on page 141.

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 [-g diskgroup] 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.

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,?] mydg01
   ```

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:

   ```
   VxVM NOTICE V-5-2-284 Requested operation is to remove disk
   mydg01 from group mydg.

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

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

   ```
   VxVM INFO V-5-2-268 Removal of disk mydg01 is complete.
   ```

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

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:

```
VxVM ERROR V-5-2-369 The following volumes currently use part
of disk mydg02:

home usrvol

Volumes must be moved from mydg02 before it can be removed.

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

If you choose **`y`**, 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 216 and "Taking Plexes Offline" on page 164 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 `mydg02`:

```
Enter disk name [<disk>,list,q,?] mydg02

VxVM NOTICE V-5-2-284 Requested operation is to remove disk mydg02
from group mydg.

Continue with operation? [y,n,q,?] (default: y) y
VxVM INFO V-5-2-268 Removal of disk mydg02 is complete.
```

Removing a Disk from VxVM Control

After removing a disk from a disk group, you can clear the VxVM configuration on the disk and permanently remove it from VxVM control by running the `vxdiskunsetup` command:

```
# /usr/lib/vxvm/bin/vxdiskunsetup -C c#t#d#
```

Caution The `vxdiskunsetup` command removes a disk from VERITAS Volume Manager control by erasing the VxVM metadata on the disk. To prevent data loss, any data on the disk should first be evacuated from the disk. The `vxdiskunsetup` command should only be used by a system administrator who is trained and knowledgeable about VERITAS Volume Manager.

If the `vxdisk list` command shows that the disk is in the `error` state, use the following commands to reinitialize the disk with the default layout, and then remove it from the VxVM configuration:

```
# fdisk -B -n /dev/rdsk/c#t#d#[p0]
# vxdisk rm c#t#d#s2
# vxdisk scandisks
```

On an x86 system, the partition 0 device (for example, `c2t4d7p0`) must be specified to the `fdisk` command.

The `vxdisk list` command should now show the disk's type as `auto:none` and its state as `online invalid`. If the disk is not shown as being in this state, use the following command to clear the first 512 blocks on the disk before rerunning the `fdisk` and `vxdisk` commands:

```
# dd if=/dev/zero of=/dev/rdsk/c#t#d#[p0] count=512
```

As before, you must specify the partition 0 device on an x86 platform.

Removing and Replacing Disks

Note A replacement disk should have the same disk geometry as the disk that failed. That is, the replacement disk should have the same bytes per sector, sectors per track, tracks per cylinder and sectors per cylinder, same number of cylinders, and the same number of accessible cylinders. (You can use the `prtvtoc` command to obtain disk information.)

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,?] mydg02
```

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

```
VxVM NOTICE V-5-2-371 The following volumes will lose mirrors as
a result of this operation:

home src

No data on these volumes will be lost.

The following volumes are in use, and will be disabled as a
result of this operation:

mkting

Any applications using these volumes will fail future accesses.
These volumes will require restoration from backup.
```

```
Are you sure you want do this? [y,n,q,?] (default: n)
```

To remove the disk, causing the named volumes to be disabled and data to be lost when the disk is replaced, enter **y** or press Return.

To abandon removal of the disk, and back up or move the data associated with the volumes that would otherwise be disabled, enter **n** or **q** and press Return.

For example, to move the volume `mkting` to a disk other than `mydg02`, use this command:

```
# vxassist move mkting !mydg02
```

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** if you are going to replace the physical disk:

```
The following devices are available as replacements:
c0t1d0

You can choose one of these disks now, to replace mydg02.
Select "none" if you do not wish to select a replacement disk.

Choose a device, or select "none" [<device>,none,q,?]
(default: c0t1d0)
```

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 enter **none** because you intend to replace the physical disk, see the section "Replacing a Failed or Removed Disk" on page 87.

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

```
VxVM NOTICE V-5-2-285 Requested operation is to remove mydg02
from group mydg. The removed disk will be replaced with disk
device c0t1d0.

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

`vxdiskadm` displays the following messages to indicate that the original disk is being removed:

```
VxVM NOTICE V-5-2-265 Removal of disk mydg02 completed
successfully.
VxVM NOTICE V-5-2-260 Proceeding to replace mydg02 with device
c0t1d0.
```

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 c1t0d0 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: y) y
```

Caution It is recommended that you do not enter **`n`** at this prompt. This results in an invalid `VTOC` that makes the disk unbootable.

Entering **`y`** at the prompt destroys any data that is on the disk. Ensure that you have at least one valid copy of the data on other disks before proceeding.

6. You can now choose whether the disk is to be formatted as a CDS disk that is portable between different operating systems, or as a non-portable sliced or simple disk:

```
Enter the desired format [cdsdisk,sliced,simple,q,?]
(default: cdsdisk)
```

Enter the format that is appropriate for your needs. In most cases, this is the default format, `cdsdisk`.

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

```
Enter desired private region length [<privlen>,q,?]
(default: 2048)
```

8. If one of more mirror plexes were moved from the disk, you are now prompted whether FastResync should be used to resynchronize the plexes:

```
Use FMR for plex resync? [y,n,q,?] (default: n) y
```

`vxdiskadm` displays the following success message:

```
VxVM NOTICE V-5-2-158 Disk replacement completed successfully.
```

9. At the following prompt, indicate whether you want to remove another disk (**`y`**) or return to the `vxdiskadm` main menu (**`n`**):

```
Remove another disk? [y,n,q,?] (default: n)
```

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 253.

Replacing a Failed or Removed Disk

Note You may need to run commands that are specific to the operating system or disk array when replacing a physical disk.

Use the following procedure after you have replaced 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

VxVM INFO V-5-2-479 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,?] mydg02
```

3. The `vxdiskadm` program displays the device names of the disk devices available for use as replacement disks. Your system may use a device name that differs from the examples. Enter the device name of the disk or press Return to select the default device:

```
The following devices are available as replacements:
c0t1d0 c1t1d0

You can choose one of these disks to replace mydg02.
```

```
Choose "none" to initialize another disk to replace mydg02.

Choose a device, or select "none" [<device>,none,q,?]
(default: c0t1d0)
```

4. Depending on whether the replacement disk was previously initialized, perform the appropriate step from the following:

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

```
VxVM INFO V-5-2-378 The requested operation is to initialize disk
device c0t1d0 and to then use that device to replace the removed
or failed disk mydg02 in disk group mydg.
Continue with operation? [y,n,q,?] (default: y)
```

❖ If the disk has already been initialized, press Return at the following prompt to replace the disk:

```
VxVM INFO V-5-2-382 The requested operation is to use the
initialized device c0t1d0 to replace the removed or failed disk
mydg02 in disk group mydg.
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:

```
VxVM INFO V-5-2-876 The disk c0t1d0 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: y) y
```

Caution It is recommended that you do not enter **`n`** at this prompt. This can result in an invalid `VTOC` that makes the disk unbootable.

Entering **`y`** at the prompt destroys any data that is on the disk. Ensure that you have at least one valid copy of the data on other disks before proceeding.

5. You can now choose whether the disk is to be formatted as a CDS disk that is portable between different operating systems, or as a non-portable sliced or simple disk:

```
Enter the desired format [cdsdisk,sliced,simple,q,?]
(default: cdsdisk)
```

Enter the format that is appropriate for your needs. In most cases, this is the default format, `cdsdisk`.

6. At the following prompt, `vxdiskadm` asks if you want to use the default private region size of 2048 blocks. Press Return to confirm that you want to use the default value, or enter a different value. (The maximum value that you can specify is 524288 blocks.)

   ```
   Enter desired private region length [<privlen>,q,?]
   (default: 2048)
   ```

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

   ```
   VxVM NOTICE V-5-2-158 Disk replacement completed successfully.
   ```

   At the following prompt, indicate whether you want to replace another disk (**`y`**) or return to the `vxdiskadm` main menu (**`n`**):

   ```
   Replace another disk? [y,n,q,?] (default: n)
   ```

Enabling a 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

   VxVM INFO V-5-2-998 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 most needs 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

VxVM INFO V-5-2-474 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 [-g diskgroup] rename old_diskname new_diskname
```

For example, you might want to rename disk `mydg03`, as shown in the following output from `vxdisk list`, to `mydg02`: #

```
# vxdisk list
DEVICE      TYPE          DISK      GROUP     STATUS
c0t0d0s2    auto:sliced   mydg01    mydg      online
c1t0d0s2    auto:sliced   mydg03    mydg      online
c1t1d0s2    auto:sliced   -         -         online
```

You would use the following command to rename the disk.

```
# vxedit -g mydg rename mydg03 mydg02
```

To confirm that the name change took place, use the `vxdisk list` command again:

```
# vxdisk list
DEVICE      TYPE          DISK      GROUP     STATUS
c0t0d0s2    auto:sliced   mydg01    mydg      online
c1t0d0s2    auto:sliced   mydg02    mydg      online
c1t1d0s2    auto:sliced   -         -         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 [-g diskgroup] set reserve=on diskname
```

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

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

The `vxassist` command overrides the reservation and creates a 20 megabyte volume on mydg03. However, the command:

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

does not use mydg03, even if there is no free space on any other disk.

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

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

See the `vxedit`(1M) manual page 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    auto:sliced   mydg04    mydg      online
c1t0d0s2    auto:sliced   mydg03    mydg      online
c1t1d0s2    auto:sliced   -         -         online invalid
enc0_2      auto:sliced   mydg02    mydg      online
enc0_3      auto:sliced   mydg05    mydg      online
sena0_0     auto:sliced   -         -         online
sena0_1     auto:sliced   -         -         online
```

Note The phrase `online invalid` in the `STATUS` line indicates that a disk has not yet been added to VxVM control. These disks may or may not have been initialized by VxVM previously. Disks that are listed as `online` are already under VxVM control.

To display detailed information about a particular disk, use the following command:

```
# vxdisk list diskname
```

The following is sample output from this command:

```
# vxdisk list c3t3d7s2
Device:    c3t3d7s2
devicetag: c3t3d7
type:      auto
hostid:
disk:      name= id=1107972998.2800.test18
group:     name= id=
info:      format=cdsdisk,privoffset=256,pubslice=2,privslice=2
flags:     online ready private autoconfig autoimport
pubpaths:  block=/dev/vx/dmp/c3t3d7s2 char=/dev/vx/rdmp/c3t3d7s2
version:   3.1
iosize:    min=512 (bytes) max=2048 (blocks)
public:    slice=2 offset=2304 len=142799472 disk_offset=0
private:   slice=2 offset=256 len=2048 disk_offset=0
update:    time=1107972998 seqno=0.1
ssb:       actual_seqno=0.0
headers:   0 240
configs:   count=1 len=1280
logs:      count=1 len=192
Defined regions:
 config   priv 000048-000239[000192]: copy=01 offset=000000
disabled
 config   priv 000256-001343[001088]: copy=01 offset=000192
disabled
 log      priv 001344-001535[000192]: copy=01 offset=000000
disabled
 lockrgn  priv 001536-001679[000144]: part=00 offset=000000
Multipathing information:
numpaths:   2
c3t3d7s2       state=enabled
c2t11d7s2      state=enabled
```

See “Displaying Multipaths to a VM Disk” on page 102 for more information on interpreting the output from this command.

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

   VxVM INFO V-5-2-475 Use this menu operation to display a list of
   disks. You can also choose to list detailed information about the
   disk at a specific disk device address.

   Enter disk device or "all" [<address>,all,q,?] (default: all)
   ```

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

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

Administering Dynamic Multipathing (DMP) 3

Note No Active/Passive (A/P) arrays are supported in this release.

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.

How DMP Works

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.

See "Discovering and Configuring Newly Added Disk Devices" on page 53 for a description of how to make newly added disk hardware known to a host system.

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

◆ An *Active/Passive* array (*A/P array*) allows access to its LUNs (*logical units*; real disks or virtual disks created using hardware) via the *primary* (active) path on a single controller (also known as an *access port* or a *storage processor*) during normal operation.

  In *implicit failover mode* (or *autotrespass mode*), an A/P array automatically fails over by scheduling I/O to the *secondary* (passive) path on a separate controller if the primary path fails. This passive port is not used for I/O until the active port fails. In A/P arrays, path failover can occur for a single LUN if I/O fails on the primary path.

  In Active/Passive arrays with *LUN group failover* (*A/PG arrays*), a group of LUNs that are connected through a controller is treated as a single failover entity. Unlike A/P arrays, failover occurs at the controller level, and not for individual LUNs. The primary 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.

Active/Passive arrays in *explicit failover mode* (or *non-autotrespass mode*) are termed *A/PF arrays.* DMP issues the appropriate low-level command to make the LUNs fail over to the secondary path.

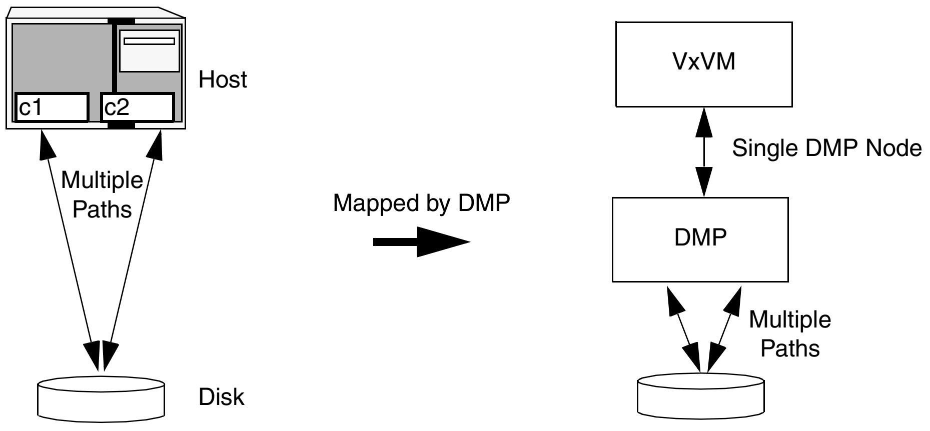
A/P-C, A/PF-C and A/PG-C arrays are variants of the A/P, AP/F and A/PG array types that support concurrent I/O and load balancing by having multiple primary paths into a controller. This functionality is provided by a controller with multiple ports, or by the insertion of a SAN hub or switch between an array and a controller. Failover to the secondary (passive) path occurs only if all the active primary paths fail.

- An *Active/Active* disk array (*A/A arrays*) permits several paths to be used concurrently for I/O. Such arrays allow DMP to provide greater I/O throughput by balancing the I/O load uniformly across the multiple paths to the LUNs. In the event that one path fails, DMP automatically routes I/O over the other available paths.

VxVM uses *DMP metanodes* (*DMP nodes*) to access disk devices connected to the system. For each disk in a supported array, DMP maps one node to the set of paths that are connected to the disk. Additionally, DMP associates the appropriate multipathing policy for the disk array with the node. For disks in an unsupported array, DMP maps a separate node to each path that is connected to a disk. The raw and block devices for the nodes are created in the directories `/dev/vx/rdmp` and `/dev/vx/dmp` respectively.

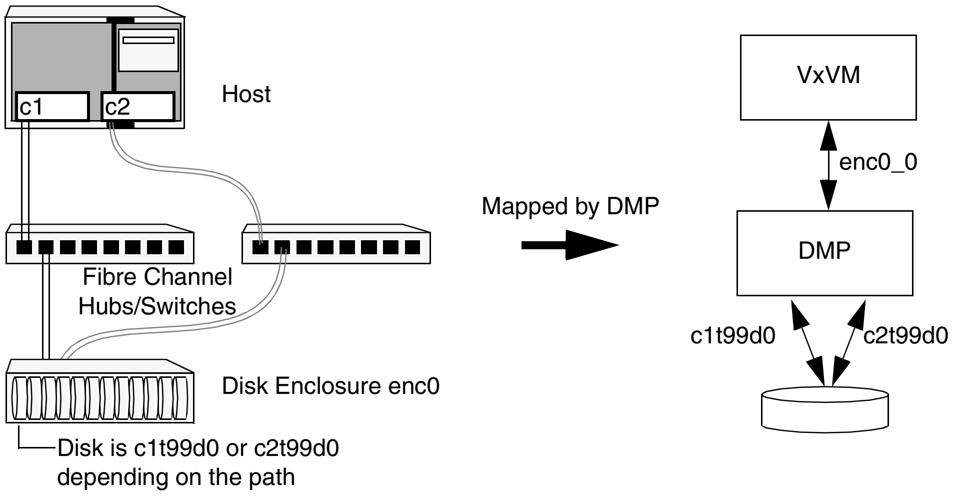
See the figure, "How DMP Represents Multiple Physical Paths to a Disk as One Node" on page 96, for an illustration of how DMP sets up a node for a disk in a supported disk array.

How DMP Represents Multiple Physical Paths to a Disk as One Node



As described in "Enclosure-Based Naming" on page 7, VxVM implements a disk device naming scheme that allows you to 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 node, `enc0_0`, to access it.

Example of Multipathing for a Disk Enclosure in a SAN Environment


c1
c2
Host
Fibre Channel Hubs/Switches
Disk Enclosure enc0
Disk is c1t99d0 or c2t99d0 depending on the path
Mapped by DMP
VxVM
enc0_0
DMP
c1t99d0
c2t99d0

See "Changing the Disk-Naming Scheme" on page 63 for details of how to change the naming scheme that VxVM uses for disk devices.

Note The persistent device naming feature, introduced in VxVM 4.1, makes the names of disk devices (DMP node names) persistent across system reboots. If operating system-based naming is selected, each disk name is usually set to the name of one of the paths to the disk. After hardware reconfiguration and a subsequent reboot, the operating system may generate different names for the paths to the disks. As DDL assigns persistent disk names using the out-of-date information that is stored in the persistent device name database, a disk name may no longer correspond to an actual path to the disk. Since DMP device node names are arbitrary, this does not prevent the disks from being used. See "Regenerating the Persistent Device Name Database" on page 65 for details of how to regenerate the persistent device name database, and restore the relationship between the disk and path names.

See "Discovering and Configuring Newly Added Disk Devices" on page 53 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

By default, DMP uses the *balanced path mechanism* to provide load balancing across paths for Active/Active, A/P-C, A/PF-C and A/PG-C 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 unless they support concurrent I/O.

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).

You can use the `vxdmpadm` command to change the I/O policy for the paths to an enclosure or disk array as described in “Specifying the I/O Policy” on page 113.

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*.

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" on page 99 and "Enabling Multipathing and Making Devices Visible to VxVM" on page 100.

Disabling Multipathing and Making Devices Invisible to VxVM

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

1. 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. You are prompted to confirm whether you want to continue.

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

```
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
```

   Help text and examples are provided onscreen for all the menu items.

❖ 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.

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

- Select option 3 to exclude disks from the view of VxVM that match a specified Vendor ID and Product ID.
- 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.
- Select option 5 to disable multipathing for all disks on a specified controller.
- Select option 6 to disable multipathing for specified paths. The disks that correspond to a specified path are claimed in the OTHER_DISKS category and are not multipathed.
- Select option 7 to disable multipathing for disks that match a specified Vendor ID and Product ID. The disks that correspond to a specified Vendor ID and Product ID combination are claimed in the OTHER_DISKS category and are not multipathed.
- Select option 8 to list the devices that are currently suppressed or 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.

1. 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. You are prompted to confirm whether you want to continue.

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

```
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 option 1 to make all paths through a specified controller visible to VxVM.
- Select option 2 to make specified paths visible to VxVM.
- Select option 3 to make disks visible to VxVM that match a specified Vendor ID and Product ID.
- 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.
- Select option 5 to enable multipathing for all disks that have paths through the specified controller.
- Select option 6 to enable multipathing for specified paths.
- Select option 7 to enable multipathing for disks that match a specified Vendor ID and Product ID.
- Select option 8 to list the devices that are currently suppressed or not multipathed.

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 an Active/Active storage array and you need to change an interface board that is connected to this disk array (if supported by the hardware), 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.

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 104.

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.

You can use the `vxdisk` path command to display the relationships between the device paths, disk access names, disk media names and disk groups on a system as shown here:

```
# vxdisk path
SUBPATH        DANAME        DMNAME       GROUP        STATE
c1t0d0s2       c1t0d0s2      mydg01       mydg         ENABLED
c4t0d0s2       c1t0d0s2      mydg01       mydg         ENABLED
c1t1d0s2       c1t1d0s2      mydg02       mydg         ENABLED
c4t1d0s2       c1t1d0s2      mydg02       mydg         ENABLED
...
```

This shows that two paths exist to each of the two disks, `mydg01` and `mydg02`, and also indicates that each disk is in the `ENABLED` state.

To view 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 from the `vxdisk list` command is as follows:

```
Device     c2t0d0
devicetag  c2t0d0
type       sliced
hostid     aparajita
disk       name=mydg01 id=861086917.1052.aparajita
group      name=mydg 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 the EMC CLARiiON, Hitachi HDS 9200 and 9500, Sun StorEdge 6xxx product family, and Sun StorEdge T3 array. This field indicates the *primary* and *secondary* paths to the disk.

The `type` field is not displayed for disks on Active/Active type disk arrays such as the EMC Symmetrix, Hitachi HDS 99xx and Sun StorEdge 99xx Series, and IBM ESS Series. Such arrays have no concept of primary and secondary paths.

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.

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

- Retrieve the name of the DMP device corresponding to a particular path.
- Display the members of a LUN group.
- List all paths under a DMP device.
- List all controllers connected to disks attached to the host.
- List all the paths connected to a particular controller.
- Set the attributes of the paths to an enclosure.
- Set the I/O policy that is used for the paths to an enclosure.
- Enable or disable 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 is specified by argument to 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 the Members of a LUN Group

The following command displays the DMP nodes that are in the same LUN group as a specified DMP node:

```
# vxdmpadm getlungroup dmpnodename=c11t0d10s2
```

The above command displays output such as the following:

```
NAME        STATE      ENCLR-TYPE  PATHS  ENBL   DSBL    ENCLR-NAME
===================================================================
c11t0d8s2   ENABLED    ACME        2      2      0       enc1
c11t0d9s2   ENABLED    ACME        2      2      0       enc1
c11t0d10s2  ENABLED    ACME        2      2      0       enc1
c11t0d11s2  ENABLED    ACME        2      2      0       enc1
```

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[A]  PATH-TYPE[M]CTLR-NAME ENCLR-TYPE ENCLR-NAME ATTRS
=======================================================================
c2t1d0s2    ENABLED   PRIMARY     c2         ACME        enc0       -
c3t2d0s2    ENABLED   SECONDARY   c3         ACME        enc0       -
```

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

The state of a path that is currently enabled and available for I/O is shown as `ENABLED(A)`:

```
# vxdmpadm getsubpaths dmpnodename=c2t66d0s2

NAME        STATE[A]  PATH-TYPE[M]CTLR-NAME ENCLR-TYPE ENCLR-NAME ATTRS
======================================================================
c2t66d0s2   ENABLED(A)PRIMARY     c2        ACME       enc0       -
c1t66d0s2   ENABLED   PRIMARY     c1        ACME       enc0       -
```

For A/A arrays, all enabled paths that are available for I/O are shown as `ENABLED(A)`.

For A/P arrays in which the I/O policy is set to `singleactive`, only one path is shown as `ENABLED(A)`. The other paths are enabled but not available for I/O. If the I/O policy is not set to `singleactive`, DMP can use a group of paths (all primary or all secondary) for I/O, which are shown as `ENABLED(A)`. See "Specifying the I/O Policy" on page 113 for more information.

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

```
# vxdmpadm getsubpaths ctlr=c2

NAME        STATE[A]  PATH-TYPE[M]CTLR-NAME ENCLR-TYPE ENCLR-NAME ATTRS
======================================================================
c2t1d0s2    ENABLED    PRIMARY     c2t1d0s2  ACME       enc0       -
c2t2d0s2    ENABLED    PRIMARY     c2t2d0s2  ACME       enc0       -
c2t3d0s2    ENABLED    SECONDARY   c2t3d0s2  ACME       enc0       -
c2t4d0s2    ENABLED    SECONDARY   c2t4d0s2  ACME       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              X1             ENABLED   jbod0
c2              ACME           ENABLED   enc0
c3              ACME           ENABLED   enc0
```

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

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

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

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     ARRAY_TYPE
====================================================================
enc0       T3         60020f20000001a90000 CONNECTED  A/P
```

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     ARRAY_TYPE
====================================================================
Disk        Disk      DISKS                 CONNECTED  Disk
SENA0       SENA      508002000001d660      CONNECTED  A/A
enc0        T3        60020f20000001a90000 CONNECTED  A/P
```

Displaying Information About TPD-Controlled Devices

The third-party driver (TPD) coexistence feature allows I/O that is controlled by third-party multipathing drivers to bypass DMP while retaining the monitoring capabilities of DMP. The following commands allow you to display the paths that DMP has discovered for a given TPD device, and the TPD device that corresponds to a given TPD-controlled node discovered by DMP:

```
# vxdmpadm getsubpaths tpdnodename=TPD_node_name
# vxdmpadm gettpdnode nodename=DMP_node_name
```

See “Changing Device Naming for TPD-Controlled Enclosures” on page 63 for information on how to select whether OS or TPD-based device names are displayed.

For example, consider the following disks in an EMC Symmetrix array controlled by PowerPath, which are known to DMP:

```
# vxdisk list
DEVICE         TYPE           DISK      GROUP      STATUS
emcpower10s2   auto:sliced    disk1     ppdg       online
emcpower11s2   auto:sliced    disk2     ppdg       online
emcpower12s2   auto:sliced    disk3     ppdg       online
emcpower13s2   auto:sliced    disk4     ppdg       online
emcpower14s2   auto:sliced    disk5     ppdg       online
emcpower15s2   auto:sliced    disk6     ppdg       online
emcpower16s2   auto:sliced    disk7     ppdg       online
emcpower17s2   auto:sliced    disk8     ppdg       online
emcpower18s2   auto:sliced    disk9     ppdg       online
emcpower19s2   auto:sliced    disk10    ppdg       online
```

The following command displays the paths that DMP has discovered, and which correspond to the PowerPath-controlled node, `emcpower10s2`:

```
# vxdmpadm getsubpaths tpdnodename=emcpower10s2

NAME      TPDNODENAME  PATH-TYPE[-] DMP-NODENAME ENCLR-TYPE ENCLR-NAME
======================================================================
c7t0d10s2 emcpower10s2 -            emcpower10s2 EMC        EMC0
c6t0d10s2 emcpower10s2 -            emcpower10s2 EMC        EMC0
```

Conversely, the next command displays information about the PowerPath node that corresponds to the path, `c7t0d10s2`, discovered by DMP:

```
# vxdmpadm gettpdnode nodename=c7t0d10s2
NAME           STATE     PATHS     ENCLR-TYPE     ENCLR-NAME
======================================================================
emcpower20s2   ENABLED   2         EMC            EMC0
```

Gathering and Displaying I/O Statistics

You can use the `vxdmpadm iostat` command to gather and display I/O statistics for a specified DMP node, enclosure or path.

To enable the gathering of statistics, enter this command:

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

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

```
# vxdmpadm iostat reset
```

The `memory` attribute can be used to limit the maximum amount of memory that is used to record I/O statistics for each CPU. The default limit is `32k` (32 kilobytes) per CPU.

To display the accumulated statistics at regular intervals, use the following command:

```
# vxdmpadm iostat show {all | dmpnodename=dmp-node | \
  enclosure=enclr-name | pathname=path_name}
  [interval=seconds [count=N]]
```

This command displays I/O statistics for all controllers (`all`), or for a specified DMP node, enclosure or path. The statistics displayed are the CPU usage and amount of memory per CPU used to accumulate statistics, the number of read and write operations, the number of blocks read and written, and the average time in milliseconds per read and write operation.

The `interval` and `count` attributes may be used to specify the interval in seconds between displaying the I/O statistics, and the number of lines to be displayed. The actual interval may be smaller than the value specified if insufficient memory is available to record the statistics.

To disable the gathering of statistics, enter this command:

```
# vxdmpadm iostat stop
```

Examples of Using the vxdmpadm iostat Command

The follow is an example session using the `vxdmpadm iostat` command. The first command enables the gathering of I/O statistics:

```
# vxdmpadm iostat start
```

The next command displays the current statistics including the accumulated total numbers of read and write operations and kilobytes read and written, on all paths:

```
# vxdmpadm iostat show all
                    cpu usage = 7952us    per cpu me mory = 8192b
              OPERATIONS          K  BYTES            AVG TIME(ms)
PATHNAME   READS    WR ITES   R   EADS  W   RITES  R   EADS  W RITES
c0t0d0      1088          0    557 056          0  0. 009542 0.000000
c2t118d0      87          0     44 544         0   0.001194 0.000000
c3t118d0       0    0          0          0       0 .000000 0.000000
c2t122d0      87          0     44 544         0   0.007265 0.000000
c3t122d0       0    0          0          0       0 .000000 0.000000
c2t115d0      87          0     44 544         0   0.001200 0.000000
c3t115d0       0    0          0          0       0 .000000 0.000000
c2t103d0      87          0     44 544         0   0.007315 0.000000
c3t103d0       0    0          0          0       0 .000000 0.000000
c2t102d0      87          0     44 544         0   0.001132 0.000000
c3t102d0       0    0          0          0       0 .000000 0.000000
c2t121d0      87          0     44 544         0   0.000997 0.000000
c3t121d0       0    0          0          0       0 .000000 0.000000
c2t112d0      87          0     44 544         0   0.001559 0.000000
c3t112d0       0    0          0          0       0 .000000 0.000000
c2t96d0       87         0      44 544         0   0.007057 0.000000
c3t96d0        0    0          0          0       0 .000000 0.000000
c2t106d0      87          0     44 544         0   0.007247 0.000000
c3t106d0       0    0          0          0       0 .000000 0.000000
c2t113d0      87          0     44 544         0   0.007235 0.000000
c3t113d0       0    0          0          0       0 .000000 0.000000
c2t119d0      87          0     44 544         0   0.001390 0.000000
c3t119d0       0    0          0          0       0 .000000 0.000000
```

The following command changes the amount of memory that vxdmpadm can use to accumulate the statistics:

```
# vxdmpadm iostat start memory=4096
```

The displayed statistics can be filtered by path name, DMP node name, and enclosure name (note that the per-CPU memory has changed following the previous command):

```
# vxdmpadm iostat show pathname=c3t115d0s2
                    cpu usage = 8132us    per cpu memory = 4096b
               OPERATIONS           BY  TES           AVG TIME(ms)
PATHNAME   READS    WR ITES   R   EADS  W   RITES   R   EADS  W RITES
c3t115d0s2     0    0          0          0       0 .000000 0.000000
```

```
# vxdmpadm iostat show dmpnodename=c0t0d0s2
                    cpu usage = 8501us    per cpu memory = 4096b
              OPERATIONS      B        YTES      A        VG TIME(ms)
PATHNAME   READS     WR ITES    R    EADS   W   RITES     READS  W RITES
c0t0d0s2    1088          0    5  57056          0  0.009542 0.000000
```

```
# vxdmpadm iostat show enclosure=Disk
                    cpu usage = 8626us    per cpu memory = 4096b
cpu usage = 8501us    per cpu memory = 4096b
              OPERATIONS      B        YTES      A        VG TIME(ms)
PATHNAME   READS     WR ITES    R    EADS   W   RITES     READS  W RITES
c0t0d0s2    1088          0     557 056          0  0.009542 0.000000
```

You can also specify the number of times to display the statistics and the time interval. Here the incremental statistics for a path are displayed twice with a 2-second interval:

```
# vxdmpadm iostat show pathname=c3t115d0s2 interval=2 count=2
                    cpu usage = 8195us    per cpu memory = 4096b
              OPERATIONS     B        YTES      A        VG TIME(ms)
PATHNAME   READS     WR ITES    R    EADS   W   RITES     READS  W RITES
c3t115d0s2     0     0         0          0       0 .000000 0.000000

                      cpu usage = 59us    pe r cpu memory = 4096b
              OPERATIONS     B        YTES      A        VG TIME(ms)
PATHNAME   READS     WR ITES    R    EADS   W   RITES     READS  W RITES
c3t115d0s2     0          0          0          0  0.000000 0.000000
```

Setting the Attributes of the Paths to an Enclosure

You can use the `vxdmpadm setattr` command to set the following attributes of the paths to an enclosure or disk array:

- `active`

  Changes a *standby* (failover) path to an active path. The example below specifies an active path for an A/P-C disk array:

  ```
  # vxdmpadm setattr path c2t10d0s2 pathtype=active
  ```

- `nomanual`

  Restores the original primary or secondary attributes of a path. This example restores the attributes for a path to an A/P disk array:

  ```
  # vxdmpadm setattr path c3t10d0s2 pathtype=nomanual
  ```

- `nopreferred`

  Restores the normal priority of a path. The following example restores the default priority to a path:

  ```
  # vxdmpadm setattr path c1t20d0s2 pathtype=nopreferred
  ```

- `preferred priority=`*`N`*

  Specifies a path as preferred and assigns a priority number to it. The priority number must be an integer that is greater than or equal to one. Higher priority numbers indicate that a path is able to carry a greater I/O load.

  Note Setting a priority for path does not change the I/O policy. The I/O policy must be set independently as described in "Specifying the I/O Policy" on page 113.

  This example first sets the I/O policy to `priority` for an Active/Active disk array, and then specifies a preferred path with an assigned priority of 2:

  ```
  # vxdmpadm setattr enclosure enc0 iopolicy=priority
  ```

  ```
  # vxdmpadm setattr path c1t20d0s2 pathtype=preferred \
   priority=2
  ```

- `primary`

  Defines a path as being the primary path for an Active/Passive disk array. The following example specifies a primary path for an A/P disk array:

  ```
  # vxdmpadm setattr path c3t10d0s2 pathtype=primary
  ```

- `secondary`

  Defines a path as being the secondary path for an Active/Passive disk array. This example specifies a secondary path for an A/P disk array:

  ```
  # vxdmpadm setattr path c4t10d0s2 pathtype=secondary
  ```

- `standby`

  Marks a standby (failover) path that it is not used for normal I/O scheduling. This path is used if there are no active paths available for I/O. The next example specifies a standby path for an A/P-C disk array:

  ```
  # vxdmpadm setattr path c2t10d0s2 pathtype=standby
  ```

Displaying the I/O Policy

To display the current and default settings of the I/O policy for an enclosure, array or array type, use the `vxdmpadm getattr` command.

The following example displays the default and current setting of `iopolicy` for JBOD disks:

```
# vxdmpadm getattr enclosure Disk iopolicy

ENCLR_NAME      DEFAULT      CURRENT
---------------------------------------
Disk            balanced     MinimumQ
```

The next example displays the setting of `partitionsize` for the enclosure `enc0`, on which the `balanced` I/O policy with a partition size of 2MB has been set:

```
# vxdmpadm getattr enclosure enc0 partitionsize

ENCLR_NAME      DEFAULT      CURRENT
---------------------------------------
enc0            2048         4096
```

Specifying the I/O Policy

You can use the `vxdmpadm setattr` command to change the I/O policy for distributing I/O load across multiple paths to a disk array or enclosure. You can set policies for an enclosure (for example, `HDS01`), for all enclosures of a particular type (such as `HDS`), or for all enclosures of a particular array type (`A/A` for Active/Active, or `A/P` for Active/Passive).

Note Starting with release 4.1 of VxVM, I/O policies are recorded in the file `/etc/vx/dmppolicy.info`, and are persistent across reboots of the system. Do not edit this file yourself.

The following policies may be set:

◆ `adaptive`

This policy attempts to maximize overall I/O throughput from/to the disks by dynamically scheduling I/O on the paths. It is suggested for use where I/O loads can vary over time. For example, I/O from/to a database may exhibit both long transfers (table scans) and short transfers (random look ups). The policy is also useful for a SAN environment where different paths may have different number of hops. No further configuration is possible as this policy is automatically managed by DMP.

In this example, the `adaptive` I/O policy is set for the enclosure `enc1`:

```
# vxdmpadm setattr enclosure enc1 iopolicy=adaptive
```

- `balanced [partitionsize=`*`size`*`]`

  This policy is designed to optimize the use of caching in disk drives and RAID controllers, and is the default policy for A/A arrays. The size of the cache typically ranges from 120KB to 500KB or more, depending on the characteristics of the particular hardware. During normal operation, the disks (or LUNs) are logically divided into a number of regions (or *partitions*), and I/O from/to a given region is sent on only one of the active paths. Should that path fail, the workload is automatically redistributed across the remaining paths.

  You can use the *size* argument to the `partitionsize` attribute to specify the partition size. The partition size in blocks is adjustable in powers of 2 from 2 up to 2^31 as illustrated in the table below.

Partition Size in Blocks	**Equivalent Size in Bytes**
2	1,024
4	2,048
8	4,096
16	8,192
32	16,384
64	32,768
128	65,536
256	131,072
512	262,144
1024	524,288
2048 (default)	1,048,576
4096	2,097,152

  The default value for the partition size is 2048 blocks (1MB). A value that is not a power of 2 is silently rounded down to the nearest acceptable value. Specifying a partition size of 0 is equivalent to the default partition size of 2048 blocks (1MB). For example, the suggested partition size for an Hitachi HDS 9960 A/A array is from 32,768 to 131,072 blocks (16MB to 64MB) for an I/O activity pattern that consists mostly of sequential reads or writes.

Note The benefit of this policy is lost if the value is set larger than the cache size.

The default value can be changed by adjusting the value of a tunable parameter (see "dmp_pathswitch_blks_shift" on page 292) and rebooting the system.

The next example sets the `balanced` I/O policy with a partition size of 4096 blocks (2MB) on the enclosure `enc0`:

```
# vxdmpadm setattr enclosure enc0 iopolicy=balanced \
  partitionsize=4096
```

- `minimumq`

  This policy sends I/O on paths that have the minimum number of outstanding I/O requests in the queue for a LUN. This is suitable for low-end disks or JBODs where a significant track cache does not exist. No further configuration is possible as DMP automatically determines the path with the shortest queue.

  The following example sets the I/O policy to `minimumq` for a JBOD:

  ```
  # vxdmpadm setattr enclosure Disk iopolicy=minimumq
  ```

- `priority`

  This policy is useful when the paths in a SAN have unequal performance, and you want to enforce load balancing manually. You can assign priorities to each path based on your knowledge of the configuration and performance characteristics of the available paths, and of other aspects of your system. See "Setting the Attributes of the Paths to an Enclosure" on page 111 for details of how to assign priority values to individual paths.

  In this example, the I/O policy is set to `priority` for all SENA arrays:

  ```
  # vxdmpadm setattr arrayname SENA iopolicy=priority
  ```

- `round-robin`

  This policy shares I/O equally between the paths in a round-robin sequence. For example, if there are three paths, the first I/O request would use one path, the second would use a different path, the third would be sent down the remaining path, the fourth would go down the first path, and so on. No further configuration is possible as this policy is automatically managed by DMP. This is the default policy for A/PC configurations with multiple active paths per controller.

  The next example sets the I/O policy to `round-robin` for all Active/Active arrays:

  ```
  # vxdmpadm setattr arraytype A/A iopolicy=round-robin
  ```

- `singleactive`

  This policy routes I/O down one single active path. This is the default policy for A/P arrays with one active path per controller, where the other paths are used in case of failover. If configured for A/A arrays, there is no load balancing across the paths, and the alternate paths are only used to provide high availability (HA). If the currently active path fails, I/O is switched to an alternate active path. No further configuration is possible as the single active path is selected by DMP.

  The following example sets the I/O policy to `singleactive` for JBOD disks:

  ```
  # vxdmpadm setattr arrayname DISK iopolicy=singleactive
  ```

Example of Applying Load Balancing in a SAN

This example describes how to configure load balancing in a SAN environment where there are multiple primary paths to an Active/Passive device through several SAN switches. As can be seen in this sample output from the `vxdisk list` command, the device `c3t2d15s2` has eight primary paths:

```
# vxdisk list c3t2d15s2

Device: c3t2d15s2
...
numpaths: 8
c2t0d15s2 state=enabled type=primary
c2t1d15s2 state=enabled type=primary
c3t1d15s2 state=enabled type=primary
c3t2d15s2 state=enabled type=primary
c4t2d15s2 state=enabled type=primary
c4t3d15s2 state=enabled type=primary
c5t3d15s2 state=enabled type=primary
c5t4d15s2 state=enabled type=primary
```

In addition, the device is in the enclosure `ENC0`, belongs to the disk group `mydg`, and contains a simple concatenated volume `myvol1`.

The first step is to enable the gathering of DMP statistics:

```
# vxdmpadm iostat start
```

Next the `dd` command is used to apply an input workload from the volume:

```
# dd if=/dev/vx/rdsk/mydg/myvol1 of=/dev/null &
```

By running the `vxdmpadm iostat` command to display the DMP statistics for the device, it can be seen that all I/O is being directed to one path, `c5t4d15s2`:

```
# vxdmpadm iostat show dmpnodename=c3t2d15s2 interval=5 count=2
...
cpu usage = 11294us per cpu memory = 32768b
             OPERATIONS        KBYTES          AVG TIME(ms)
PATHNAME  READS  WRITES   READS    WRITES    READS     WRITES
c2t0d15s2 0      0        0        0         0.000000  0.000000
c2t1d15s2 0      0        0        0         0.000000  0.000000
c3t1d15s2 0      0        0        0         0.000000  0.000000
c3t2d15s2 0      0        0        0         0.000000  0.000000
c4t2d15s2 0      0        0        0         0.000000  0.000000
c4t3d15s2 0      0        0        0         0.000000  0.000000
c5t3d15s2 0      0        0        0         0.000000  0.000000
c5t4d15s2 10986  0        5493     0         0.411069  0.000000
```

The `vxdmpadm` command is used to display the I/O policy for the enclosure that contains the device:

```
# vxdmpadm getattr enclosure ENC0 iopolicy

ENCLR_NAME     DEFAULT        CURRENT
============================================
ENC0           Single-Active  Single-Active
```

This shows that the policy for the enclosure is set to `singleactive`, which explains why all the I/O is taking place on one path.

To balance the I/O load across the multiple primary paths, the policy is set to `round-robin` as shown here:

```
# vxdmpadm setattr enclosure ENC0 iopolicy=round-robin
# vxdmpadm getattr enclosure ENC0 iopolicy

ENCLR_NAME     DEFAULT        CURRENT
============================================
ENC0           Single-Active  Round-Robin
```

The DMP statistics are now reset:

```
# vxdmpadm iostat reset
```

With the workload still running, the effect of changing the I/O policy to balance the load across the primary paths can now be seen.

```
# vxdmpadm iostat show dmpnodename=c3t2d15s2 interval=5 count=2
...
cpu usage = 14403us per cpu memory = 32768b
             OPERATIONS        KBYTES         AVG TIME(ms)
PATHNAME  READS  WRITES   READS    WRITES    READS     WRITES
c2t0d15s2 2041   0        1021     0         0.396670  0.000000
c2t1d15s2 1894   0        947      0         0.391763  0.000000
c3t1d15s2 2008   0        1004     0         0.393426  0.000000
c3t2d15s2 2054   0        1027     0         0.402142  0.000000
c4t2d15s2 2171   0        1086     0         0.390424  0.000000
c4t3d15s2 2095   0        1048     0         0.391221  0.000000
c5t3d15s2 2073   0        1036     0         0.390927  0.000000
c5t4d15s2 2042   0        1021     0         0.392752  0.000000
```

The enclosure can be returned to the single active I/O policy by entering the following command:

```
# vxdmpadm setattr enclosure ENC0 iopolicy=singleactive
```

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 [-f] 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
```

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 enc`0` to `GRP1`.

Note The maximum length of the enclosure name prefix is 25 characters. The name must not contain an underbar character (_).

The following command shows the changed name.

```
# vxdmpadm listenclosure all
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
```

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

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
```

Configuring Array Policy Modules

An array policy module (APM) is a dynamically loadable kernel module that may be provided by some vendors for use in conjunction with an array. An APM defines procedures to:

- Select an I/O path when multiple paths to a disk within the array are available.
- Select the path failover mechanism.
- Select the alternate path in the case of a path failure.
- Put a path change into effect.
- Respond to SCSI reservation or release requests.

DMP supplies default procedures for these functions when an array is registered. An APM may modify some or all of the existing procedures that are provided by DMP or by another version of the APM.

You can use the following command to display all the APMs that are configured for a system:

```
# vxdmpadm listapm all
```

The output from this command includes the file name of each module, the supported array type, the APM name, the APM version, and whether the module is currently in use (*loaded*). To see detailed information for an individual module, specify the module name as the argument to the command:

```
# vxdmpadm listapm module_name
```

To add and configure an APM, use the following command:

```
# vxdmpadm -a cfgapm module_name [attr1=value1 [attr2=value2 ...]]
```

The optional configuration attributes and their values are specific to the APM for an array. Consult the documentation that is provided by the array vendor for details.

Note By default, DMP uses the most recent APM that is available. Specify the `-u` option instead of the `-a` option if you want to force DMP to use an earlier version of the APM. The current version of an APM is replaced only if it is not in use.

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

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

For more information about configuring APMs, see the `vxdmpadm`(1M) manual page.

Creating and Administering Disk Groups 4

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.

Note For a discussion of disk groups that are compatible with the Cross-platform Data Sharing (CDS) feature of VERITAS Volume Manager, see the *VERITAS Storage Foundation Cross-Platform Data Sharing Administrator's Guide.* The CDS feature allows you to move VxVM disks and objects between machines that are running under different operating systems.

The disk group split and join feature is not supported in this release.

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 can be initialized, reserved, and added to disk groups at any time. Disks need not be added to disk groups until the disks are needed to create VxVM objects.

When a disk is added to a disk group, it is given a name (for example, `mydg02`). This name identifies a disk for operations such as volume creation or mirroring. The name also relates directly to the underlying physical disk. If a physical disk is moved to a different target address or to a different controller, the name `mydg02` continues to refer to it. 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 disk groups that contain many disks and VxVM objects causes the private region to fill. In the case of large disk groups that are expected to contain more than several hundred disks and VxVM objects, 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 126.

For information on backing up and restoring disk group configurations, see “Backing Up and Restoring Disk Group Configuration Data” on page 147.

Specifying a Disk Group to Commands

Note Most VxVM commands require superuser or equivalent privileges.

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

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

The block and character special devices corresponding to this volume are:

`/dev/vx/dsk/mktdg/mktvol`	Block device
`/dev/vx/rdsk/mktdg/mktvol`	Character device

System-Wide Reserved Disk Groups

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

`bootdg` Specifes the boot disk group. This is an alias for the disk group that contains the volumes that are used to boot the system. VxVM sets `bootdg` to the appropriate disk group if it takes control of the root disk. Otherwise, `bootdg` is set to `nodg` (no disk group; see below).

Caution Do not attempt to change the assigned value of `bootdg`. Doing so may render your system unbootable.

`defaultdg` Specifies the default disk group. This is an alias for the disk group name that should be assumed if the `-g` option is not specified to a command, or if the `VXVM_DEFAULTDG` environment variable is undefined. By default, `defaultdg` is set to `nodg` (no disk group; see below).

`nodg` Specifies to an operation that no disk group has been defined. For example, if the root disk is not under VxVM control, `bootdg` is set to `nodg`.

Note There is no requirement that `defaultdg` and `bootdg` refer to the same disk group.

Rules for Determining the Default Disk Group

It is recommended that you use the `-g` option to specify a disk group to VxVM commands that accept this option. If you do not specify the disk group, VxVM applies the following rules in order until it determines a disk group name:

1. Use the default disk group name that is specified by the environment variable `VXVM_DEFAULTDG`. This variable can also be set to one of the reserved system-wide disk group names: `bootdg`, `defaultdg`, or `nodg`. If the variable is undefined, the following rule is applied.

2. Use the disk group that has been assigned to the system-wide default disk group alias, `defaultdg`. See "Displaying and Specifying the System-Wide Default Disk Group" on page 125. If this alias is undefined, the following rule is applied.

3. If the operation can be performed without requiring a disk group name (for example, an edit operation on disk access records), do so.

If none of these rules succeeds, the requested operation fails.

Displaying the System-Wide Boot Disk Group

To display the currently defined system-wide boot disk group, use the following command:

```
# vxdg bootdg
```

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

Displaying and Specifying the System-Wide Default Disk Group

To display the currently defined system-wide default disk group, use the following command:

```
# vxdg defaultdg
```

If a default disk group has not been defined, `nodg` is displayed. Alternatively, you can use the following command to display the default disk group:

```
# vxprint -Gng defaultdg 2>/dev/null
```

In this case, if there is no default disk group, nothing is displayed.

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

```
# vxdctl defaultdg diskgroup
```

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

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

Note The specified *diskgroup* need not currently exist on the system.

See the `vxdctl`(1M) and `vxdg`(1M) manual pages for more information.

Displaying Disk Group Information

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

```
# vxdg list
NAME    STATE     ID
rootdg  enabled   730344554.1025.tweety
newdg   enabled   731118794.1213.tweety
```

To display more detailed information on a specific disk group, use the following command:

```
# vxdg list diskgroup
```

The output from this command is similar to the following:

```
Group: mydg
dgid: 962910960.1025.bass
import-id: 0.1
flags:
version: 110
local-activation: read-write
alignment : 512 (bytes)
ssb: on
detach-policy: local
copies: nconfig=default nlog=default
config: seqno=0.1183 permlen=3448 free=3428 templen=12 loglen=522
config disk c0t10d0 copy 1 len=3448 state=clean online
config disk c0t11d0 copy 1 len=3448 state=clean online
log disk c0t10d0 copy 1 len=522
log disk c0t11d0 copy 1 len=522
```

Note In this example, the administrator has chosen to name the boot disk group as `rootdg`.

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:  s imple
flags:  o nline ready private autoconfig autoimport imported
diskid: 963504891.1070.bass
dgname: newdg
dgid:  9 63504895.1075.bass
hostid: bass
info:  p rivoffset=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
mydg   mydg01    c0t10d0      c0t10d0      0       4444228 -
mydg   mydg02    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 disk group, `mydg`, use the following command:

```
# vxdg -g mydg free
```

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

```
DISK     DEVICE       TAG          OFFSET  LENGTH    FLAGS
mydg01   c0t10d0      c0t10d0      0       4444228   -
mydg02   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.

Disks must be placed in one or more disk groups before VxVM can use the disks for volumes. It is important that you locate data related to particular applications or users on an identifiable set of disks. When you need to move these disks, this allows you to move only the application or user data that should be moved.

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 Disks to VxVM" on page 68. The disks to be added to a disk group must not belong to an existing disk group.

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

```
# vxdiskadd c1t0d0
```

where `c1t0d0` in this example is the device name of a disk that is not currently assigned to a disk group. The command dialog is identical to that described for the `vxdiskadm` command starting at step 3 on page 70.

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

```
# vxdg init diskgroup [cds=on|off] 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, `c1t0d0`**`s2`**, must have been previously initialized with `vxdiskadd` or `vxdiskadm`, and must not currently belong to a disk group.

You can use the `cds` attribute with the `vxdg init` command to specify whether a new disk group is compatible with the Cross-platform Data Sharing (CDS) feature. In VERITAS Volume Manager 4.0 and later releases, newly created disk groups are compatible with CDS by default (equivalent to specifying `cds=on`). If you want to change this behavior, edit the file `/etc/default/vxdg`, and set the attribute-value pair `cds=off` in this file before creating a new disk group.

Alternatively, you can use the following command to set this attribute for a disk group:

```
# vxdg -g diskgroup set cds=on|off
```

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. For details of this procedure, see "Adding Disks to VxVM" on page 68.

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

```
# vxdiskadd c1t1d0
```

where `c1t1d0` is the device name of a disk that is not currently assigned to a disk group. The command dialog is identical to that described for the `vxdiskadm` command starting at step 3 on page 70.

Removing a Disk from a Disk Group

Note Before you can remove the last disk from a disk group, you must disable the disk group as described in "Disabling a Disk Group" on page 140. Alternatively, you can destroy the disk group as described in "Destroying a Disk Group" on page 141.

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

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

For example, to remove `mydg02` from the disk group, `mydg`, use this command:

```
# vxdg -g mydg rmdisk mydg02
```

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

```
VxVM vxdg ERROR V-5-1-552 Disk diskname is used by one or more
subdisks
Use -k to remove device assignment.
```

Using the `-k` option 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:

```
# vxdiskunsetup devicename
```

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

```
# vxdiskunsetup 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 volumes, this message is displayed:

```
VxVM ERROR V-5-2-369 The following volumes currently use part of disk
mydg02:

   home usrvol

   Volumes must be moved from mydg02 before it can be removed.

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

If you choose **`y`**, then all volumes are moved off the disk, if possible. Some volumes may not be movable. The most common reasons why a volume may not be movable are as follows:

- There is not enough space on the remaining disks.
- Plexes or striped subdisks cannot be allocated on different disks from existing plexes or striped subdisks in the volume.

If `vxdiskadm` cannot move some volumes, you may need to remove some plexes from some disks to free more space before proceeding with the disk removal operation.

Deporting a Disk Group

Deporting a disk group disables access to a disk group that is 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.

   Note Deportation fails if the disk group contains volumes that are in use (for example, by mounted file systems or databases).

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:

```
VxVM INFO V-5-2-377 The requested operation is to disable access
to the removable disk group named newdg. This disk group is
stored on the following disks:
   n ewdg01 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:

```
VxVM INFO V-5-2-269 Removal of disk group newdg was successful.
```

7. At the following prompt, indicate whether you want to disable another disk group (`y`) or return to the `vxdiskadm` main menu (`n`):

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

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
```

If the disks are not in the online state, use menu item `10 (Enable (online) a disk device)` from the `vxdiskadm` main menu to bring them online:

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:

```
VxVM INFO V-5-2-374 The import of newdg was successful.
```

4. At the following prompt, indicate whether you want to import another disk group (`y`) or return to the `vxdiskadm` main menu (`n`):

```
Select another disk group? [y,n,q,?] (default: n)
```

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.

To rename a disk group during import, use the following command:

```
# vxdg [-t] -n newdg import diskgroup
```

If the `-t` option is included, the import is temporary and does not persist across reboots. In this case, the stored name of the disk group remains unchanged on its original host, but the disk group is known by the name specified by *newdg* to the importing host. If the `-t` option is not used, the name change is permanent.

For example, this command temporarily renames the disk group, `mydg`, as `mytempdg` on import:

```
# vxdg -t -n mytempdg import mydg
```

To rename a disk group during deport, use the following command:

```
# vxdg [-h hostname] -n newdg deport diskgroup
```

When renaming on deport, you can specify the `-h` *hostname* option to assign a lock to an alternate host. This ensures that the disk group is automatically imported when the alternate host reboots.

For example, this command renames the disk group, `mydg`, as `myexdg`, and deports it to the host, `jingo`:

```
# vxdg -h jingo -n myexdg deport mydg
```

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 `salesdg04`) from disk group `salesdg` and add it to disk group `mktdg`, use the following commands:

```
# vxdg -g salesdg rmdisk salesdg04
# vxdg -g mktdg adddisk mktdg02=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)`.

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 access the same disks at the same time, this must be managed using software such as the clustering functionality of VxVM. Otherwise, configuration information stored on the disk may be corrupted, and the data on the disk may become unusable.

Handling Errors when Importing Disks

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

```
VxVM vxdg ERROR V-5-1-587 disk group groupname: import failed:
Disk is in use by another host
```

The next message indicates that the disk group does not contains any valid disks (not that it does not contains any disks):

```
VxVM vxdg ERROR V-5-1-587 Disk group groupname: import failed:
No valid disk found containing disk group
```

The disks may be considered invalid due to a mismatch between the host ID in their configuration copies and that stored in the `/etc/vx/volboot` file.

To clear locks on a specific set of devices, use the following command:

```
# vxdisk clearimport devicename ...
```

To clear the locks during import, use the following command:

```
# vxdg -C import diskgroup
```

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:

```
VxVM vxdg ERROR V-5-1-587 Disk group groupname: import failed:
Disk group has no valid configuration copies
```

The following message indicates a recoverable error.

```
VxVM vxdg ERROR V-5-1-587 Disk group groupname: import failed:
Disk for disk group not found
```

If some of the disks in the disk group have failed, 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.

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

VxVM reserves the range of minor numbers from 0 to 999 for use with volumes in the boot disk group. For example, the `rootvol` volume is always assigned minor number 0.

If you do not specify the base of the minor number range for a disk group, VxVM chooses one at random. The number chosen is at least 1000, is a multiple of 1000, and yields a usable range of 1000 device numbers. The chosen number also does not overlap within a range of 1000 of any currently imported disk groups, and it does not overlap any currently allocated volume device numbers.

Note The default policy ensures that a small number of disk groups can be merged successfully between a set of machines. However, where disk groups are merged automatically using failover mechanisms, select ranges that avoid overlap.

To view the base minor number for an existing disk group, use the `vxprint` command as shown in the following examples for the disk group, `mydg`:

```
# vxprint -l mydg | egrep minors
minors: >=46000
# vxprint -g mydg -m | egrep base_minor
base_minor=46000
```

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

```
# vxdg init diskgroup minor=base_minor disk_access_name ...
```

For example, the following command creates the disk group, `newdg`, that includes the specified disks, and has a base minor number of 30000:

```
# vxdg init newdg minor=30000 c1d0t0s2 c1t1d0s2
```

If a disk group already exists, you can use the `vxdg reminor` command to change its base minor number:

```
# vxdg -g diskgroup reminor new_base_minor
```

For example, the following command changes the base minor number to 30000 for the disk group, `mydg`:

```
# vxprint -g mydg reminor 30000
```

If a volume is open, its old device number remains in effect until the system is rebooted or until the disk group is deported and re-imported. If you close the open volume, you can run `vxdg reminor` again to allow the renumbering to take effect without rebooting or re-importing.

An example of where it is necessary to change the base minor number is for a cluster-shareable disk group. The volumes in a shared disk group must have the same minor number on all the nodes. If there is a conflict between the minor numbers when a node attempts to join the cluster, the join fails. You can use the `reminor` operation on the nodes that are in the cluster to resolve the conflict. In a cluster where more than one node is joined, use a base minor number which does not conflict on any node.

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

Compatibility of Disk Groups Between Platforms

For disk groups that support the Cross-platform Data Sharing (CDS) feature, the upper limit on the minor number range is restricted on AIX, HP-UX, Linux (with a 2.6 or later kernel) and Solaris to 65,535 to ensure portability between these operating systems.

On a Linux platform with a pre-2.6 kernel, the number of minor numbers per major number is limited to 255 with a base of 0. This has the effect of limiting the number of volumes that can be supported system-wide to a smaller value than is allowed on other operating system platforms. With the extended major numbering scheme used in VxVM 4.0 on Linux, the maximum number of volumes was limited to 4079 provided that a contiguous block of 15 extended major numbers is available.

The number of disks that can be supported is limited by the Linux pre-2.6 kernel (typically to a few hundred disks), but not by the dynamic multipathing (DMP) feature of VERITAS Volume Manager. DMP in VxVM 4.0 and later releases on Linux can support more than 4 billion disk devices with 16 partitions per disk. VxVM 4.1 runs on a 2.6 version Linux kernel, which allows an almost unlimited number of volumes and disk devices to be configured on a system (although the number of volumes that can be configured in a disk group is still limited by the size of the private region).

When a CDS-compatible disk group is imported on a Linux system with a pre-2.6 kernel, VxVM attempts to reassign the minor numbers of the volumes, and fails if this is not possible.

To help ensure that a CDS-compatible disk group is portable between operating systems, including Linux with a pre-2.6 kernel, use the following command to set the `maxdev` attribute on the disk group:

```
# vxdg -g diskgroup set maxdev=4079
```

Note Such a disk group may still not be importable by VxVM 4.0 on Linux with a pre-2.6 kernel if it would increase the number of minor numbers on the system that are assigned to volumes to more than 4079, or if the number of available extended major numbers is smaller than 15.

You can use the following command to discover the maximum number of volumes that are supported by VxVM on a Linux host:

```
# cat /proc/sys/vxvm/vxio/vol_max_volumes
4079
```

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

Handling Conflicting Configuration Copies in a Disk Group

If an incomplete disk group is imported on several different systems, this can create inconsistencies in the disk group configuration copies that you may need to resolve manually. The following section describe how to correct such a condition.

Correcting Conflicting Configuration Information

Note This procedure requires that the disk group has a version number of at least 120.

To resolve conflicting configuration information, you must decide which disk contains the correct version of the disk group configuration database. To assist you in doing this, you can run the `vxsplitlines` command to show the actual serial ID on each disk in the disk group and the serial ID that was expected from the configuration database. For each disk, the command also shows the `vxdg` command that you must run to select the configuration database copy on that disk as being the definitive copy to use for importing the disk group.

The following is sample output from running `vxsplitlines` on the disk group `newdg`:

```
# vxsplitlines -g newdg
The following splits were found in disk group newdg
They are listed in da(dm) name pairs.

Pool 0.
  c2t5d0s2 ( c2t5d0s2 ), c2t6d0s2 ( c2t6d0s2 ),
The configuration from any of the disks in this split should appear
to be be the same.
To see the configuration from any of the disks in this split, run:
  /etc/vx/diag.d/vxprivutil dumpconfig /dev/vx/dmp/c2t5d0s2
To import the dg with the configuration from this split, run:
  /usr/sbin/vxdg -o selectcp=1045852127.32.olancha import newdg
To get more information about this particular configuration, run:
  /usr/sbin/vxsplitlines -g newdg -c c2t5d0s2

Split 1.
c2t7d0s2 ( c2t7d0s2 ), c2t8d0s2 ( c2t8d0s2 ),
The configuration from any of the disks in this split should appear
to be be the same.
To see the configuration from any of the disks in this split, run:
  /etc/vx/diag.d/vxprivutil dumpconfig /dev/vx/dmp/c2t7d0s2
```

In this example, the disk group has four disks, and is split so that two disks appear to be on each side of the split.

You can specify the `-c` option to `vxsplitlines` to print detailed information about each of the disk IDs from the configuration copy on a disk specified by its disk access name:

```
# vxsplitlines  - g newdg -c c2t6d0s2
DANAME(DMNAME)          || Actual SSB  || Expected SSB
c2t5d0s2( c2t5d0s2 ) || 0.1            || 0.0 ssb ids don't match
c2t6d0s2( c2t6d0s2 ) || 0.1            || 0.1 ssb ids match
c2t7d0s2( c2t7d0s2 ) || 0.1            || 0.1 ssb ids match
c2t8d0s2( c2t8d0s2 ) || 0.1            || 0.0 ssb ids don't match

Please note that even though some disks ssb ids might match
that does not necessarily mean that those disks' config copies
have all the changes. From some other configuration copies, those
disks' ssb ids might not match.
To see the configuration from this disk, run
/etc/vx/diag.d/vxprivutil dumpconfig /dev/vx/dmp/c2t6d0s2
```

Based on your knowledge of how the serial split brain condition came about, you must choose one disk's configuration to be used to import the disk group. For example, the following command imports the disk group using the configuration copy that is on side 0 of the split:

```
# /usr/sbin/vxdg -o selectcp=1045852127.32.olancha import newdg
```

When you have selected a preferred configuration copy, and the disk group has been imported, VxVM resets the serial IDs to 0 for the imported disks. The actual and expected serial IDs for any disks in the disk group that are not imported at this time remain unaltered.

Disabling a Disk Group

To disable a disk group, unmount and stop any volumes in the disk group, and then use the following command to deport it:

```
# vxdg deport diskgroup
```

Deporting a disk group does not actually remove the disk group. It disables use of the disk group by the system. Disks in a deported disk group can be reused, reinitialized, added to other disk groups, or imported for use on other systems. Use the `vxdg import` command to re-enable access to the disk group.

Destroying a Disk Group

The `vxdg` command provides a `destroy` option that removes a disk group from the system and frees the disks in that disk group for reinitialization:

```
# vxdg destroy diskgroup
```

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.

Recovering a Destroyed Disk Group

If a disk group has been accidentally destroyed, you can use the following procedure to recover it, provided that the disks that were in the disk group have not been modified or reused elsewhere:

1. Enter the following command to find out the disk group ID (*dgid*) of one of the disks that was in the disk group:

   ```
   # vxdisk -s list disk_access_name
   ```

   The disk must be specified by its disk access name, such as `c0t12d0`. Examine the output from the command for a line similar to the following that specifies the disk group ID.

   ```
   dgid:  9 63504895.1075.bass
   ```

2. Use the disk group ID to import the disk group:

   ```
   # vxdg import dgid
   ```

Upgrading a Disk Group

Note The first release of VERITAS Volume Manager on this platform is 4.1, for which the default disk group version number is 120. All CDS disk groups on all 4.1 platforms must also necessarily have a disk group version number of 120. For these reasons, the details presented here are for information only. In addition, not all of the features that are supported by version 120 disk groups are supported on this platform.

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 V-5-1-2829 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
4.0	110	20-110
4.1	120	20-120

Importing the disk group of a previous version on a VERITAS Volume Manager 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 120:

Features Supported by Disk Group Versions

Disk Group Version	New Features Supported	Previous Version Features Supported
120	◆ Automatic Cluster-wide Failback for A/P arrays ◆ DMP Co-existence with Third-Party Drivers ◆ Migration of Volumes to ISP ◆ Persistent DMP Policies ◆ Shared Disk Group Failure Policy	20, 30, 40, 50, 60, 70, 80, 90, 110
110	◆ Cross-platform Data Sharing (CDS) ◆ Device Discovery Layer (DDL) 2.0 ◆ Disk Group Configuration Backup and Restore ◆ Elimination of `rootdg` as a Special Disk Group ◆ Full-Sized and Space-Optimized Instant Snapshots ◆ Intelligent Storage Provisioning (ISP) ◆ Serial Split Brain Detection ◆ Volume Sets (Multiple Device Support for VxFS)	20, 30, 40, 50, 60, 70, 80, 90
90	◆ Cluster Support for Oracle Resilvering ◆ Disk Group Move, Split and Join ◆ Device Discovery Layer (DDL) 1.0 ◆ Layered Volume Support in Clusters ◆ Ordered Allocation ◆ OS Independent Naming Support ◆ Persistent FastResync	20, 30, 40, 50, 60, 70, 80
80	◆ VERITAS Volume Replicator (VVR) Enhancements	20, 30, 40, 50, 60, 70
70	◆ Non-Persistent FastResync ◆ Sequential DRL ◆ Unrelocate ◆ VERITAS Volume Replicator (VVR) Enhancements	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

Features Supported by Disk Group Versions

Disk Group Version	New Features Supported	Previous Version Features Supported
20	◆ Dirty Region Logging (DRL) ◆ 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 4.1 creates disk groups with version 120.

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 4.1 is 120. Such a disk group cannot be imported on a system running VERITAS Volume Manager 2.3, as that release only supports up to version 40. Therefore, to create a disk group on a system running VERITAS Volume Manager 4.1 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 command-line interface to the `vxconfigd` daemon.

You can use `vxdctl` to:

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

Note In VxVM 4.0 and later releases, disk access records are no longer stored in the `/etc/vx/volboot` file. Non-persistent disk access records are created by scanning the disks at system startup. Persistent disk access records for `simple` and `nopriv` disks are permanently stored in the `/etc/vx/darecs` file in the `root` file system. The `vxconfigd` daemon reads the contents of this file to locate the disks and the configuration databases for their disk groups. (The `/etc/vx/darecs` file is also used to store definitions of foreign devices that are not autoconfigurable. Such entries may be added by using the `vxddladm addforeign` command. See the `vxddladm`(1M) manual page for more information.)

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.

Backing Up and Restoring Disk Group Configuration Data

The disk group configuration backup and restoration feature allows you to back up and restore all configuration data for disk groups, and for VxVM objects such as volumes that are configured within the disk groups. The `vxconfigbackupd` daemon monitors changes to the VxVM configuration and automatically records any configuration changes that occur. Two utilities, `vxconfigbackup` and `vxconfigrestore`, are provided for backing up and restoring a VxVM configuration for a disk group.

For information on backing up and restoring disk group configurations, see the “Backing Up and Restoring Disk Group Configurations” chapter in the *VERITAS Volume Manager Troubleshooting Guide*, and the `vxconfigbackup`(1M) and `vxconfigrestore`(1M) manual pages.

Using vxnotify to Monitor Configuration Changes

You can use the `vxnotify` utility to display events relating to disk and configuration changes that are managed by the `vxconfigd` configuration daemon. The `vxnotify` utility displays the requested event types until you kill it, until it has received a specified number of events, or until a specified period of time has elapsed.

Examples of configuration events that can be detected include disabling and enabling of controllers, paths and DMP nodes, RAID-5 volumes entering degraded mode, and detachment of disks, plexes and volumes.

For example, the following `vxnotify` command displays information about all disk, plex, and volume detachments as they occur:

```
# vxnotify -f
```

For more information about the `vxnotify` utility, and the types of configuration events that it can report, see the `vxnotify`(1M) manual page.

Creating and Administering Subdisks 5

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 171.

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

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

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

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

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

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

If you intend to use the new subdisk to build a volume, you must associate the subdisk with a plex (see "Associating Subdisks with Plexes" on page 152). 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    D ISKOFFS LENGTH [COL/]OFF DEVICE   M ODE
SV NAME      PLEX     VOLNAME NVOLLAYR LENGTH [COL/]OFF AM/NM    M ODE
sd mydg01-01 vol1-01  mydg01  0        102400 0     c0t10d0  ENA
sd mydg02-01 vol2-01  mydg02  0        102400 0     c0t11d0  ENA
```

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

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

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

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

This command provides the following output:

```
Disk group: mydg

Subdisk:  mydg02-01
info:     disk=mydg02 offset=0 len=205632
assoc:    vol=mvol plex=mvol-02 (offset=0)
flags:    enabled
device:   d evice=c0t11d0s2 path=/dev/vx/dmp/c0t11d0s2
diskdev=32/68
```

Moving Subdisks

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

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

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

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

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

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

See "Configuring Hot-Relocation to Use Only Spare Disks" on page 253 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 [-g diskgroup] -s size split subdisk newsd1 newsd2
```

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

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

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

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

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

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

Joining Subdisks

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

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

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

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

Associating Subdisks with Plexes

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

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

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

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

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

Subdisks are associated in order starting at offset 0. If you use this type of command, you do not have to specify the multiple commands needed to create the plex and then associate each of the subdisks with that plex. In this example, the subdisks are associated to the plex in the order they are listed (after `sd=`). The disk space defined as `mydg02-01` is first, `mydg02-00` is second, and `mydg02-02` is third. This method of associating subdisks is convenient during initial configuration.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Associating Log Subdisks

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

For a description of DRL, see “Dirty Region Logging (DRL)” on page 43. 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 [-g diskgroup] aslog plex subdisk
```

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

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

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

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

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

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

Dissociating Subdisks from Plexes

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

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

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

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

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

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

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 [-g diskgroup] rm subdisk
```

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

```
# vxedit -g mydg rm mydg02-01
```

Changing Subdisk Attributes

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 [-g diskgroup] set attribute=value ... subdisk ...
```

Subdisk fields that can be changed using the `vxedit` command include:

- `name`
- `putiln`
- `tutiln`
- `len`
- `comment`

The `putiln` field attributes are maintained on reboot; `tutiln` 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 `mydg02-01` in the disk group, `mydg`, use the following command:

```
# vxedit -g mydg set comment="subdisk comment" mydg02-01
```

To prevent a particular subdisk from being associated with a plex, set the `putil0` field to a non-null string, as shown in the following command:

```
# vxedit -g mydg set putil0="DO-NOT-USE" mydg02-01
```

See the `vxedit`(1M) manual page for more information about using the `vxedit` command to change the attribute fields of VxVM objects.

Creating and Administering Plexes 6

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 171.

Use the `vxmake` command to create VxVM objects, such as plexes. When creating a plex, identify the subdisks that are to be associated with it:

To create a plex from existing subdisks, use the following command:

```
# vxmake [-g diskgroup] plex plex sd=subdisk1[,subdisk2,...]
```

For example, to create a concatenated plex named `vol01-02` from two existing subdisks named `mydg02-01` and `mydg02-02` in the disk group, `mydg`, use the following command:

```
# vxmake -g mydg plex vol01-02 sd=mydg02-01,mydg02-02
```

Creating a Striped Plex

To create a striped plex, you must specify additional attributes. For example, to create a striped plex named `pl-01` in the disk group, `mydg`, with a stripe width of 32 sectors and 2 columns, use the following command:

```
# vxmake -g mydg plex pl-01 layout=stripe stwidth=32 ncolumn=2 \
sd=mydg01-01,mydg02-01
```

To use a plex to build a volume, you must associate the plex with the volume. For more information, see the section, "Attaching and Associating Plexes" on page 163.

Displaying Plex Information

Listing plexes helps identify free plexes for building volumes. Use the plex (`-p`) option to the `vxprint` command to list information about all plexes.

To display detailed information about all plexes in the system, use the following command:

```
# vxprint -lp
```

To display detailed information about a specific plex, use the following command:

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

The `-t` option prints a single line of information about the plex. To list free plexes, use the following command:

```
# vxprint -pt
```

The following section describes the meaning of the various plex states that may be displayed in the STATE field of `vxprint` output.

Plex States

Plex states reflect whether or not plexes are complete and are consistent copies (mirrors) of the volume contents. VxVM utilities automatically maintain the plex state. However, if a volume should not be written to because there are changes to that volume and if a plex is associated with that volume, you can modify the state of the plex. For example, if a disk with a particular plex located on it begins to fail, you can temporarily disable that plex.

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 was found for one of the subdisks in the plex. This implies either that the physical disk failed, making it unrecognizable, or that the physical disk is no longer attached through a known access path. The plex cannot be used until this condition is fixed, or the affected subdisk is dissociated.

NODEVICE 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` in the disk group, `mydg`, use the following command:

```
# vxplex -g mydg 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 -g mydg -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 [-g diskgroup] 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` in the disk group, `mydg`, had subdisks on a drive to be repaired, use the following command to take these plexes offline:

```
# vxmend -g mydg off vol01-02 vol02-02
```

This command places `vol01-02` and `vol02-02` in the `OFFLINE` state, and they remain in that state until it is changed. The plexes are not automatically recovered on rebooting the system.

Detaching Plexes

To temporarily detach one data plex in a mirrored volume, use the following command:

```
# vxplex [-g diskgroup] det plex
```

For example, to temporarily detach a plex named `vol01-02` in the disk group, `mydg`, and place it in maintenance mode, use the following command:

```
# vxplex -g mydg det vol01-02
```

This command temporarily detaches the plex, but maintains the association between the plex and its volume. However, the plex is not used for I/O. A plex detached with the preceding command is recovered at system reboot. The plex state is set to `STALE`, so that if a `vxvol start` command is run on the appropriate volume (for example, on system reboot), the contents of the plex is recovered and made `ACTIVE`.

When the plex is ready to return as an active part of its volume, 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 [-g diskgroup] att volume plex ...
  ```

  For example, for a plex named `vol01-02` on a volume named `vol01` in the disk group, `mydg`, use the following command:

  ```
  # vxplex -g mydg att vol01 vol01-02
  ```

  As when returning an `OFFLINE` plex to `ACTIVE`, this command starts to recover the contents of the plex and, after the 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 [-g diskgroup] on plex
  ```

  For example, to re-enable a plex named `vol01-02` in the disk group, `mydg`, enter:

  ```
  # vxmend -g mydg on vol01-02
  ```

  In this case, the state of `vol01-02` is set to `STALE`. When the volume is next started, the data on the plex is revived from another plex, and incorporated into the volume with its state set to `ACTIVE`.

If the `vxinfo` command shows that the volume is unstartable (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 [-g diskgroup] fix clean plex
```

Start the volume using the following command:

```
# vxvol [-g diskgroup] 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 [-g diskgroup] mv original_plex new_plex
```

For a move task to be successful, the following criteria must be met:

- The old plex must be an active part of an active (`ENABLED`) volume.
- The new plex must be at least the same size or larger than the old plex.
- The new plex must not be associated with another volume.

The size of the plex has several implications:

- If the new plex is smaller or more sparse than the original plex, an incomplete copy is made of the data on the original plex. If an incomplete copy is desired, use the `-o force` option to `vxplex`.
- If the new plex is longer or less sparse than the original plex, the data that exists on the original plex is copied onto the new plex. Any area that is not on the original plex, but is represented on the new plex, is filled from other complete plexes associated with the same volume.
- If the new plex is longer than the volume itself, then the remaining area of the new plex above the size of the volume is not initialized and remains unused.

Copying Plexes

This task copies the contents of a volume onto a specified plex. The volume to be copied must not be enabled. The plex cannot be associated with any other volume. To copy a plex, use the following command:

```
# vxplex [-g diskgroup] cp volume new_plex
```

After the copy task is complete, *new_plex* is not associated with the specified volume *volume*. The plex contains a complete copy of the volume data. The plex that is being copied should be the same size or larger than the volume. If the plex being copied is larger than the volume, an incomplete copy of the data results. For the same reason, *new_plex* should not be sparse.

Dissociating and Removing Plexes

When a plex is no longer needed, you can dissociate it from its volume and remove it as an object from VxVM. You might want to remove a plex for the following reasons:

- to provide free disk space
- to reduce the number of mirrors in a volume so you can increase the length of another mirror and its associated volume. When the plexes and subdisks are removed, the resulting space can be added to other volumes
- to remove a temporary mirror that was created to back up a volume and is no longer needed
- to change the layout of a plex

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 158" for more information.

To dissociate a plex from the associated volume and remove it as an object from VxVM, use the following command:

```
# vxplex [-g diskgroup] -o rm dis plex
```

For example, to dissociate and remove a plex named `vol01-02` in the disk group, `mydg`, use the following command:

```
# vxplex -g mydg -o rm dis vol01-02
```

This command removes the plex `vol01-02` and all associated subdisks.

Alternatively, you can first dissociate the plex and subdisks, and then remove them with the following commands:

```
# vxplex [-g diskgroup] dis plex
# vxedit [-g diskgroup] -r rm plex
```

When used together, these commands produce the same result as the `vxplex -o rm dis` command. The `-r` option to `vxedit rm` recursively removes all objects from the specified object downward. In this way, a plex and its associated subdisks can be removed by a single `vxedit` command.

Changing Plex Attributes

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 [-g diskgroup] 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 -g mydg set comment="plex comment" tutil2="u" vol01-02
```

To prevent a particular plex from being associated with a volume, set the `putil0` field to a non-null string, as shown in the following command:

```
# vxedit -g mydg set putil0="DO-NOT-USE" vol01-02
```

See the `vxedit`(1M) manual page for more information about using the `vxedit` command to change the attribute fields of VxVM objects.

Creating Volumes 7

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.

Note You can also use the VERITAS Intelligent Storage Provisioning (ISP) feature to create and administer application volumes. These volumes are very similar to the traditional VxVM volumes that are described in this chapter. However, there are significant differences between the functionality of the two types of volume that prevents them from being used interchangeably. Refer to the *VERITAS Storage Foundation Intelligent Storage Provisioning Administrator's Guide* for more information about creating and administering ISP application volumes.

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 49 and "Creating and Administering Disk Groups" on page 123 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 19.
- 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 22.
- 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 26.
- 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 30.
- *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 26.
- Layered Volume—A volume constructed from other volumes. Non-layered volumes are constructed by mapping their subdisks to VM disks. Layered volumes are constructed by mapping their subdisks to underlying volumes (known as *storage volumes*), and allow the creation of more complex forms of logical layout. Examples of layered volumes are *striped-mirror* and *concatenated-mirror* volumes. For more information, see "Layered Volumes" on page 35.

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 27.

A concatenated-mirror volume is created by concatenating several mirrored volumes. This provides faster recovery as the failure of a single disk does not force the entire mirror offline.

Supported Volume Logs

VERITAS Volume Manager supports the use of the following types of logs with volumes:

- Dirty region logs allow the fast recovery of mirrored volumes after a system crash (see "Dirty Region Logging (DRL)" on page 43 for details). These logs are supported either as DRL log plexes or as version 20 DCO logs. See "Creating a Volume with Dirty Region Logging Enabled" on page 184 for more information.
- RAID-5 logs are used to prevent corruption of data during recovery of RAID-5 volumes (see "RAID-5 Logging" on page 34 for details). These logs are configured as plexes on disks other than those that are used for the columns of the RAID-5 volume. See "Creating a RAID-5 Volume" on page 188 for information on creating a RAID-5 volume together with RAID-5 logs.

Creating a Volume

You can create volumes using an *advanced* approach, an *assisted* approach, or the rule-based storage allocation approach that is provided by the Intelligent Storage Provisioning (ISP) feature. Each method uses different tools. You may switch between the advanced and the assisted approaches at will. However, volumes that you have created using ISP may only be modified using ISP-specific tools such as `vxvoladm`. For more information about ISP, see the *VERITAS Storage Foundation Intelligent Storage Provisioning Administrator's Guide*.

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 149.

2. Create plexes using `vxmake plex`, and associate subdisks with them; see "Creating Plexes" on page 157, "Associating Subdisks with Plexes" on page 152 and "Creating a Volume Using vxmake" on page 189.

3. Associate plexes with the volume using `vxmake vol`; see "Creating a Volume Using vxmake" on page 189.

4. Initialize the volume using `vxvol start` or `vxvol init zero`; see "Initializing and Starting a Volume Created Using vxmake" on page 192.

See "Creating a Volume Using a vxmake Description File" on page 191 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 189 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 Enterprise Administrator Getting Started* manual and VEA online help.

Using vxassist

You can use the `vxassist` utility to create and modify volumes. Specify the basic requirements for volume creation or modification, and `vxassist` performs the necessary tasks.

The advantages of using `vxassist` rather than the advanced approach include:

- Most actions require that you enter only one command rather than several.
- You are required to specify only minimal information to `vxassist`. If necessary, you can specify additional parameters to modify or control its actions.
- `Operations` result in a set of configuration changes that either succeed or fail as a group, rather than individually. System crashes or other interruptions do not leave intermediate states that you have to clean up. If `vxassist` finds an error or an exceptional condition, it exits after leaving the system in the same state as it was prior to the attempted `operation`.

The `vxassist` utility helps you perform the following tasks:

- Creating volumes.
- Creating mirrors for existing volumes.
- Growing or shrinking existing volumes.
- Backing up volumes online.
- Reconfiguring a volume’s layout online.

`vxassist` obtains most of the information it needs from sources other than your input. `vxassist` obtains information about the existing objects and their layouts from the objects themselves.

For tasks requiring new disk space, `vxassist` seeks out available disk space and allocates it in the configuration that conforms to the layout specifications and that offers the best use of free space.

The `vxassist` command takes this form:

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

where *keyword* selects the task to perform. The first argument after a `vxassist` keyword, *volume*, is a volume name, which is followed by a set of desired volume attributes. For example, the keyword `make` allows you to create a new volume:

```
# vxassist [options] make volume length [attributes]
```

The *length* of the volume can be specified in sectors, kilobytes, megabytes, or gigabytes 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 a default disk group according to the rules given in “Rules for Determining the Default Disk Group” on page 125. 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 177 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 177 for more information.

Disk Group Alignment Constraints on Volumes

Certain constraints apply to the length of volumes and to the numeric values of size attributes that apply to volumes. If a volume is created in a disk group that is compatible with the Cross-platform Data Sharing (CDS) feature, the volume’s length and the values of volume attributes that define the sizes of objects such as logs or stripe units, must be an integer multiple of the alignment value of 16 blocks (8 kilobytes). If the disk group is not compatible with the CDS feature, the volume’s length and attribute size values must be multiples of 1 block (512 bytes).

To discover the value in blocks of the alignment that is set on a disk group, use this command:

```
# vxprint -g diskgroup -G -F %align
```

By default, `vxassist` automatically rounds up the volume size and attribute size values to a multiple of the alignment value. (This is equivalent to specifying the attribute `dgalign_checking=round` as an additional argument to the `vxassist` command.)

If you specify the attribute `dgalign_checking=strict` to `vxassist`, the command fails with an error if you specify a volume length or attribute size value that is not a multiple of the alignment value for the disk group.

Creating a Volume on Any Disk

By default, the `vxassist make` command creates a concatenated volume that uses one or more sections of disk space. On a fragmented disk, this allows you to put together a volume larger than any individual section of free disk space available.

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 192 for details.

For example, to create the concatenated volume `voldefault` with a length of 10 gigabytes in the default disk group:

```
# vxassist -b make voldefault 10g
```

Creating a Volume on Specific Disks

VxVM automatically selects the disks on which each volume resides, unless you specify otherwise. If you want a volume to be created on specific disks, you must designate those disks to VxVM. More than one disk can be specified.

To create a volume on a specific disk or disks, use the following command:

```
# vxassist [-b] [-g diskgroup] make volume length [layout=layout] \
  diskname ...
```

For example, to create the volume `volspec` with length 5 gigabytes on disks `mydg03` and `mydg04`, use the following command:

```
# vxassist -b -g mydg make volspec 5g mydg03 mydg04
```

The vxassist command allows you to specify storage attributes. These give you control over the devices, including disks, controllers and targets, which vxassist uses to configure a volume. For example, you can specifically exclude disk mydg05:

```
# vxassist -b -g mydg make volspec 5g !mydg05
```

or exclude all disks that are on controller c2:

```
# vxassist -b -g mydg make volspec 5g !ctlr:c2
```

or include only disks on controller c1 except for target t5:

```
# vxassist -b -g mydg 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 -g bigone -b make volmega 20g bigone10 bigone11
```

or alternatively, use the diskgroup attribute:

```
# vxassist -b make volmega 20g diskgroup=bigone bigone10 bigone11
```

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 dgrp07 and dgrp08 when calculating the maximum size of RAID-5 volume that vxassist can create using the disks in the disk group dg:

```
# vxassist -b -g dgrp maxsize layout=raid5 nlog=2 !dgrp07 !dgrp08
```

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."

Specifying Ordered Allocation of Storage to Volumes

Ordered allocation gives you complete control of space allocation. It requires that the number of disks that you specify to the `vxassist` command must match the number of disks that are required to create a volume. The order in which you specify the disks to `vxassist` is also significant.

If you specify the `-o ordered` option to `vxassist` when creating a volume, any storage that you also specify is allocated in the following order:

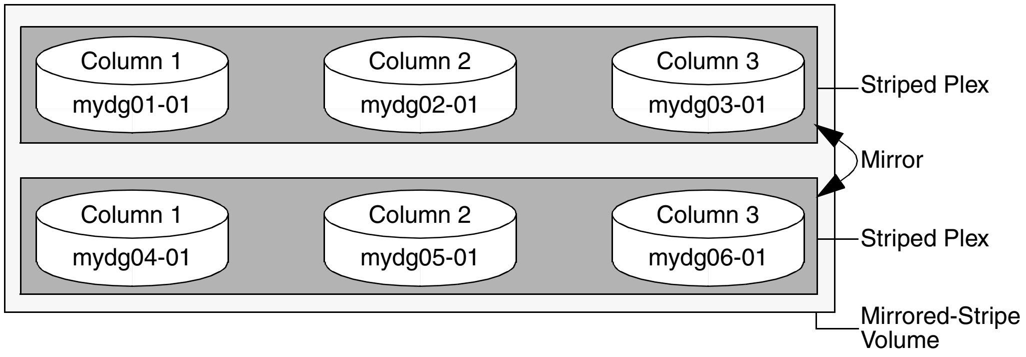
1. Concatenate disks.
2. Form columns.
3. Form mirrors.

For example, the following command creates a mirrored-stripe volume with 3 columns and 2 mirrors on 6 disks in the disk group, `mydg`:

```
# vxassist -b -g mydg -o ordered make mirstrvol 10g \
 layout=mirror-stripe ncol=3 \
 mydg01 mydg02 mydg03 mydg04 mydg05 mydg06
```

This command places columns 1, 2 and 3 of the first mirror on disks `mydg01`, `mydg02` and `mydg03` respectively, and columns 1, 2 and 3 of the second mirror on disks `mydg04`, `mydg05` and `mydg06` respectively. 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

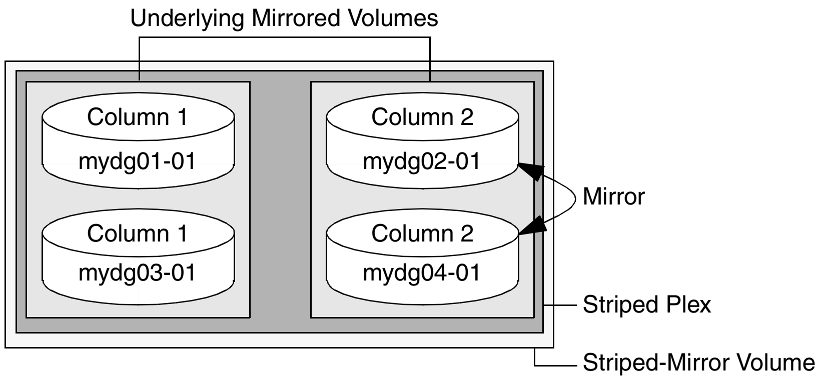


For layered volumes, vxassist applies the same rules to allocate storage as for non-layered volumes. For example, the following command creates a striped-mirror volume with 2 columns:

```
# vxassist -b -g mydg -o ordered make strmirvol 10g \
  layout=stripe-mirror ncol=2 mydg01 mydg02 mydg03 mydg04
```

This command mirrors column 1 across disks mydg01 and mydg03, and column 2 across disks mydg02 and mydg04, as illustrated in "Example of Using Ordered Allocation to Create a Striped-Mirror Volume."

Example of Using Ordered Allocation to Create a Striped-Mirror Volume

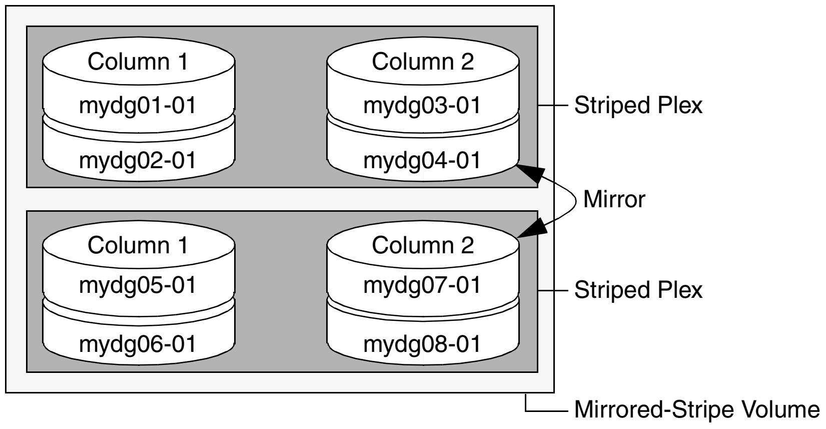


Additionally, you can use the col_switch attribute to specify how to concatenate space on the disks into columns. For example, the following command creates a mirrored-stripe volume with 2 columns:

```
# vxassist -b -g mydg -o ordered make strmir2vol 10g \
  layout=mirror-stripe ncol=2 col_switch=3g,2g \
  mydg01 mydg02 mydg03 mydg04 mydg05 mydg06 mydg07 mydg08
```

This command allocates 3 gigabytes from mydg01 and 2 gigabytes from mydg02 to column 1, and 3 gigabytes from mydg03 and 2 gigabytes from mydg04 to column 2. The mirrors of these columns are then similarly formed from disks mydg05 through mydg08. This arrangement is illustrated in "Example of Using Concatenated Disk Space to Create a Mirrored-Stripe Volume" on page 181.

Example of Using Concatenated Disk Space to Create a Mirrored-Stripe Volume

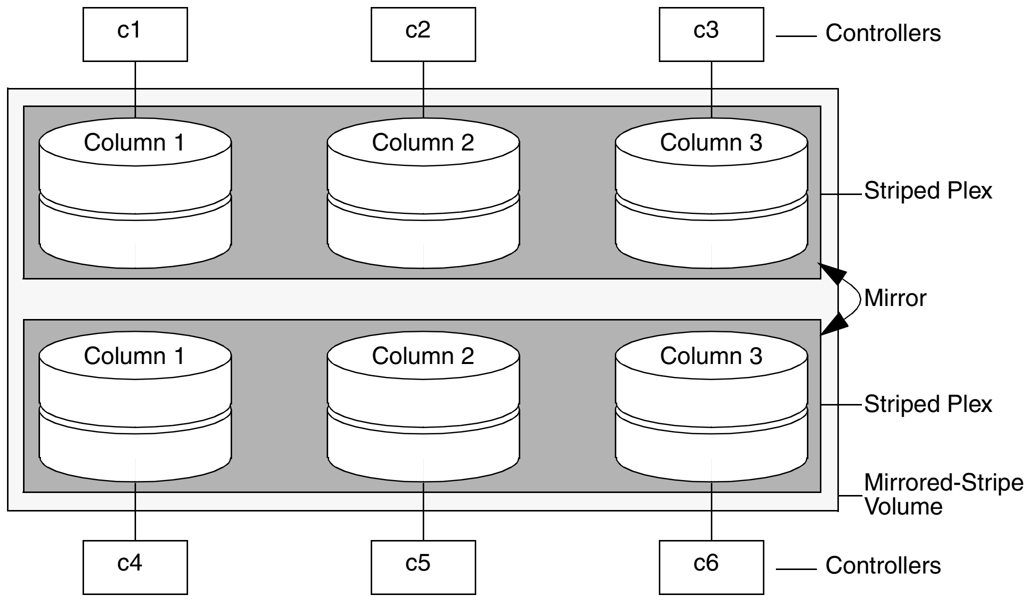


Other storage specification classes for controllers, enclosures, targets and trays can be used with ordered allocation. For example, the following command creates a 3-column mirrored-stripe volume between specified controllers:

```
# vxassist -b -g mydg -o ordered make mirstr2vol 80g \
  layout=mirror-stripe ncol=3 \
  ctlr:c1 ctlr:c2 ctlr:c3 ctlr:c4 ctlr:c5 ctlr:c6
```

This command allocates space for column 1 from disks on controllers `c1`, for column 2 from disks on controller `c2`, and so on as illustrated in "Example of Storage Allocation Used to Create a Mirrored-Stripe Volume Across Controllers" on page 182.

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 187.

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]
```

For example, to create the mirrored volume, `volmir`, in the disk group, `mydg`, use the following command:

```
# vxassist -b -g mydg make volmir 5g layout=mirror
```

To create a volume with 3 instead of the default of 2 mirrors, modify the command to read:

```
# vxassist -b -g mydg make volmir 5g layout=mirror nmirror=3
```

Creating a Mirrored-Concatenated Volume

A mirrored-concatenated volume mirrors several concatenated plexes. To create a concatenated-mirror volume, use the following command:

```
# vxassist [-b] [-g diskgroup] make volume length \
  layout=mirror-concat [nmirror=number]
```

Alternatively, first create a concatenated volume, and then mirror it as described in "Adding a Mirror to a Volume" on page 203.

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]
```

Creating a Volume with Dirty Region Logging Enabled

Note The procedure described in this section creates a traditional DRL log that is configured within a dedicated DRL plex. It is recommended that a version 20 DCO volume layout be used instead, which includes space for a DRL log. For full details, see "Creating a Volume with DCO-Based DRL Enabled" on page 184.

Dirty region logging (DRL), if enabled, speeds recovery of mirrored volumes after a system crash. To enable DRL on a volume, specify the `logtype=drl` attribute to the `vxassist make` command as shown in this example usage:

```
# vxassist [-g diskgroup] make volume length layout=layout \
  logtype=drl [nlog=n] [other attributes]
```

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 create a mirrored 10GB volume, `vol02`, with two log plexes in the disk group, `mydg`, use the following command:

```
# vxassist -g mydg make vol02 10g layout=mirror logtype=drl \
  nlog=2 nmirror=2
```

Sequential DRL limits the number of dirty regions for volumes that are written to sequentially, such as database replay logs. To enable sequential DRL on a volume that is created within a disk group with a version number between 70 and 100, specify the `logtype=drlseq` attribute to the `vxassist make` command.

```
# vxassist [-g diskgroup] make volume length layout=layout \
  logtype=drlseq [nlog=n] [other attributes]
```

Creating a Volume with DCO-Based DRL Enabled

Use the following command to create a volume in which DRL is configured to use a version 20 DCO volume (you may need to specify additional attributes to create a volume with the desired characteristics):

```
# vxassist [-g diskgroup] make volume length layout=layout \
  logtype=dco dcoversion=20 [drl=on|sequential|off] \
  [ndcomirror=number] [other attributes]
```

Set the value of the `drl` attribute to `on` if dirty region logging (DRL) is to be used with the volume (this is the default setting). For a volume that will be written to sequentially, such as a database log volume, set the value to `sequential` to enable sequential DRL. The DRL logs are created in the DCO volume. The redundancy of the logs is determined by the number of mirrors that you specify using the `ndcomirror` attribute.

For more information, see the `vxassist`(1M) manual page.

Sequential DRL limits the number of dirty regions for volumes that are written to sequentially, such as database replay logs. To enable sequential DRL on a volume that is created within a disk group with a version number between 70 and 100, specify the `logtype=drlseq` attribute to the `vxassist make` command.

```
# vxassist [-g diskgroup] make volume length layout=layout \
  logtype=drlseq [nlog=n] [other attributes]
```

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 22.

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
```

For example, to create the 10-gigabyte striped volume `volzebra`, in the disk group, `mydg`, use the following command:

```
# vxassist -b -g mydg make volzebra 10g layout=stripe
```

This creates a striped volume with the default stripe unit size (64 kilobytes) and the default number of stripes (2).

You can specify the disks on which the volumes are to be created by including the disk names on the command line. For example, to create a 30-gigabyte striped volume on three specific disks, `mydg03`, `mydg04`, and `mydg05`, use the following command:

```
# vxassist -b -g mydg make stripevol 30g layout=stripe \
  mydg03 mydg04 mydg05
```

To change the number of columns or the stripe width, use the `ncolumn` and `stripeunit` modifiers with `vxassist`. For example, the following command creates a striped volume with 5 columns and a 32-kilobyte stripe size:

```
# vxassist -b -g mydg make stripevol 30g layout=stripe \
  stripeunit=32k ncol=5
```

Creating a Mirrored-Stripe Volume

A mirrored-stripe volume mirrors several striped data plexes.

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_of_columns] [stripewidth=size]
```

Alternatively, first create a striped volume, and then mirror it as described in "Adding a Mirror to a Volume" on page 203. 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_of_columns] [stripewidth=size]
```

By default, VxVM attempts to create the underlying volumes by mirroring subdisks rather than columns if the size of each column is greater than the value for the attribute `stripe-mirror-col-split-trigger-pt` that is defined in the `vxassist` defaults file.

If there are multiple subdisks per column, you can choose to mirror each subdisk individually instead of each column. To mirror at the subdisk level, specify the layout as `stripe-mirror-sd` rather than `stripe-mirror`. To mirror at the column level, specify the layout as `stripe-mirror-col` rather than `stripe-mirror`.

Mirroring across Targets, Controllers or Enclosures

To create a volume whose mirrored data plexes lie on different controllers (also known as *disk duplexing*) or in different enclosures, use the `vxassist` command as described in this section.

In the following command, the attribute `mirror=target` specifies that volumes should be mirrored between identical target IDs on different controllers.

```
# vxassist [-b] [-g diskgroup] make volume length layout=layout \
  mirror=target [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:

```
# vxassist [-b] [-g diskgroup] make volume length layout=layout \
  mirror=ctlr [attributes]
```

Note Both paths of an active/passive array are not considered to be on different controllers when mirroring across controllers.

The following command creates a mirrored volume with two data plexes in the disk group, `mydg`:

```
# vxassist -b -g mydg make volspec 10g layout=mirror nmirror=2 \
  mirror=ctlr ctlr:c2 ctlr:c3
```

The disks in one data plex are all attached to controller `c2`, and the disks in the other data plex are all attached to controller `c3`. This arrangement ensures continued availability of the volume should either controller fail.

The attribute `mirror=enclr` specifies that disks in one mirror should not be in the same enclosure as disks in other mirrors within the same volume.

The following command creates a mirrored volume with two data plexes:

```
# vxassist -b make -g mydg volspec 10g layout=mirror nmirror=2 \
  mirror=enclr enclr:enc1 enclr:enc2
```

The disks in one data plex are all taken from enclosure `enc1`, and the disks in the other data plex are all taken from enclosure `enc2`. This arrangement ensures continued availability of the volume should either enclosure become unavailable.

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

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 30.

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_of_columns] [stripewidth=size] [nlog=number] \
  [loglen=log_length]
```

For example, to create the RAID-5 volume `volraid` together with 2 RAID-5 logs in the disk group, `mydg`, use the following command:

```
# vxassist -b -g mydg make volraid 10g layout=raid5 nlog=2
```

This creates a RAID-5 volume with the default stripe unit size on the default number of disks. It also creates two RAID-5 logs rather than the default of one log.

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 mydg04, mydg05 and mydg06. It also creates two RAID-5 logs on disks mydg07 and mydg08.

```
# vxassist -b -g mydg -o ordered make volraid 10g layout=raid5 \
  ncol=3 nlog=2 logdisk=mydg07,mydg08 mydg04 mydg05 mydg06
```

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 210.

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 149. The example given in this section is to create a RAID-5 volume using vxmake.

Creating a RAID-5 plex for a RAID-5 volume is similar to creating striped plexes, except that the layout attribute is set to raid5. Subdisks can be implicitly associated in the same way as with striped plexes. For example, to create a four-column RAID-5 plex with a stripe unit size of 32 sectors, use the following command:

```
# vxmake -g mydg plex raidplex layout=raid5 stwidth=32 \
  sd=mydg00-01,mydg01-00,mydg02-00,mydg03-00
```

Note that because four subdisks are specified, but the number of columns is not specified, the vxmake command assumes a four-column RAID-5 plex and places one subdisk in each column. Striped plexes are created using the same method except that the layout is specified as stripe. If the subdisks are to be created and added later, use the following command to create the plex:

```
# vxmake -g mydg plex raidplex layout=raid5 ncolumn=4 stwidth=32
```

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 152).

If each column in a RAID-5 plex is to be created from multiple subdisks which may span several physical disks, you can specify to which column each subdisk should be added. For example, to create a three-column RAID-5 plex using six subdisks, use the following form of the vxmake command:

```
# vxmake -g mydg plex raidplex layout=raid5 stwidth=32 \
 sd=mydg00-00:0,mydg01-00:1,mydg02-00:2,mydg03-00:0, \
 mydg04-00:1,mydg05-00:2
```

This command stacks subdisks mydg00-00 and mydg03-00 consecutively in column 0, subdisks mydg01-00 and mydg04-00 consecutively in column 1, and subdisks mydg02-00 and mydg05-00 in column 2. Offsets can also be specified to create sparse RAID-5 plexes, as for striped plexes.

Log plexes may be created as default concatenated plexes by not specifying a layout, for example:

```
# vxmake -g mydg plex raidlog1 sd=mydg06-00
# vxmake -g mydg plex raidlog2 sd=mydg07-00
```

The following command creates a RAID-5 volume, and associates the prepared RAID-5 plex and RAID-5 log plexes with it:

```
# vxmake -g mydg -Uraid5 vol raidvol \
 plex=raidplex,raidlog1,raidlog2
```

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 192.

Creating a Volume Using a vxmake Description File

You can use the `vxmake` command to add a new volume, plex or subdisk to the set of objects managed by VxVM. `vxmake` adds a record for each new object to the VxVM configuration database. You can create records either by specifying parameters to `vxmake` on the command line, or by using a file which contains plain-text descriptions of the objects. The file can also contain commands for performing a list of tasks. Use the following form of the command to have `vxmake` read the file from the standard input:

```
# vxmake [-g diskgroup] < description_file
```

Alternatively, you can specify the file to `vxmake` using the `-d` option:

```
# vxmake [-g diskgroup] -d description_file
```

The following sample description file defines a volume, `db`, with two plexes, `db-01` and `db-02`:

```
#rectyp #name      #options
sd      mydg03-01  disk=mydg03 offset=0 len=10000
sd      mydg03-02  disk=mydg03 offset=25000 len=10480
sd      mydg04-01  disk=mydg04 offset=0 len=8000
sd      mydg04-02  disk=mydg04 offset=15000 len=8000
sd      mydg04-03  disk=mydg04 offset=30000 len=4480
plex    db-01      layout=STRIPE ncolumn=2 stwidth=16k
                   sd=mydg03-01:0/0,mydg03-02:0/10000,mydg04-01:1/0,
mydg04-02:1/8000,mydg04-03:1/16000
sd      ramd1-01   disk=ramd1 len=640
                   comment="Hot spot for dbvol
plex    db-02      sd=ramd1-01:40320
vol     db         usetype=gen plex=db-01,db-02
                   readpol=prefer prefname=db-02
                   comment="Uses mem1 for hot spot in last 5m
```

Note The subdisk definition for plex, `db-01`, must be specified on a single line. It is shown here split across two lines because of space constraints.

The first plex, `db-01`, is striped and has five subdisks on two physical disks, `mydg03` and `mydg04`. The second plex, `db-02`, is the preferred plex in the mirror, and has one subdisk, `ramd1-01`, on a volatile memory disk.

For detailed information about how to use `vxmake`, refer to the `vxmake`(1M) manual page.

After creating a volume using `vxmake`, you must initialize it before it can be used. The procedure is described in "Initializing and Starting a Volume Created Using vxmake" on page 192.

Initializing and Starting a Volume

If you create a volume using the `vxassist` command, `vxassist` initializes and starts the volume automatically unless you specify the attribute `init=none`.

When creating a volume, you can make it immediately available for use by specifying the `-b` option to the `vxassist` command, as shown here:

```
# vxassist -b [-g diskgroup] make volume length layout=mirror
```

The `-b` option makes VxVM carry out any required initialization as a background task. It also greatly speeds up the creation of striped volumes by initializing the columns in parallel.

As an alternative to the `-b` option, you can specify the `init=active` attribute to make a new volume immediately available for use. In this example, `init=active` is specified to prevent VxVM from synchronizing the empty data plexes of a new mirrored volume:

```
# vxassist [-g diskgroup] make volume length layout=mirror \
  init=active
```

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 uninitialized following a system crash. If in doubt, use the `-b` option to `vxassist` instead.

This command writes zeroes to the entire length of the volume and to any log plexes. It then makes the volume active. You can also zero out a volume by specifying the attribute `init=zero` to `vxassist`, as shown in this example:

```
# vxassist [-g diskgroup] make volume length layout=raid5 \
  init=zero
```

Note You cannot use the `-b` option to make this operation a background task.

Initializing and Starting a Volume Created Using vxmake

A volume may be initialized by running the `vxvol` command if the volume was created by the `vxmake` command and has not yet been initialized, or if the volume has been set to an uninitialized state.

To initialize and start a volume, use the following command:

```
# vxvol [-g diskgroup] start volume
```

The following command can be used to enable a volume without initializing it:

```
# vxvol [-g diskgroup] init enable volume
```

This allows you to restore data on the volume from a backup before using the following command to make the volume fully active:

```
# vxvol [-g diskgroup] init active volume
```

If you want to zero out the contents of an entire volume, use this command to initialize it:

```
# vxvol [-g diskgroup] init zero volume
```

Accessing a Volume

As soon as a volume has been created and initialized, it is available for use as a virtual disk partition by the operating system for the creation of a file system, or by application programs such as relational databases and other data management software.

Creating a volume in a disk group sets up block and character (raw) device files that can be used to access the volume:

`/dev/vx/dsk/`*diskgroup*`/`*volume*	block device file for *volume*
`/dev/vx/rdsk/`*diskgroup*`/`*volume*	character device file for *volume*

The pathnames include a directory named for the disk group. Use the appropriate device node to create, mount and repair file systems, and to lay out databases that require raw partitions.

Administering Volumes 8

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, and changing the layout of volumes without taking them offline.

Note You can also use the VERITAS Intelligent Storage Provisioning (ISP) feature to create and administer application volumes. These volumes are very similar to the traditional VxVM volumes that are described in this chapter. However, there are significant differences between the functionality of the two types of volumes that prevents them from being used interchangeably. Refer to the *VERITAS Storage Foundation Intelligent Storage Provisioning Administrator's Guide* for more information about creating and administering ISP application volumes.

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 -hvt
```

The `vxprint` command can also be applied to a single disk group:

```
# vxprint -g mydg -hvt
```

This is example output from this command:

```
Disk group: mydg

V   NAME      RVG/VSET/CO KSTATE   STATE    LENGTH   READPOL    PREFPLEX  UTYPE
PL  NAME      VOLUME      KSTATE   STATE    LENGTH   LAYOUT     NCOL/WID  MODE
SD  NAME      PLEX        DISK     DISKOFFS LENGTH   [COL/]OFF  DEVICE    MODE
SV  NAME      PLEX        VOLNAME  NVOLLAYR LENGTH   [COL/]OFF  AM/NM     MODE
SC  NAME      PLEX        CACHE    DISKOFFS LENGTH   [COL/]OFF  DEVICE    MODE
DC  NAME      PARENTVOL   LOGVOL
SP  NAME      SNAPVOL     DCO

v   pubs      -           ENABLED  ACTIVE   22880    SELECT     -         fsgen
pl  pubs-01   pubs        ENABLED  ACTIVE   22880    CONCAT     -         RW
sd  mydg11-01 pubs-01     mydg11   0        22880    0          c1t0d0    ENA

v   voldef    -           ENABLED  ACTIVE   20480    SELECT     -         fsgen
pl  voldef-01 voldef      ENABLED  ACTIVE   20480    CONCAT     -         RW
sd  mydg12-02 voldef-0    mydg12   0        20480    0          c1t1d0    ENA
```

Here `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.

Note The headings for sub-volumes (`SV`), storage caches (`SC`), data change objects (`DCO`) and snappoints (`SP`) can be ignored here. No such objects are associated with these volumes.

To display volume-related information for a specific volume, use the following command:

```
# vxprint [-g diskgroup] -t volume
```

For example, to display information about the volume, `voldef`, in the disk group, `mydg`, use the following command:

```
# vxprint -g mydg -t voldef
```

This is example output from this command:

```
Disk group: mydg

V   NAME      RVG/VSET/CO KSTATE   STATE    LENGTH   READPOL    PREFPLEX  UTYPE

v   voldef    -           ENABLED  ACTIVE   20480    SELECT     -         fsgen
```

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 "Discovering the Association between Enclosure-Based Disk Names and OS-Based Disk Names" on page 64 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.

INVALID Volume State

The contents of an instant snapshot volume no longer represent a true point-in-time image of the original volume.

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 from or written to, but certain plex 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 `vxassist`, `vxevac`, `vxplex`, `vxmirror`, `vxrecover`, `vxrelayout`, `vxresize`, `vxsd`, and `vxvol` utilities allow you to specify a tag using the `-t` option. 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 -g mydg -t myrecovery -b mydg05
```

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, specify 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 224.

Stopping a Volume

Stopping a volume renders it unavailable to the user, and changes the volume kernel state from ENABLED or DETACHED to DISABLED. If the volume cannot be disabled, it remains in its current state. To stop a volume, use the following command:

```
# vxvol [-g diskgroup] [-f] stop volume ...
```

To stop all volumes in a specified disk group, use the following command:

```
# vxvol [-g diskgroup] [-f] stopall
```

Caution If you use the `-f` option to forcibly disable a volume that is currently open to an application, the volume remains open, but its contents are inaccessible. I/O operations on the volume fail, and this may cause data loss. It is not possible to deport a disk group until all of its volumes are closed.

If you need to prevent a closed volume from being opened, it is recommended that you use the `vxvol maint` command, as described in the following section.

Putting a Volume in Maintenance Mode

If all mirrors of a volume become `STALE`, you can place the volume in maintenance mode. Then you can view the plexes while the volume is `DETACHED` and determine which plex to use for reviving the others. To place a volume in maintenance mode, use the following command:

```
# vxvol [-g diskgroup] maint volume
```

To assist in choosing the revival source plex, use `vxprint` to list the stopped volume and its plexes.

To take a plex (in this example, `vol01-02` in the disk group, `mydg`) offline, use the following command:

```
# vxmend -g mydg 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 -g mydg 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` in the disk group, `mydg`, use the following command:

```
# vxassist -b -g mydg 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 [-g diskgroup] plex plex sd=subdisk ...
# vxplex [-g diskgroup] att volume plex
```

Mirroring All Volumes

To mirror all volumes in a disk group to available disk space, use the following command:

```
# /etc/vx/bin/vxmirror -g diskgroup -a
```

To configure VxVM to create mirrored volumes by default, use the following command:

```
# /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` in the disk group, `mydg`, use the following command:

```
# vxassist -g mydg make nomirror 20g nmirror=1
```

Mirroring Volumes on a VM Disk

Mirroring volumes 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.

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,?] mydg02
   ```

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 mydg02 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) mydg01
```

4. At the following prompt, press Return to make the mirror:

   ```
   The requested operation is to mirror all volumes on disk mydg02
   in disk group mydg onto available disk space on disk mydg01.

   VxVM NOTICE V-5-2-3650 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:

   ```
   VxVM vxmirror INFO V-5-2-22 Mirror volume voltest-bk00 ...

   VxVM INFO V-5-2-674 Mirroring of disk mydg01 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 [-g diskgroup] remove mirror volume
```

Additionally, you can use storage attributes to specify the storage to be removed. For example, to remove a mirror on disk `mydg01` from volume vol01, enter:

```
# vxassist -g mydg remove mirror vol01 !mydg01
```

For more information about storage attributes, see "Creating a Volume on Specific Disks" on page 177.

Alternatively, use the following command to dissociate and remove a mirror from a volume:

```
# vxplex [-g diskgroup] -o rm dis plex
```

For example, to dissociate and remove a mirror named `vol01-02` from the disk group, `mydg`, use the following command:

```
# vxplex -g mydg -o rm dis vol01-02
```

This command removes the mirror `vol01-02` and all associated subdisks. This is equivalent to entering the following separate commands:

```
# vxplex -g mydg dis vol01-02
# vxedit -g mydg -r rm vol01-02
```

Adding Logs to Volumes

In VERITAS Volume Manager, the following types of logs are supported:

- Dirty Region Logs allow the fast recovery of mirrored volumes after a system crash (see "Dirty Region Logging (DRL)" on page 43 for details). These logs are supported either as DRL log plexes or as version 20 DCO logs. See "Adding Dirty Region Logging to a Mirrored Volume" on page 206 and "Preparing a Volume to Use DCO-Based DRL" on page 207 for more information.
- RAID-5 logs are used to prevent corruption of data during recovery of RAID-5 volumes (see "RAID-5 Logging" on page 34 for details). These logs are configured as plexes on disks other than those that are used for the columns of the RAID-5 volume. See "Adding a RAID-5 Log" on page 210 for information on adding RAID-5 logs to a RAID-5 volume.

Adding Dirty Region Logging to a Mirrored Volume

Note The procedure described in this section creates a traditional DRL log that is configured within a dedicated DRL plex. It is recommended that a version 20 DCO volume layout be used instead, which includes space for a DRL log. For full details, see "Preparing a Volume to Use DCO-Based DRL" on page 207.

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] [-g diskgroup] 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`, in the disk group, `mydg`, use the following command:

```
# vxassist -g mydg 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 -g mydg 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 [-g diskgroup] 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.

Preparing a Volume to Use DCO-Based DRL

Note This procedure describes how to add a version 20 data change object (DCO) and DCO volume to a volume. If you are creating a new volume in a disk group, you can specify the co-creation of a DCO and DCO volume and enable DRL as described in "Creating a Volume with DCO-Based DRL Enabled" on page 184.

You need a full VxVM license to use the DRL feature.

Use the following command to add a version 20 DCO and DCO volume to a volume:

```
# vxsnap [-g diskgroup] prepare volume [ndcomirs=number] \
  [regionsize=size] [drl=on|sequential|off] \
  [storage_attribute ...]
```

The `ndcomirs` attribute specifies the number of DCO plexes that are created in the DCO volume. It is recommended that you configure as many DCO plexes as there are data in the volume. For example, specify `ndcomirs=3` for a volume with 3 data plexes.

The value of the `regionsize` attribute specifies the size of the tracked regions in the volume. A write to a region is tracked by setting a bit in the change map. The default value is `64k` (64KB). A smaller value requires more disk space for the change maps, but the finer granularity provides faster resynchronization.

To enable DRL logging on the volume, specify `drl=on` (this is the default setting). If sequential DRL is required, specify `drl=sequential`. If DRL is not required, specify `drl=off`.

You can also specify `vxassist`-style storage attributes to define the disks that can and/or cannot be used for the plexes of the DCO volume. See "Determining if DRL is Enabled on a Volume" on page 208 for details.

Note By default, a version 20 DCO volume contains 32 per-volume maps. If you require more maps than this, you can use the `vxsnap addmap` command to add more maps. See the `vxsnap`(1M) manual page for details of this command.

Determining if DRL is Enabled on a Volume

To determine if DRL (configured using a version 20 DCO volume) is enabled on a volume:

1. Use the `vxprint` command on the volume to discover the name of its DCO:

   ```
   # DCONAME=`vxprint [-g diskgroup] -F%dco_name volume`
   ```

2. To determine if DRL is enabled on the volume, use the following command with the volume's DCO:

   ```
   # vxprint [-g diskgroup] -F%drl $DCONAME
   ```

   DRL is enabled if this command displays `on`.

3. If DRL is enabled, use this command with the DCO to determine if sequential DRL is enabled:

   ```
   # vxprint [-g diskgroup] -F%sequentialdrl $DCONAME
   ```

   Sequential DRL is enabled if this command displays `on`.

Alternatively, you can use this command with the volume:

```
# vxprint [-g diskgroup] -F%log_type volume
```

This displays the logging type as `REGION` for DRL, `DRLSEQ` for sequential DRL, or `NONE` if DRL is not enabled.

Note If the number of active mirrors in the volume is less than 2, DRL logging is not performed even if DRL is enabled on the volume. To find out if DRL logging is active, see "Determining if DRL Logging is Active on a Volume" on page 209.

Determining if DRL Logging is Active on a Volume

To determine if DRL logging is active on a mirrored volume:

1. Use the following `vxprint` commands to discover the name of the volume's DCO volume:

   ```
   # DCONAME=`vxprint [-g diskgroup] -F%dco_name volume`
   # DCOVOL=`vxprint [-g diskgroup] -F%parent_vol $DCONAME`
   ```

2. Use the `vxprint` command on the DCO volume to find out if DRL logging is active:

   ```
   # vxprint [-g diskgroup] -F%drllogging $DCOVOL
   ```

   This command returns `on` if DRL logging is enabled.

Disabling and Re-enabling DRL

To disable DRL (configured using a version 20 DCO volume) on a volume, enter the following command:

```
# vxvol [-g diskgroup] set drl=off volume
```

To re-enable DRL on a volume, enter this command:

```
# vxvol [-g diskgroup] set drl=on volume
```

To re-enable sequential DRL on a volume, enter:

```
# vxvol [-g diskgroup] set drl=sequential volume
```

You can use these commands to change the DRL policy on a volume by first disabling and then re-enabling DRL as required. DRL is automatically disabled if a data change map (DCM, used with VERITAS Volume Replicator) is attached to a volume.

Removing Support for DRL from a Volume

To remove support for DRL (configured using a version 20 DCO volume) from a volume, use the following command to remove the DCO and DCO volume that are associated with the volume:

```
# vxsnap [-g diskgroup] unprepare volume
```

Adding a RAID-5 Log

Note You need a full 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] [-g diskgroup] addlog volume [loglen=length]
```

Note If specified, the `-b` option makes adding the new log a background task.

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`, in the disk group, `mydg`, use the following command:

```
# vxassist -g mydg 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`, in the disk group, `mydg`, use the following command:

```
# vxplex -g mydg att r5vol r5log
```

The attach operation can only proceed if the size of the new log is large enough to hold all 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 [-g diskgroup] -ht volume
```

where *volume* is the name of the RAID-5 volume. For a RAID-5 log, the output lists a plex with a `STATE` field entry of `LOG`.

To dissociate and remove a RAID-5 log and any associated subdisks from an existing volume, use the following command:

```
# vxplex [-g diskgroup] -o rm dis plex
```

For example, to dissociate and remove the log plex `volraid-02` from `volraid` in the disk group, `mydg`, use the following command:

```
# vxplex -g mydg -o rm dis volraid-02
```

You can also remove a RAID-5 log with the `vxassist` command, as follows:

```
# vxassist [-g diskgroup] remove log volume [nlog=n]
```

By default, the `vxassist` command removes one log. Use the optional attribute `nlog=n` to specify the number of logs that are to remain after the operation completes.

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.

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 [-g diskgroup] maxgrow volume
```

When you resize a volume, you can specify the length of a new volume in sectors, kilobytes, megabytes, or gigabytes. The unit of measure is added as a suffix to the length (`s`, `m`, `k`, or `g`). If 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`, in the disk group, `mydg`, that contains a VxFS file system to 10 gigabytes using the spare disks `mydg10` and `mydg11`:

```
# vxresize -g mydg -b -F vxfs -t homevolresize homevol 10g \
  mydg10 mydg11
```

The `-b` option specifies that this operation runs in the background. 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 V-5-1-2536 Volume volume has different
  organization in each mirror
  ```

  To resize such a volume successfully, you must first reconfigure it so that each data plex has the same layout.

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

Extending to a Given Length

To extend a volume *to* a specific length, use the following command:

```
# vxassist [-b] [-g diskgroup] 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 -g mydg 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] [-g diskgroup] 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 -g mydg 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 [-g diskgroup] shrinkto volume length
```

For example, to shrink `volcat` to 1300 sectors, use the following command:

```
# vxassist -g mydg 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 [-g diskgroup] shrinkby volume length
```

For example, to shrink `volcat` by 300 sectors, use the following command:

```
# vxassist -g mydg 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 [-g diskgroup] set len=length volume
```

For example, to change the length of the volume, `vol01`, in the disk group, `mydg`, to 100000 sectors, use the following command:

```
# vxvol -g mydg set len=100000 vol01
```

Note 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 [-g diskgroup] 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 [-g diskgroup] rdpol round volume
```

For example, to set the read policy for the volume, `vol01`, in disk group, `mydg`, to round-robin, use the following command:

```
# vxvol -g mydg rdpol round vol01
```

To set the read policy to `prefer`, use the following command:

```
# vxvol [-g diskgroup] rdpol prefer volume preferred_plex
```

For example, to set the policy for `vol01` to read preferentially from the plex `vol01-02`, use the following command:

```
# vxvol -g mydg rdpol prefer vol01 vol01-02
```

To set the read policy to `select`, use the following command:

```
# vxvol [-g diskgroup] rdpol select volume
```

For more information about how read policies affect performance, see “Volume Read Policies” on page 282.

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 [-g diskgroup] stop volume
```

After following these steps, remove the volume with the `vxassist` command:

```
# vxassist [-g diskgroup] remove volume volume
```

Alternatively, you can use the `vxedit` command to remove a volume:

```
# vxedit [-g diskgroup] [-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,?] mydg01
```

You can now optionally specify a list of disks to which the volume(s) should be moved:

```
VxVM INFO V-5-2-516 You can now specify a list of disks to move
```

```
  onto.  Specify a list of disk media names (e.g., mydg01) 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 prompt, press Return to move the volumes onto available space in the disk group, or specify the disks in the disk group that should be used:

```
Enter disks [<disk ...>,list]
  VxVM  NOTICE V-5-2-283 Requested operation is to move all
  volumes from disk mydg01 in group mydg.

  NOTE: This operation can take a long time to complete.

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

As the volumes are moved from the disk, the `vxdiskadm` program displays the status of the operation:

```
  VxVM vxevac INFO V-5-2-24   Move volume voltest ...
```

When the volumes have all been moved, the `vxdiskadm` program displays the following success message:

```
   VxVM  INFO V-5-2-188 Evacuation of disk mydg02 is complete.
```

3. At the following prompt, indicate whether you want to move volumes from another disk (**`y`**) or return to the `vxdiskadm` main menu (**`n`**):

```
  Move volumes from another disk? [y,n,q,?] (default: n)
```

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 220:

`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, `vol02`, in disk group, `mydg`, to a striped volume with the default number of columns, 2, and default stripe unit size, 64 kilobytes:

```
# vxassist -g mydg 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 223 for more information.

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.
stripe-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 be changed.
stripe-mirror	Yes. The stripe width and number of columns may 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.

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 [-g diskgroup] 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" on page 223 and "Controlling the Progress of a Relayout" on page 224, 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 -g mydg 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 -g mydg -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 10-megabyte region:

```
# vxrelayout -g mydg -o bg,slow=1000,iosize=10m start vol04
```

The default delay and region size values are 250 milliseconds and 1 megabyte respectively.

To reverse the direction of relayout operation that is currently stopped, specify the `reverse` keyword to `vxrelayout` as shown in this example:

```
# vxrelayout -g mydg -o bg reverse vol04
```

This undoes changes made to the volume so far, and returns it to its original layout.

If you cancel a relayout using `vxtask abort`, the direction of the conversion is also reversed, and the volume is returned to its original configuration.

See the `vxrelayout`(1M) and vxtask(1M) manual pages for more information about these commands. See “Managing Tasks with vxtask” on page 200 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] [-g diskgroup] 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 as shown here:

```
# vxassist -g mydg 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 -g mydg convert vol1 layout=mirror-stripe
```

Note If the system crashes during relayout or conversion, the process continues when the system is rebooted. However, if the 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 Volume Snapshots 9

Note The Data Change Object (DCO) and FastResync features, which allow fast resynchronization of snapshots, and instant snapshot functionality are not supported in this release.

VERITAS Volume Manager (VxVM) provides the capability for taking an image of a volume at a given point in time. Such an image is referred to as a *volume snapshot*.

Volume snapshots allow you to make backup copies of your volumes online with minimal interruption to users. You can then use the backup copies to restore data that has been lost due to disk failure, software errors or human mistakes, or to create replica volumes for the purposes of report generation, application development, or testing.

For an introduction to the volume snapshot feature, see "Volume Snapshots" on page 46. More detailed descriptions of each type of volume snapshot and the operations that you can perform on them may be found in the following sections:

- "Third-Mirror Break-Off Snapshots" on page 228
- "Creating Multiple Snapshots" on page 229
- "Restoring the Original Volume from a Snapshot" on page 230

Note A volume snapshot represents the data that exists in a volume at a given point in time. As such, VxVM does not have any knowledge of data that is cached by the overlying file system, or by applications such as databases that have files open in the file system. If the `fsgen` volume usage type is set on a volume that contains a VERITAS File System (VxFS), intent logging of the file system metadata ensures the internal consistency of the file system that is backed up. For other file system types, depending on the intent logging capabilities of the file system, there may potentially be inconsistencies between data in memory and in the snapshot image.

For databases, a suitable mechanism must additionally be used to ensure the integrity of tablespace data when the volume snapshot is taken. The facility to temporarily suspend file system I/O is provided by most modern database software. For ordinary files in a file system, which may be open to a wide variety of different applications, there may be no way to ensure the complete integrity of the file data other than by shutting down the applications and temporarily unmounting the file system. In many cases, it may only be important to ensure the integrity of file data that is not in active use at the time that you take the snapshot.

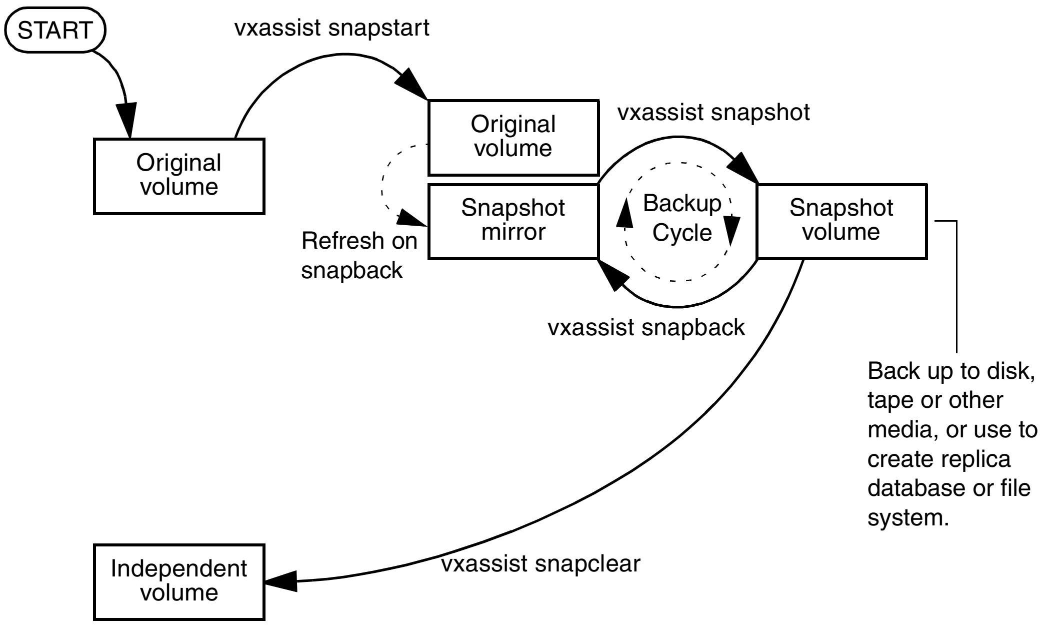
Methods of creating and administering third-mirror volume snapshots are described in "Creating Third-Mirror Break-Off Snapshots" on page 230.

Note Most VxVM commands require superuser or equivalent privileges.

Third-Mirror Break-Off Snapshots

The *third-mirror break-off* volume snapshot model that is supported by the `vxassist` command is shown in "Third-Mirror Snapshot Creation and Usage." This figure also shows the transitions that are supported by the `snapback` and `snapclear` commands to `vxassist`.

Third-Mirror Snapshot Creation and Usage



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

Note As is usual when creating a mirror, the process of copying the volume's contents to the new snapshot plexes can take some time to complete.

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. You can also use a variation of the same command to restore the contents of the original volume from a snapshot that you took at an earlier point in time. See "Restoring the Original Volume from a Snapshot" on page 230 for more information.

Finally, 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.

See "Creating Third-Mirror Break-Off Snapshots" on page 230 for a description of the procedures for creating and using this type of snapshot.

Creating Multiple Snapshots

To make it easier to create snapshots of several volumes at the same time, the `vxassist snapshot` command accepts more than one volume name as its argument.

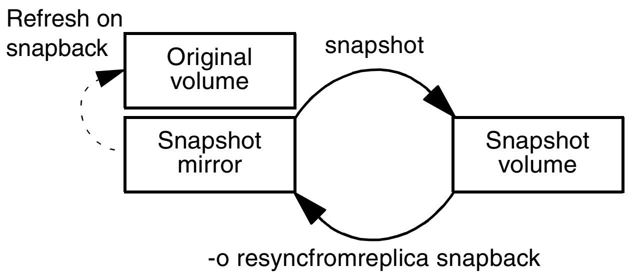
You can create snapshots of all the volumes in a single disk group by specifying the option `-o allvols` to the `vxassist snapshot` command.

By default, each replica volume is named `SNAP`*number*-*volume*, where number is a unique serial number, and *volume* is the name of the volume for which a snapshot is being taken. This default can be overridden by using the option `-o name=`*pattern*, as described on the `vxassist`(1M) manual page.

Restoring the Original Volume from a Snapshot

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

Resynchronizing an Original Volume from a 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.

Creating Third-Mirror Break-Off Snapshots

VxVM provides third-mirror break-off snapshot images of volume devices using `vxassist` and other commands.

The procedure described in this section requires a plex that is large enough to store the complete contents of the volume.

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. Run `vxassist snapstart` to create a snapshot mirror.

2. Run `vxassist snapshot` to create a snapshot volume.

The `vxassist snapstart` step creates a write-only backup plex which gets attached to and synchronized with the volume. When synchronized with the volume, the backup plex is ready to be used as a `snapshot` mirror. The end of the update procedure is indicated by the new `snapshot` mirror changing its state to SNAPDONE. This change can be tracked by the `vxassist snapwait` task, which waits until at least one of the mirrors changes its state to SNAPDONE. If the attach process fails, the `snapshot` mirror is removed and its space is released.

Note If the `snapstart` procedure is interrupted, the snapshot mirror is automatically removed when the volume is started.

Once the `snapshot` mirror is synchronized, it continues being updated until it is detached. You can then select a convenient time at which to create a `snapshot` volume as an image of the existing volume. You can also ask users to refrain from using the system during the brief time required to perform the `snapshot` (typically less than a minute). The amount of time involved in creating the `snapshot` mirror is long in contrast to the brief amount of time that it takes to create the `snapshot` volume.

The online backup procedure is completed by running the `vxassist snapshot` command on a volume with a SNAPDONE mirror. This task detaches the finished `snapshot` (which becomes a normal mirror), creates a new normal volume and attaches the `snapshot` mirror to the `snapshot` volume. The `snapshot` then becomes a normal, functioning volume and the state of the `snapshot` is set to ACTIVE.

To back up a volume 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 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. 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 177 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 233 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 "Creating Multiple Snapshots" on page 234 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 with a VxFS file system:

```
# fsck -F vxfs /dev/vx/rdsk/diskgroup/snapshot
```

5. If you require a backup of the data in the snapshot, use an appropriate utility or operating system command to copy the contents of the snapshot to tape, or to some other backup medium.

When the backup is complete, you have the following choices for what to do with the snapshot volume:

- Reattach some or all of the plexes of the snapshot volume with the original volume as described in "Reattaching a Snapshot Volume (vxassist snapback)" on page 234.
- Dissociate the snapshot volume entirely from the original volume as described in "Dissociating a Snapshot Volume (vxassist snapclear)" on page 236. 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
```

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.

The procedure can also be used to speed up the creation of a snapshot volume when a mirrored volume is created with more than two plexes and `init=active` is specified.

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 in the SNAPDONE state, use the following command:

```
# vxplex [-g diskgroup] convert state=SNAPDONE plex
```

A converted plex in the SNAPDONE state 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.

Creating Multiple 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.

Reattaching a Snapshot Volume (vxassist snapback)

Note The information in this section does not apply to RAID-5 volumes.

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.

To merge one snapshot plex with the original volume, use the following command:

```
# vxassist [-g diskgroup] snapback snapshot
```

where *snapshot* is the snapshot copy of the volume.

To merge all snapshot plexes in the snapshot volume with the original volume, use the following command:

```
# vxassist [-g diskgroup] -o allplexes snapback snapshot
```

To merge a specified number of plexes from the snapshot volume with the original volume, use the following command:

```
# vxassist [-g diskgroup] snapback nmirror=number snapshot
```

Here the `nmirror` attribute specifies the number of mirrors in the snapshot volume that are to be re-attached.

Once the snapshot plexes have been reattached and their data resynchronized, they are ready to be used in another `snapshot` operation.

By default, the data in the original volume is used to update the snapshot plexes that have been re-attached. To copy the data from the replica volume instead, use the following command:

```
# vxassist [-g diskgroup] -o resyncfromreplica snapback snapshot
```

Caution Always unmount the snapshot volume (if mounted) before performing a snapback. In addition, you must unmount the file system corresponding to the primary volume before using the `resyncfromreplica` option.

Adding Plexes to a Snapshot Volume

If you want to retain the existing plexes in a snapshot volume after a snapback operation, create additional snapshot plexes that are to be used for the snapback:

1. Use the following `vxprint` commands to discover the names of the snapshot volume's data change object (DCO) and DCO volume:

   ```
   # DCONAME=`vxprint [-g diskgroup] -F%dco_name snapshot`
   # DCOVOL=`vxprint [-g diskgroup] -F%parent_vol $DCONAME`
   ```

2. Use the `vxassist mirror` command to create mirrors of the existing snapshot volume and its DCO volume:

   ```
   # vxassist -g diskgroup mirror snapshot
   # vxassist -g diskgroup mirror $DCOVOL
   ```

   Note The new plex in the DCO volume is required for use with the new data plex in the snapshot.

3. Use the `vxprint` command to find out the name of the additional snapshot plex:

   ```
   # vxprint -g diskgroup snapshot
   ```

4. Use the `vxprint` command to find out the record ID of the additional DCO plex:

   ```
   # vxprint -g diskgroup -F%rid $DCOVOL
   ```

5. Use the `vxedit` command to set the `dco_plex_rid` field of the new data plex to the name of the new DCO plex:

   ```
   # vxedit -g diskgroup set dco_plex_rid=dco_plex_rid new_plex
   ```

The new data plex is now ready to be used to perform a snapback operation.

Dissociating a Snapshot Volume (vxassist snapclear)

The link between a snapshot and its original volume can be permanently broken so that the snapshot volume becomes an independent volume.

Use the following command to dissociate the snapshot volume, *snapshot*:

```
# vxassist snapclear snapshot
```

Displaying Snapshot Information (vxassist snapprint)

The `vxassist snapprint` command displays the associations between the original volumes and their respective replicas (snapshot copies):

```
# vxassist snapprint [volume]
```

Note The `snapprint` command displays an error message if no FastResync maps are enabled for a volume.

Creating and Administering Volume Sets 10

This chapter describes how to use the `vxvset` command to create and administer volume sets in VERITAS Volume Manager (VxVM). Volume sets enable the use of the Multi-Volume Support feature with VERITAS File System (VxFS). It is also possible to use the VERITAS Enterprise Administrator (VEA) to create and administer volumes sets. For more information, see the VEA online help.

For full details of the usage of the `vxvset` command, see the vxvset(1M) manual page.

Note Most VxVM commands require superuser or equivalent privileges.

Please note the following limitation of volume sets:

- A maximum of 256 volumes may be configured in a volume set.
- Only VERITAS File System is supported on a volume set.
- The first volume (index 0) in a volume set must be larger than the sum of the total volume size divided by 4000, the size of the VxFS intent log, and 1MB.
- Raw I/O from and to a volume set is not supported.
- A snapshot of a volume set must itself be a volume set with the same number of volumes and the same volume index numbers as the parent. The corresponding volumes in the parent and snapshot volume sets are also subject to the same restrictions as apply between standalone volumes and their snapshots.

Creating a Volume Set

To create a volume set for use by VERITAS File System (VxFS), use the following command:

```
# vxvset [-g diskgroup] -t vxfs make volset volume
```

Here *volset* is the name of the volume set, and *volume* is the name of the first volume in the volume set. The `-t` option defines the content handler subdirectory for the application that is to be used with the volume. This subdirectory contains utilities that an application uses to operate on the volume set. As the operation of these utilities is determined by the requirements of the application and not by VxVM, it is not discussed further here.

For example, to create a volume set named `myvset` that contains the volume `vol1`, in the disk group `mydg`, you would use the following command:

```
# vxvset -g mydg -t vxfs make myvset vol1
```

Adding a Volume to a Volume Set

Having created a volume set containing a single volume, you can use the following command to add further volumes to the volume set:

```
# vxvset [-g diskgroup] [-f] addvol volset volume
```

For example, to add the volume `vol2`, to the volume set `myvset`, use the following command:

```
# vxvset -g mydg addvol myvset vol2
```

Caution The `-f` (force) option must be specified if the volume being added, or any volume in the volume set, is either a snapshot or the parent of a snapshot. Using this option can potentially cause inconsistencies in a snapshot hierarchy if any of the volumes involved in the operation is already in a snapshot chain.

Listing Details of Volume Sets

To list the details of the component volumes of a volume set, use the following command:

```
# vxvset [-g diskgroup] list [volset]
```

If the name of a volume set is not specified, the command lists the details of all volume sets in a disk group, as shown in the following example:

```
# vxvset -g mydg list
NAME         GROUP     NVOLS     CONTEXT
set1         mydg          3     -
set2         mydg          2     -
```

To list the details of each volume in a volume set, specify the name of the volume set as an argument to the command:

```
# vxvset -g mydg list set1
VOLUME            INDEX         LENGTH     KSTATE  CONTEXT
vol1                  0       12582912    ENABLED  -
vol2                  1       12582912    ENABLED  -
vol3                  2       12582912    ENABLED  -
```

The context field contains details of any string that the application has set up for the volume or volume set to tag its purpose.

Stopping and Starting Volume Sets

Under some circumstances, you may need to stop and restart a volume set. For example, a volume within the set may have become detached, as shown here:

```
# vxvset -g mydg list set1
VOLUME            INDEX         LENGTH     KSTATE  CONTEXT
vol1                  0       12582912   DETACHED  -
vol2                  1       12582912    ENABLED  -
vol3                  2       12582912    ENABLED  -
```

To stop and restart one or more volume sets, use the following commands:

```
# vxvset [-g diskgroup] stop volset ...
# vxvset [-g diskgroup] start volset ...
```

For the example given previously, the effect of running these commands on the component volumes is shown below:

```
# vxvset -g mydg stop set1

# vxvset -g mydg list set1
VOLUME           INDEX        LENGTH     KSTATE  CONTEXT
vol1                 0      12582912   DISABLED  -
vol2                 1      12582912   DISABLED  -
vol3                 2      12582912   DISABLED  -

# vxvset -g mydg start set1

# vxvset -g mydg list set1
VOLUME           INDEX        LENGTH     KSTATE  CONTEXT
vol1                 0      12582912    ENABLED  -
vol2                 1      12582912    ENABLED  -
vol3                 2      12582912    ENABLED  -
```

Removing a Volume from a Volume Set

To remove a component volume from a volume set, use the following command:

```
# vxvset [-g diskgroup] [-f] rmvol volset volume
```

For example, the following commands remove the volumes, `vol1` and `vol2`, from the volume set `myvset`:

```
# vxvset -g mydg rmvol myvset vol1
# vxvset -g mydg rmvol myvset vol2
```

Note When the final volume is removed, this deletes the volume set.

Caution The `-f` (force) option must be specified if the volume being removed, or any volume in the volume set, is either a snapshot or the parent of a snapshot. Using this option can potentially cause inconsistencies in a snapshot hierarchy if any of the volumes involved in the operation is already in a snapshot chain.

Administering Hot-Relocation 11

If a volume has a disk I/O failure (for example, the disk has an uncorrectable error), VERITAS Volume Manager (VxVM) can detach the plex involved in the failure. I/O stops on that plex but continues on the remaining plexes of the volume.

If a disk fails completely, VxVM can detach the disk from its disk group. All plexes on the disk are disabled. If there are any unmirrored volumes on a disk when it is detached, those volumes are also disabled.

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 section, "How Hot-Relocation Works" on page 242.

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 258) 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:

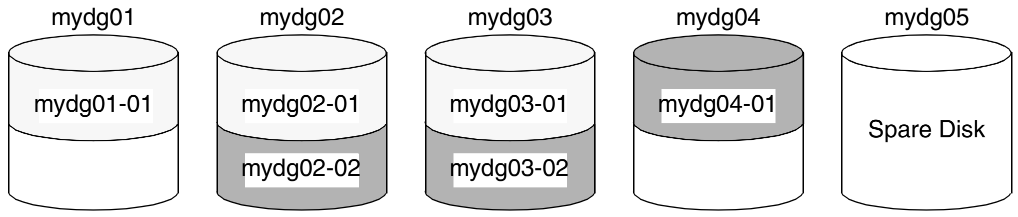
- The failing subdisks are on non-redundant volumes (that is, volumes of types other than mirrored or RAID-5).
- There are insufficient spare disks or free disk space in the disk group.
- The only available space is on a disk that already contains a mirror of the failing plex.
- The only available space 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 `vxrelocd`(1M) manual page for more information about the hot-relocation daemon.

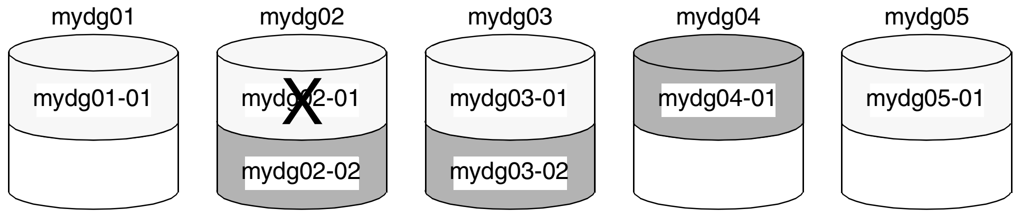
"Example of Hot-Relocation for a Subdisk in a RAID-5 Volume" on page 244 illustrates the hot-relocation process in the case of the failure of a single subdisk of a RAID-5 volume.

Example of Hot-Relocation for a Subdisk in a RAID-5 Volume

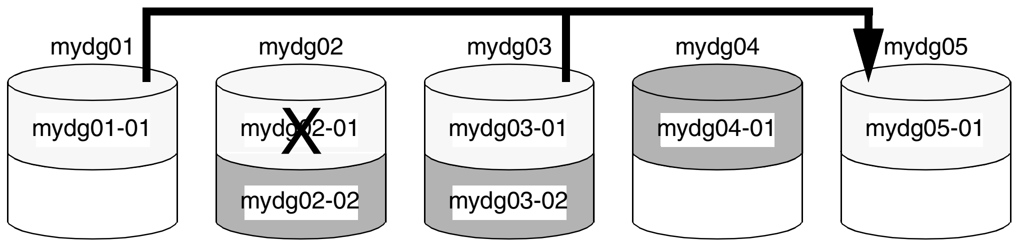
a) Disk group contains five disks. Two RAID-5 volumes are configured across four of the disks.
One spare disk is available for hot-relocation.



b) Subdisk mydg02-01 in one RAID-5 volume fails. Hot-relocation replaces it with subdisk mydg05-0
that it has created on the spare disk, and then initiates recovery of the RAID-5 volume.



c) RAID-5 recovery recreates subdisk mydg02-01's data and parity on subdisk mydg05-01 from
the data and parity information remaining on subdisks mydg01-01 and mydg03-01.



Partial Disk Failure Mail Messages

If hot-relocation is enabled when a plex or disk is detached by a failure, mail indicating the failed objects is sent to `root`. If a partial disk failure occurs, the mail identifies the failed plexes. For example, if a disk containing mirrored volumes fails, you can receive mail information as shown in the following example:

```
To: root
Subject: Volume Manager failures on host teal
Failures have been detected by the VERITAS Volume Manager:

failed plexes:
home-02
src-02
```

See "Modifying the Behavior of Hot-Relocation" on page 258 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 -g mydg -s -ff home-02 src-02
```

The `-s` option asks for information about individual subdisks, and the `-ff` option displays the number of failed read and write operations. The following output display is typical:

```
                        FAILED
TYP NAME              READS   WRITES
sd mydg01-04              0        0
sd mydg01-06              0        0
sd mydg02-03              1        0
sd mydg02-04              1        0
```

This example shows failures on reading from subdisks `mydg02-03` and `mydg02-04` of disk `mydg02`.

Hot-relocation automatically relocates the affected subdisks and initiates any necessary recovery procedures. However, if relocation is not possible or the hot-relocation feature is disabled, you must investigate the problem and attempt to recover the plexes. Errors can be caused by cabling failures, so check the cables connecting your disks to your system. If there are obvious problems, correct them and recover the plexes using the following command:

```
# vxrecover -b -g mydg home src
```

This starts recovery of the failed plexes in the background (the command prompt reappears before the operation completes). If an error message appears later, or if the plexes become detached again and there are no obvious cabling failures, replace the disk (see "Removing and Replacing Disks" on page 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:
mydg02

failed plexes:
home-02
src-02
mkting-01

failing disks:
mydg02
```

This message shows that `mydg02` was detached by a failure. When a disk is detached, I/O cannot get to that disk. The plexes `home-02`, `src-02`, and `mkting-01` were also detached (probably because of the failure of the disk).

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.

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 249.
- To remove a disk from use as a hot-relocation spare, see "Removing a Disk from Use as a Hot-Relocation Spare" on page 250.

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 251.
- To make a disk available for hot-relocation use, see "Making a Disk Available for Hot-Relocation Use" on page 252.

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 253).

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 [-g diskgroup] spare
```

The following is example output:

```
GROUP   DISK     DEVICE    TAG       OFFSET  LENGTH   FLAGS
mydg    mydg02   c0t2d0s2  c0t2d0s2  0       658007   s
```

Here `mydg02` is the only disk designated as a spare in the `mydg` disk group. The `LENGTH` field indicates how much spare space is currently available on `mydg02` for relocation.

The following commands can also be used to display information about disks that are currently designated as spares:

- `vxdisk list` lists disk information and displays spare disks with a `spare` flag.
- `vxprint` lists disk and other information and displays spare disks with a `SPARE` flag.
- The `list` menu item on the `vxdiskadm` main menu lists all disks including spare disks.

Marking a Disk as a Hot-Relocation Spare

Hot-relocation allows the system to react automatically to I/O failure by relocating redundant subdisks to other disks. Hot-relocation then restores the affected VxVM objects and data. If a disk has already been designated as a spare in the disk group, the subdisks from the failed disk are relocated to the spare disk. Otherwise, any suitable free space in the disk group is used 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 [-g diskgroup] set spare=on diskname
```

where *diskname* is the disk media name.

For example, to designate `mydg01` as a spare in the disk group, `mydg`, enter the following command:

```
# vxedit -g mydg set spare=on mydg01
```

You can use the `vxdisk list` command to confirm that this disk is now a spare; `mydg01` should be listed with a `spare` flag.

Any VM disk in this disk group can now use this disk as a spare in the event of a failure. If a disk fails, hot-relocation automatically occurs (if possible). You are notified of the failure and relocation through electronic mail. After successful relocation, you may want to replace the failed disk.

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 media name (such as `mydg01`):

   ```
   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,?] mydg01
   ```

   The following notice is displayed when the disk has been marked as spare:

   ```
   VxVM NOTICE V-5-2-219 Marking of mydg01 in mydg as a spare disk is
   complete.
   ```

3. At the following prompt, indicate whether you want to add more disks as spares (**y**) or return to the `vxdiskadm` main menu (**n**):

   ```
   Mark another disk as a spare? [y,n,q,?] (default: n)
   ```

   Any VM disk in this disk group can now use this disk as a spare in the event of a failure. If a disk fails, hot-relocation should automatically occur (if possible). You should be notified of the failure and relocation through electronic mail. After successful relocation, you may want to replace the failed disk.

Removing a Disk from Use as a Hot-Relocation Spare

While a disk is designated as a spare, the space on that disk is not used for the creation of VxVM objects within its disk group. If necessary, you can free a spare disk for general use by removing it from the pool of hot-relocation disks.

To remove a spare from the hot-relocation pool, use the following command:

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

where *diskname* is the disk media name.

For example, to make `mydg01` available for normal use in the disk group, `mydg`, use the following command:

```
# vxedit -g mydg set spare=off mydg01
```

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 disk media name of a spare disk (such as `mydg01`):

   ```
   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,?] mydg01
   ```

   The following confirmation is displayed:

   ```
   VxVM NOTICE V-5-2-143 Disk mydg01 in mydg no longer marked as a
   spare disk.
   ```

3. At the following prompt, indicate whether you want to disable more spare disks (**y**) or return to the `vxdiskadm` main menu (**n**):

   ```
   Turn-off spare flag on another disk? [y,n,q,?] (default: n)
   ```

Excluding a Disk from Hot-Relocation Use

To exclude a disk from hot-relocation use, use the following command:

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

where *diskname* is the disk media name.

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 media name (such as `mydg01`):

   ```
   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,?] mydg01
   ```

   The following confirmation is displayed:

   ```
   VxVM INFO V-5-2-925 Excluding mydg01 in mydg from hot-relocation
   use is complete.
   ```

3. At the following prompt, indicate whether you want to add more disks to be excluded from hot-relocation (**y**) or return to the `vxdiskadm` main menu (**n**):

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

Making a Disk Available for Hot-Relocation Use

Free space is used automatically by hot-relocation in case spare space is not sufficient to relocate failed subdisks. You can limit this free space usage by hot-relocation by specifying which free disks should not be touched by hot-relocation. If a disk was previously excluded from hot-relocation use, you can undo the exclusion and add the disk back to the hot-relocation pool.

To make a disk available for hot-relocation use, use the following command:

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

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 media name (such as `mydg01`):

```
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,?] mydg01
```

   The following confirmation is displayed:

```
V-5-2-932 Making mydg01 in mydg available for hot-relocation use is
complete.
```

3. At the following prompt, indicate whether you want to add more disks to be excluded from hot-relocation (**`y`**) or return to the `vxdiskadm` main menu (**`n`**):

```
Make another disk available for hot-relocation use? [y,n,q,?]
(default: n)
```

Configuring Hot-Relocation to Use Only Spare Disks

If you want VxVM to use only spare disks for hot-relocation, add the following line to the file `/etc/default/vxassist`:

```
spare=only
```

If not enough storage can be located on disks marked as spare, the relocation fails. Any free space on non-spare disks is not used.

Moving 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 mydg02-03 from plex home-02.
Dev_offset 0 length 1164 dm_name mydg02 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 mydg02-03 relocated to mydg05-01,
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 mydg10 back to mydg10 of disk group mydg.
   Continue with operation? [y,n,q,?] (default: y)
   ```

   A status message is displayed at the end of the operation.

   ```
   VxVM INFO V-5-2-954 Unrelocate to disk mydg10 is complete.
   ```

As an alternative to this procedure, use either the `vxassist` command or the `vxunreloc` command directly, as described in "Moving and Unrelocating Subdisks Using vxassist" on page 255 and "Moving and Unrelocating Subdisks Using vxunreloc" on page 255.

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 `mydg05` belonging to the volume `home` back to `mydg02`, enter the following command:

```
# vxassist -g mydg move home !mydg05 mydg02
```

Here, `!mydg05` specifies the current location of the subdisks, and `mydg02` specifies where the subdisks should be relocated.

If the volume is enabled, subdisks within detached or disabled plexes, and detached log or RAID-5 subdisks, are moved without recovery of data.

If the volume is not enabled, subdisks within STALE or OFFLINE plexes, and stale log or RAID-5 subdisks, are moved without recovery. If there are other subdisks within a non-enabled volume that require moving, the relocation fails.

For enabled subdisks in enabled plexes within an enabled volume, data is moved to the new location, without loss of either availability or redundancy of the volume.

Moving 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 examples in the following sections demonstrate the use of `vxunreloc`.

Moving Hot-Relocated Subdisks back to their Original Disk

Assume that `mydg01` failed and all the subdisks were relocated. After `mydg01` is replaced, `vxunreloc` can be used to move all the hot-relocated subdisks back to `mydg01`.

```
# vxunreloc -g mydg mydg01
```

Moving Hot-Relocated Subdisks to a Different Disk

The `vxunreloc` utility provides the `-n` option to move the subdisks to a different disk from where they were originally relocated.

Assume that `mydg01` failed, and that all of the subdisks that resided on it were hot-relocated to other disks. `vxunreloc` provides an option to move the subdisks to a different disk from where they were originally relocated. After the disk is repaired, it is added back to the disk group using a different name, e.g, `mydg05`. If you want to move all the hot-relocated subdisks back to the new disk, the following command can be used:

```
# vxunreloc -g mydg -n mydg05 mydg01
```

The destination disk should have at least as much storage capacity as was in use on the original disk. If there is not enough space, the unrelocate operation will fail and none of the subdisks will be moved.

Forcing Hot-Relocated Subdisks to Accept Different Offsets

By default, `vxunreloc` attempts to move hot-relocated subdisks to their original offsets. However, `vxunreloc` fails if any subdisks already occupy part or all of the area on the destination disk. In such a case, you have two choices:

- Move the existing subdisks somewhere else, and then re-run `vxunreloc`.
- Use the `-f` option provided by `vxunreloc` to move the subdisks to the destination disk, but leave it to `vxunreloc` to find the space on the disk. As long as the destination disk is large enough so that the region of the disk for storing subdisks can accommodate all subdisks, all the hot-relocated subdisks will be unrelocated without using the original offsets.

Assume that `mydg01` failed and the subdisks were relocated and that you want to move the hot-relocated subdisks to `mydg05` where some subdisks already reside. You can use the force option to move the hot-relocated subdisks to `mydg05`, but not to the exact offsets:

```
# vxunreloc -g mydg -f -n mydg05 mydg01
```

Examining Which Subdisks Were Hot-Relocated from a Disk

If a subdisk was hot relocated more than once due to multiple disk failures, it can still be unrelocated back to its original location. For instance, if `mydg01` failed and a subdisk named `mydg01-01` was moved to `mydg02`, and then `mydg02` experienced disk failure, all of the subdisks residing on it, including the one which was hot-relocated to it, will be moved again. When `mydg02` was replaced, a `vxunreloc` operation for `mydg02` will do nothing to the hot-relocated subdisk `mydg01-01`. However, a replacement of `mydg01` followed by a `vxunreloc` operation, moves `mydg01-01` back to `mydg01` if `vxunreloc` is run immediately after the replacement.

After the disk that experienced the failure is fixed or replaced, `vxunreloc` can be used to move all the hot-relocated subdisks back to the disk. When a subdisk is hot-relocated, its original disk-media name and the offset into the disk, are saved in the configuration database. When a subdisk is moved back to the original disk or to a new disk using `vxunreloc`, the information is erased. The original disk-media name and the original offset are saved in the subdisk records. To print all of the subdisks that were hot-relocated from `mydg01` in the `mydg` disk group, use the following command:

```
# vxprint -g mydg -se 'sd_orig_dmname="mydg01"'
```

Restarting vxunreloc After Errors

`vxunreloc` moves subdisks in three phases:

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 the `vxrelocd` process is running. You should normally leave hot-relocation turned on so that you can take advantage of this feature if a failure occurs. However, if you choose to disable hot-relocation (perhaps because you do not want the free space on your disks to be used for relocation), you can prevent `vxrelocd` from starting at system startup time by editing the startup file that invokes vxrelocd: `/lib/svc/method/vxvm-recover` in Solaris 10, or `/etc/init.d/vxvm-recover` in previous releases of the Solaris OS.

You can alter the behavior of `vxrelocd` 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.

On a Solaris 10 system, after making changes to the way `vxrelocd` is invoked in the startup file, run the following command to notify that the service configuration has changed:

```
# svcadm refresh vxvm/vxvm-recover
```

For previous releases of the Solaris OS, reboot the system.

You can also stop hot-relocation at any time by killing the `vxrelocd` process (this should not be done while a hot-relocation attempt is in progress).

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 &
```

Alternatively, you can use the following command:

```
# nohup /etc/vx/bin/vxrelocd root user1 user2 &
```

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

Using VERITAS Storage Expert 12

System administrators often find that gathering and interpreting data about large and complex configurations can be a difficult task. VERITAS Storage Expert (*vxse*) is designed to help in diagnosing configuration problems with VxVM.

Storage Expert consists of a set of simple commands that collect VxVM configuration data and compare it with “best practice.” Storage Expert then produces a summary report that shows which objects do not meet these criteria and makes recommendations for VxVM configuration improvements.

These user-configurable tools help you as an administrator to verify and validate systems and non-optimal configurations in both small and large VxVM installations.

See the following sections for more information about VERITAS Storage Expert:

- How Storage Expert Works
- Before Using Storage Expert
- Running Storage Expert
- Identifying Configuration Problems Using Storage Expert
- Rule Definitions and Attributes

For more information about Storage Expert, see the `vxse`(1M) manual page.

How Storage Expert Works

Storage Expert components include a set of rule scripts and a rules engine. The rules engine runs the scripts and produces ASCII output, which is organized and archived by Storage Expert's report generator. This output contains information about areas of VxVM configuration that do not meet the set criteria. By default, output is sent to the screen, but you can send it to a file using standard output redirection.

Before Using Storage Expert

Storage Expert is included in the `VRTSvxvm` package. Even if you do not plan to use the VEA graphical user interface, you must also have installed the following packages to run vxse:

- `VRTSob`
- `VRTSvmpro`

The VEA service must also be started on the system by running the command `/opt/VRTS/bin/vxsvc`. For information about installing these components and starting the VEA service, see the *Installation Guide*.

Running Storage Expert

Note You must have `root` user privileges to run Storage Expert.

Command Line Syntax

Note The executable rule files are located in the directory, `/opt/VRTS/vxse/vxvm`. The examples in this chapter assume that this directory has been added to the `PATH` variable.

The rules are invoked using the following syntax:

```
# rulename [options] keyword [attribute=value ...]
```

Each of the rules performs a different function as listed in "Rule Definitions and Attributes" on page 273.

The following options may be specified:

`-d` *defaults_file*	Specify an alternate defaults file.
`-g` *diskgroup*	Specify the disk group to be examined.
`-v`	Specify verbose output format.

One of the following keywords must be specified:

`check`	List the default values used by the rule's attributes.
`info`	Describe what the rule does.
`list`	List the attributes of the rule that you can set.
`run`	Run the rule.

A full list of the Storage Expert rules and their default values are listed in "Rule Definitions and Attributes" on page 273.

Discovering What a Rule Does

To obtain details about what a rule does, use the `info` keyword, as in the following example:

```
# vxse_stripes2 info
VxVM vxse:vxse_stripes2 INFO V-5-1-6153 This rule checks for stripe
volumes which have too many or too few columns
```

Displaying Rule Attributes and Their Default Values

To see the attributes that are available for a given rule, use the `list` keyword. In the following example, the single attribute, `mirror_threshold`, is shown for the rule `vxse_drl1`:

```
# vxse_drl1 list
VxVM vxse:vxse_drl1 INFO V-5-1-6004
vxse_drl1 - T UNEABLES default values
-----------------------------------------------------------
      mirror_threshold - large mirror threshold size
                         Warn if a mirror is of greater
                         than this size and does not have
                         an attached DRL log.
```

To see the default values of a specified rule's attributes, use the `check` keyword as shown here:

```
# vxse_stripes2 check
vxse_stripes2 - T UNEABLES
----------------------------------------------------------
VxVM vxse:vxse_stripes2 INFO V-5-1-5546
        too_wide_stripe - (1 6) columns in a striped volume
        too_narrow_stripe - (3) columns in a striped volume
```

Storage Expert lists the default value of each of the rule's attributes.

A full list of rules and the default values of their attributes can be found in "Rule Definitions and Attributes" on page 273.

To alter the behavior of rules, you can change the value of their attributes as described in the section, "Setting Rule Attributes" on page 265.

Running a Rule

The `run` keyword invokes a default or reconfigured rule on a disk group or file name, for example:

```
# vxse_dg1 -g mydg run
VxVM vxse:vxse_dg1 INFO V-5-1-5511 vxse_vxdg1 - RESULTS
----------------------------------------------------------

vxse_dg1 PASS:
Disk group (mydg) okay amount of disks in this disk group (4)
```

This indicates that the specified disk group (`mydg`) met the conditions specified in the rule. See "Rule Result Types" on page 265 for a list of the possible result types.

Note You can set Storage Expert to run as a `cron` job to notify administrators and automatically archive reports.

Rule Result Types

Running a rule generates output that shows the status of the objects that have been examined against the rule:

`INFO`	Information about the specified object; for example “RAID-5 does not have a log.”
`PASS`	The object met the conditions of the rule.
`VIOLATION`	The object did not meet the conditions of the rule.

Setting Rule Attributes

You can set attributes in the following ways:

- Enter an attribute on the command line, for example:

  ```
  # vxse_drl2 -g mydg run large_mirror_size=30m
  ```

- Create your own defaults file, and specify that file on the command line:

  ```
  # vxse_drl2 -d mydefaultsfile run
  ```

  Lines in this file contain attribute values definitions for a rule in this format:

  `rule_name``,`*`attribute`*`=`*`value`*

  For example, the following entry defines a value of 20 gigabytes for the attribute `large_mirror_size` of the rule `vxse_drl2`:

  ```
  vxse_drl2,large_mirror_size=20g
  ```

  You can specify values that are to be ignored by inserting a # character at the start of the line, for example:

  ```
  #vxse_drl2,large_mirror_size=20g
  ```

- Edit the attribute values that are defined in the `/etc/default/vxse` file. If you do this, make a backup copy of the file in case you need to regress your changes.

Attributes are applied using the following order of precedence from highest to lowest:

1. A value specified on the command line.
2. A value specified in a user-defined defaults file.
3. A value in the `/etc/default/vxse` file that has not been commented out.
4. A built-in value defined at compile time.

Identifying Configuration Problems Using Storage Expert

Storage Expert provides a large number of rules that help you to diagnose configuration issues that might cause problems for your storage environment. Each rule describes the issues involved, and suggests remedial actions.

The rules help you to diagnose problems in the following categories:

- Recovery Time
- Disk Groups
- Disk Striping
- Disk Sparing and Relocation Management
- Hardware Failures
- Rootability
- System Hostname

A full list of Storage Expert rules, listed in numerical order, can be found in "Rule Definitions and Attributes" on page 273.

Recovery Time

Several "best practice" rules enable you to check that your storage configuration has the resilience to withstand a disk failure or a system failure.

Checking for Multiple RAID-5 Logs on a Physical Disk (vxse_disklog)

To check whether more than one RAID-5 log exists on the same physical disk, run rule `vxse_disklog`.

RAID-5 log mirrors for the same physical volume should be located on separate physical disks to ensure redundancy. More than one RAID-5 log on a disk also makes the recovery process longer and more complicated.

Checking for Large Mirror Volumes Without a DRL (vxse_drl1)

To check whether large mirror volumes (larger than 1GB) have an associated dirty region log (DRL), run rule `vxse_drl1`.

Creating a DRL speeds recovery of mirrored volumes after a system crash. A DRL tracks those regions that have changed and uses the tracking information to recover only those portions of the volume that need to be recovered. Without a DRL, recovery is accomplished by copying the full contents of the volume between its mirrors. This process is lengthy and I/O intensive.

For information on adding a DRL log to a mirrored volume, see "Adding Dirty Region Logging to a Mirrored Volume" on page 206.

Checking for Large Mirrored Volumes Without a Mirrored DRL (vxse_drl2)

To check whether a large mirrored volume has a mirrored DRL log, run rule `vxse_drl2`.

Mirroring the DRL log provides added protection in the event of a disk failure.

For information on adding a mirror to a DRL log, see "Adding Dirty Region Logging to a Mirrored Volume" on page 206.

Checking for RAID-5 Volumes Without a RAID-5 Log (vxse_raid5log1)

To check whether a RAID-5 volume has an associated RAID-5 log, run rule `vxse_raid5log1`.

In the event of both a system failure and a failure of a disk in a RAID-5 volume, data that is not involved in an active write could be lost or corrupted if there is no RAID-5 log.

For information about adding a RAID-5 log to a RAID-5 volume, see "Adding a RAID-5 Log" on page 210.

Checking Minimum and Maximum RAID-5 Log Sizes (vxse_raid5log2)

To check that the size of RAID-5 logs falls within the minimum and maximum recommended sizes, run rule `vxse_raid5log2`.

The recommended minimum and maximum sizes are 64MB and 1GB respectively. If `vxse_raid5log2` reports that the size of the log is outside these boundaries, adjust the size by replacing the log.

Checking for Non-Mirrored RAID-5 Logs (vxse_raid5log3)

To check that the RAID-5 log of a large volume is mirrored, run the `vxse_raid5log3` rule.

A mirror of the RAID-5 log protects against loss of data due to the failure of a single disk. You are strongly advised to mirror the log if `vxse_raid5log3` reports that the log of a large RAID-5 volume does not have a mirror.

For information on adding a RAID-5 log mirror, see "Adding a RAID-5 Log" on page 210.

Disk Groups

Disks groups are the basis of VxVM storage configuration so it is critical that the integrity and resilience of your disk groups are maintained. Storage Expert provides a number of rules that enable you to check the status of disk groups and associated objects.

Checking Whether a Configuration Database Is Too Full (vxse_dg1)

To check whether the disk group configuration database has become too full, run rule `vxse_dg1`.

By default, this rule suggests a limit of 250 for the number of disks in a disk group. If one of your disk groups exceeds this figure, you should consider creating a new disk group. The number of objects that can be configured in a disk group is limited by the size of the private region which stores configuration information about every object in the disk group. Each disk in the disk group that has a private region stores a separate copy of this configuration database.

For information on creating a new disk group, see "Creating a Disk Group" on page 128.

Checking Disk Group Configuration Copies and Logs (vxse_dg2)

To check whether a disk group has too many or too few disk group configuration copies, and whether a disk group has too many or too few copies of the disk group log, run rule `vxse_dg2`.

Checking "on disk config" Size (vxse_dg3)

To check whether a disk group has the correct "on disk config" size, run rule `vxse_dg3`.

Checking Version Number of Disk Groups (vxse_dg4)

To check the version number of a disk group, run rule `vxse_dg4`.

For optimum results, your disk groups should have the latest version number that is supported by the installed version of VxVM.

If a disk group is not at the latest version number, see the section "Upgrading a Disk Group" on page 141 for information about upgrading it.

Checking the Number of Configuration Copies in a Disk Group (vxse_dg5)

To find out whether a disk group has only a single VxVM configured disk, run rule `vxse_dg5`.

See "Creating and Administering Disk Groups" on page 123 for more information.

Checking for Non-Imported Disk Groups (vxse_dg6)

To check for disk groups that are visible to VxVM but not imported, run rule `vxse_dg6`.

Importing a disk to a disk group is described in "Importing a Disk Group" on page 132.

Checking for Initialized VM Disks that are not in a Disk Group (vxse_disk)

To find out whether there are any initialized disks that are not a part of any disk group, run rule `vxse_disk`. This prints out a list of disks, indicating whether they are part of a disk group or unassociated.

For information on how to add a disk to disk group, see "Adding a Disk to a Disk Group" on page 129.

Checking Volume Redundancy (vxse_redundancy)

To check whether a volume is redundant, run rule `vxse_redundancy`.

This rule displays a list of volumes together with the number of mirrors that are associated with each volume. If `vxse_redundancy` shows that a volume does not have an associated mirror, your data is at risk in the event of a disk failure, and you should rectify the situation by creating a mirror for the volume.

See "Adding a Mirror to a Volume" on page 203 for information on adding a mirror to a volume.

Checking States of Plexes and Volumes (vxse_volplex)

To check whether your disk groups contain unused objects (such as plexes and volumes), run rule `vxse_volplex`. In particular, this rule notifies you if any of the following conditions exist:

- disabled plexes
- detached plexes
- stopped volumes
- disabled volumes
- disabled logs
- failed plexes
- volumes needing recovery

If any of these conditions exist, see the following for information on correcting the situation:

- To re-enable a disabled or detached plex, see "Reattaching Plexes" on page 165.
- To re-enable a stopped or disabled volume, see "Starting a Volume" on page 203.
- To recover a volume, see the chapter "Recovery from Hardware Failure" in the *VERITAS Volume Manager Troubleshooting Guide*.

Disk Striping

Striping enables you to enhance your system's performance. Several rules enable you to monitor important parameters such as the number of columns in a stripe plex or RAID-5 plex, and the stripe unit size of the columns.

Checking the Configuration of Large Mirrored-Stripe Volumes (vxse_mirstripe)

To check whether large mirror-striped volumes should be reconfigured as striped-mirror volumes, run rule `vxse_mirstripe`.

A large mirrored-striped volume should be reconfigured, using relayout, as a striped-mirror volume to improve redundancy and enhance recovery time after failure.

To convert a mirrored-striped volume to a striped-mirror volume, see "Converting Between Layered and Non-Layered Volumes" on page 225.

Checking the Number of Columns in RAID-5 Volumes (vxse_raid5)

To check whether RAID-5 volumes have too few or too many columns, run rule `vxse_raid5`.

By default, this rule assumes that a RAID-5 plex should have more than 4 columns and fewer than 8 columns.

See "Performing Online Relayout" on page 219 for information on changing the number of columns.

Checking the Stripe Unit Size of Striped Volumes (vxse_stripes1)

By default, rule `vxse_stripes1` reports a violation if a volume's stripe unit size is not set to an integer multiple of 8KB.

See "Performing Online Relayout" on page 219 for information on changing the stripe unit size.

Checking the Number of Columns in Striped Volumes (vxse_stripes2)

The default values for the number of columns in a striped plex are 16 and 3. By default, rule `vxse_stripes2` reports a violation if a striped plex in your volume has fewer than 3 columns or more than 16 columns.

See "Performing Online Relayout" on page 219 for information on changing the number of columns in a striped volume.

Disk Sparing and Relocation Management

The hot-relocation feature of VxVM uses spare disks in a disk group to recreate volume redundancy after disk failure.

Checking the Number of Spare Disks in a Disk Group (vxse_spares)

This "best practice" rule assumes that between 10% and 20% of disks in a disk group should be allocated as spare disks. By default, `vxse_spares` checks that a disk group falls within these limits.

See "Administering Hot-Relocation" on page 241 for information on managing the pool of spare disks.

Hardware Failures

Checking for Disk and Controller Failures (vxse_dc_failures)

Rule `vxse_dc_failures` can be used to discover if the system has any failed disks or disabled controllers.

Rootability

Checking the Validity of Root Mirrors (vxse_rootmir)

Rule `vxse_rootmir` can be used to confirm that the root mirrors are set up correctly.

System Hostname

Checking the System Name (vxse_host)

Rule `vxse_host` can be used to confirm that the system name (*hostname*) in the file `/etc/vx/volboot` is the same as the name that was assigned to the system when it was booted.

Rule Definitions and Attributes

The tables in this section list rule definitions, and rule attributes and their default values.

Note You can use the `info` keyword to show a description of a rule. See "Discovering What a Rule Does" on page 263 for details.

Rule Definitions

Rule	Description
vxse_dc_failures	Checks and points out failed disks and disabled controllers.
vxse_dg1	Checks for disk group configurations in which the disk group has become too large.
vxse_dg2	Checks for disk group configurations in which the disk group has too many or too few disk group configuration copies, and if the disk group has too many or too few disk group log copies.
vxse_dg3	Checks disk group configurations to verify that the disk group has the correct "on disk config" size.
vxse_dg4	Checks for disk groups that do not have a current version number, and which may need to be upgraded.
vxse_dg5	Checks for disk groups in which there is only one VxVM configured disk.
vxse_dg6	Checks for disk groups that are seen, but which are not imported.
vxse_disk	Checks for disks that are initialized, but are not part of any disk group.
vxse_disklog	Checks for physical disks that have more than one RAID-5 log.
vxse_drl1	Checks for large mirror volumes that do not have an associated DRL log.
vxse_drl2	Checks for large mirror volumes that do not have DRL log that is mirrored.
vxse_host	Checks that the system "hostname" in the `/etc/vx/volboot` file matches the hostname that was assigned to the system when it was booted.
vxse_mirstripe	Checks for large mirror-striped volumes that should be striped-mirrors.
vxse_raid5	Checks for RAID-5 volumes that are too narrow or too wide.

Rule Definitions

Rule	Description
vxse_raid5log1	Checks for RAID-5 volumes that do not have an associated log.
vxse_raid5log2	Checks for recommended minimum and maximum RAID-5 log sizes.
vxse_raid5log3	Checks for large RAID-5 volumes that do not have a mirrored RAID-5 log.
vxse_redundancy	Checks the redundancy of volumes.
vxse_rootmir	Checks that all root mirrors are set up correctly.
vxse_spares	Checks that the number of spare disks in a disk group is within the VxVM "Best Practices" thresholds.
vxse_stripes1	Checks for stripe volumes whose stripe unit is not a multiple of the default stripe unit size.
vxse_stripes2	Checks for stripe volumes that have too many or too few columns.
vxse_volplex	Checks for volumes and plexes that are in various states, such as: ◆ disabled plexes ◆ detached plexes ◆ stopped volumes ◆ disabled volumes ◆ disabled logs ◆ failed plexes ◆ volumes needing recovery

Note You can use the `list` and `check` keywords to show what attributes are available for a rule and to display the default values of these attributes. See "Running a Rule" on page 264 for more information.

Rule Attributes and Default Attribute Values

Rule	Attribute	Default Value	Description
vxse_dc_failures	-	-	No user-configurable variables.
vxse_dg1	max_disks_per_dg	250	Maximum number of disks in a disk group. Warn if a disk group has more disks than this.
vxse_dg2	-	-	No user-configurable variables.
vxse_dg3	-	-	No user-configurable variables.
vxse_dg4	-	-	No user-configurable variables.
vxse_dg5	-	-	No user-configurable variables.
vxse_dg6	-	-	No user-configurable variables.
vxse_disk	-	-	No user-configurable variables.
vxse_disklog	-	-	No user-configurable variables.
vxse_drl1	mirror_threshold	1g (1GB)	Large mirror threshold size. Warn if a mirror is larger than this and does not have an attached DRL log.
vxse_drl2	large_mirror_size	20g (20GB)	Large mirror-stripe threshold size. Warn if a mirror-stripe volume is larger than this.
vxse_host	-	-	No user-configurable variables.
vxse_mirstripe	large_mirror_size	1g (1GB)	Large mirror-stripe threshold size. Warn if a mirror-stripe volume is larger than this.
	nsd_threshold	8	Large mirror-stripe number of subdisks threshold. Warn if a mirror-stripe volume has more subdisks than this.

Rule Attributes and Default Attribute Values

Rule	Attribute	Default Value	Description
`vxse_raid5`	`too_narrow_raid5`	4	Minimum number of RAID-5 columns. Warn if actual number of RAID-5 columns is less than this.
	`too_wide_raid5`	8	Maximum number of RAID-5 columns. Warn if the actual number of RAID-5 columns is greater than this.
`vxse_raid5log1`	-	-	No user-configurable variables.
`vxse_raid5log2`	`r5_max_size`	1g (1GB)	Maximum RAID-5 log check size. Warn if a RAID-5 log is larger than this.
	`r5_min_size`	64m (64MB)	Minimum RAID-5 log check size. Warn if a RAID-5 log is smaller than this.
`vxse_raid5log3`	`large_vol_size`	20g (20GB)	Large RAID-5 volume threshold size. Warn if a RAID-5 volume with a non-mirrored RAID-5 log is larger than this.
`vxse_redundancy`	`volume_redundancy`	0	Volume redundancy check. The value of 2 performs a mirror redundancy check. A value of 1 performs a RAID-5 redundancy check. The default value of 0 performs no redundancy check.
`vxse_rootmir`	-	-	No user-configurable variables.
`vxse_spares`	`max_disk_spare_ratio`	20	Maximum percentage of spare disks in a disk group. Warn if the percentage of spare disks is greater than this.
	`min_disk_spare_ratio`	10	Minimum percentage of spare disks in a disk group. Warn if the percentage of spare disks is less than this.

Rule Attributes and Default Attribute Values

Rule	Attribute	Default Value	Description
`vxse_stripes1`	`default _stripeunit`	8k (8KB)	Stripe unit size for stripe volumes. Warn if a stripe does not have a stripe unit which is an integer multiple of this value.
`vxse_stripes2`	`too_narrow_stripe`	3	Minimum number of columns in a striped plex. Warn if a striped volume has fewer columns than this.
	`too_wide_stripe`	16	Maximum number of columns in a striped plex. Warn if a striped volume has more columns than this.
`vxse_volplex`	-	-	No user-configurable variables.

Performance Monitoring and Tuning 13

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 the 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 the drives. For example, if a single file system receives the most disk accesses, moving the file system to another drive also moves the bottleneck 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.

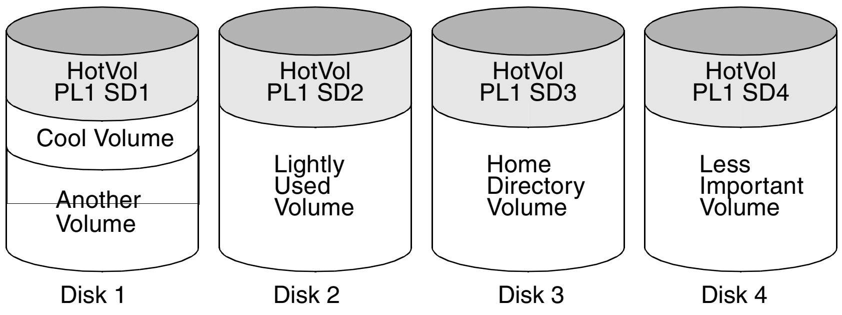
Striping

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" on page 280, 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 27). 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:

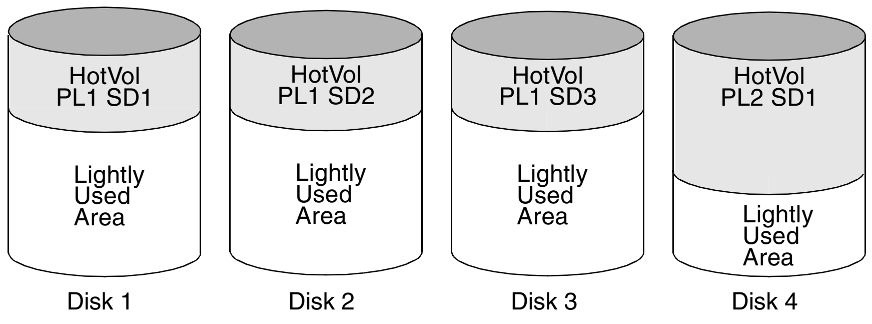
- `round`—a *round-robin* read policy, where all plexes in the volume take turns satisfying read requests to the volume.
- `prefer`—a *preferred-plex* read policy, where the plex with the highest performance usually satisfies read requests. If that plex fails, another plex is accessed.
- `select`—default 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 215.

In the configuration example shown in the figure, "Use of Mirroring and Striping for Improved Performance" on page 282, 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  mydg01        40473  174045     455898   951379       29.5     35.4
dm  mydg02        32668   16873     470337   351351       35.2    102.9
dm  mydg03        55249   60043     780779   731979       35.3     61.2
dm  mydg04        11909   13745     114508   128605       25.0    30.7
```

If you need to move the volume named `archive` onto another disk, use the following command to identify on which disks it lies:

```
# vxprint -g mydg -tvh archive
```

The following is an extract from a typical display:

```
V  NAME     RVG/VSET/COKSTATE    STATE    LENGTH   READPOL REFPLEXUTYPE
PL NAME     VOLUME      STATE    STATE    LENGTH   LAYOUT  NCOL/WDTHMODE
SD NAME     PLEX        DISK     DISKOFFSLENGTH   [COL/]OFFDEVICEMODE

v  archive -            ENABLED  ACTIVE   20480    SELECT  -       fsgen
pl archive-01archive    ENABLED  ACTIVE   20480    CONCAT  -       RW
sd mydg03-03archive-01mydg01     0        409600   0       c1t2d0  ENA
```

Note Your system may use device names that differ from these examples. For more information on device names, see "Administering Disks" on page 49.

The subdisks line (beginning `sd`) indicates that the volume `archive` is on disk `mydg03`. To move the volume off `mydg03`, use the following command:

```
# vxassist -g mydg move archive !mydg03 dest_disk
```

Here *dest_disk* is the destination disk to which you want to move the volume. It is not necessary to specify a destination disk. If you do not specify a destination disk, the volume is moved to an available disk with enough space to contain the volume.

For example, to move a volume from disk `mydg03` to disk `mydg04`, in the disk group, `mydg`, use the following command:

```
# vxassist -g mydg move archive !mydg03 mydg04
```

This command indicates that the volume is to be reorganized so that no part of it remains on `mydg03`.

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 `mydg02`, `mydg03`, and `mydg04` in the disk group, `mydg`, and then remove the original plex `archive-01`:

```
# vxassist -g mydg mirror archive layout=stripe mydg02 mydg03 \
  mydg04
# vxplex -g mydg -o rm dis archive-01
```

After reorganizing any particularly busy volumes, check the disk statistics. If some volumes have been reorganized, clear statistics first and then accumulate statistics for a reasonable period of time.

If some disks appear to be excessively busy (or have particularly long read or write times), you may want to reconfigure some volumes. If there are two relatively busy volumes on a disk, move them closer together to reduce seek times on the disk. If there are too many relatively busy volumes on one disk, move them to a disk that is less busy.

Use I/O tracing (or subdisk statistics) to determine whether volumes have excessive activity in particular regions of the volume. If the active regions can be identified, split the subdisks in the volume and move those regions to a less busy disk.

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.

For information on using the `vxdmpadm` command to gather I/O statistics for a DMP node, path, or enclosure, see “Gathering and Displaying I/O Statistics” on page 108. You can also use the `vxdmpadm` command to change the I/O load-balancing policy for an enclosure as described in “Specifying the I/O Policy” on page 113.

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_enable_restore_daemon

Set to 1 to enable the DMP restore daemon; set to 0 to disable. The value of this tunable is changed by an entry in the `/kernel/drv/vxdmp.conf` file.

dmp_failed_io_threshold

The time limit for an I/O request in DMP. If the time exceeds this value, the usual result is to mark the disk as bad. The value of this tunable is changed by an entry in the `/kernel/drv/vxdmp.conf` file.

dmp_pathswitch_blks_shift

The default number of contiguous I/O blocks (expressed as the integer exponent of a power of 2; for example 11 represents 2048 blocks) 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 value of this tunable is changed by an entry in the `/kernel/drv/vxdmp.conf` file.

Note This parameter only affects the behavior of the `balanced` I/O policy. A value of 0 disables multipathing for the policy unless the `vxdmpadm` command is used to specify a different partition size as described in "Specifying the I/O Policy" on page 113.

dmp_restore_daemon_interval

The time in seconds between two invocations of the DMP Restore Daemon. The value of this tunable is changed by an entry in the `/kernel/drv/vxdmp.conf` file.

dmp_restore_daemon_policy

The DMP restore policy, which can be set to 0 (CHECK_ALL), 1 (CHECK_DISABLED), 2 (CHECK_PERIODIC), or 3 (CHECK_ALTERNATE). The value of this tunable is changed by an entry in the `/kernel/drv/vxdmp.conf` file.

dmp_restore_daemon_cycles

If the DMP restore policy is CHECK_PERIODIC, the number of cycles after which the CHECK_ALL policy is called. The value of this tunable is changed by an entry in the `/kernel/drv/vxdmp.conf` file.

dmp_retry_count

If an inquiry succeeds on a path, but there is an I/O error, the number of retries to attempt on the path. The 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 the bitmap size, the fewer the number of blocks that are mapped to each bit. This can reduce the amount of reading and writing required on resynchronization, at the expense of requiring more non-pageable kernel memory for the bitmap. Additionally, on clustered systems, a larger bitmap size increases the latency in I/O performance, and it also increases the load on the private network between the cluster members. This is because every other member of the cluster must be informed each time a bit in the map is marked.

Since the region size must be the same on all nodes in a cluster for a shared volume, the value of 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 4KB. The maximum and minimum permitted values are 1KB and 8KB.

Note The value of this tunable does not have any effect on Persistent FastResync.

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`.

If DRL sequential logging is configured, the value of `voldrl_min_regionsz` must be set to at least half the value of `vol_maxio`.

vol_maxioctl

The maximum size of data that can be passed into VxVM via an `ioctl` call. Increasing this limit allows larger operations to be performed. Decreasing the limit is not generally recommended, because some utilities depend upon performing operations of a certain size and can fail unexpectedly if they issue oversized `ioctl` requests.

The default value for this tunable is 32768 bytes (32KB).

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 45 for more information.

voldrl_max_drtregs

The maximum number of dirty regions that can exist on the system for non-sequential DRL on volumes. A larger value may result in improved system performance at the expense of recovery time. This tunable can be used to regulate the worse-case recovery time for the system following a failure.

The default value for this tunable is 2048.

voldrl_max_seq_dirty

The maximum number of dirty regions allowed for sequential DRL. This is useful for volumes that are usually written to sequentially, such as database logs. Limiting the number of dirty regions allows for faster recovery if a crash occurs.

The default value 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.

Note If DRL sequential logging is configured, the value of `voldrl_min_regionsz` must be set to at least half the value of `vol_maxio`.

voliomem_chunk_size

The granularity of memory chunks used by VxVM when allocating or releasing system memory. A larger granularity reduces CPU overhead due to memory allocation by allowing VxVM to retain hold of a larger amount of memory.

The default 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_dflt

The default size of the buffer maintained for error tracing events. This buffer is allocated at driver load time and is not adjustable for size while VxVM is running.

The default 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_default

The default size for the creation of a tracing buffer in the absence of any other specification of desired kernel buffer size as part of the trace `ioctl`.

The default 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.

volpagemod_max_memsz

The amount of memory, measured in kilobytes, that is allocated for caching FastResync and cache object metadata. This tunable has a default value of 6144KB (6MB) of physical memory. The valid range for this tunable is from 0 to 50% of physical memory.

Note The memory allocated for this cache is exclusively dedicated to it. It is not available for other processes or applications.

Setting the value of `volpagemod_max_memsz` below 512KB fails if cache objects or volumes that have been prepared for instant snapshot operations are present on the system.

If you do not use the FastResync or DRL features that are implemented using a version 20 DCO volume, the value of `volpagemod_max_memsz` can be set to 0. However, if you subsequently decide to enable these features, you can use the `vxtune` command to change the value to a more appropriate one:

```
# vxtune volpagemod_max_memsz value
```

where the new value is specified in kilobytes. Using the `vxtune` command to adjust the value of `volpagemod_max_memsz` does not persist across system reboots unless you also adjust the value that is configured in the `/kernel/drv/vxio.conf` file.

volraid_minpool_size

The initial amount of memory that is requested from the system by VxVM for RAID-5 operations. The maximum size of this memory pool is limited by the value of `voliomem_maxpool_sz`.

The default value for this tunable is 16384 sectors (8MB).

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 commonly used commands in VERITAS Volume Manager (VxVM). References are included to longer descriptions in the remainder of this book.

Most commands (excepting daemons, library commands and supporting scripts) are linked to the /usr/sbin directory from the /opt/VRTS/bin directory. It is recommended that you add the following directories to your PATH environment variable:

- If you are using the Bourne or Korn shell (sh or ksh), use the commands:

```
$ PATH=$PATH:/usr/sbin:/opt/VRTS/bin:/opt/VRTSvxfs/sbin:\
  /opt/VRTSob/bin
$ MANPATH=/usr/share/man:/opt/VRTS/man:$MANPATH
$ export PATH MANPATH
```

- If you are using a C shell (csh or tcsh), use the commands:

```
% set path = ( $path /usr/sbin /opt/VRTSvxfs/sbin \
  /opt/VRTSob/bin:/opt/VRTS/bin )
% setenv MANPATH /usr/share/man:/opt/VRTS/man:$MANPATH
```

Note If you have not installed VERITAS File System (VxFS), you can omit /opt/VRTSvxfs/bin, as this path is only required to access VxFS commands.

VxVM library commands and supporting scripts are located under the /usr/lib/vxvm directory hierarchy. You can include these directories in your path if you need to use them on a regular basis.

For detailed information about an individual command, refer to the appropriate manual page in the 1M section. A list of manual pages is provided in "Online Manual Pages" on page 316. Commands and scripts that are provided to support other commands and scripts, and which are not intended for general use, are not located in /opt/VRTS/bin and do not have manual pages.

The following tables summarize the commonly used commands:

- "Obtaining Information About Objects in VxVM" on page 303
- "Administering Disks" on page 304
- "Creating and Administering Disk Groups" on page 306
- "Creating and Administering Subdisks" on page 307
- "Creating and Administering Plexes" on page 309
- "Creating Volumes" on page 311
- "Administering Volumes" on page 313
- "Monitoring and Controlling Tasks" on page 315

Obtaining Information About Objects in VxVM

Command	Description
`vxdctl license`	List licensed features of VxVM.
`vxdisk [-g` *diskgroup*`] list [`*diskname*`]`	Lists disks under control of VxVM. See "Displaying Disk Information" on page 92. Example: `#` **`vxdisk -g mydg list`**
`vxdg list [`*diskgroup*`]`	Lists information about disk groups. See "Displaying Disk Group Information" on page 126. Example: `#` **`vxdg list mydg`**
`vxinfo [-g` *diskgroup*`] [`*volume* `...]`	Displays information about the accessibility and usability of volumes. See "Listing Unstartable Volumes" in the VERITAS Volume Manager Troubleshooting Guide. Example: `#` **`vxinfo -g mydg myvol1 myvol2`**
`vxprint -hrt [-g` *diskgroup*`] [`*object*`]`	Prints single-line information about objects in VxVM. See "Displaying Volume Information" on page 196. Example: `#` **`vxprint -g mydg myvol1 myvol2`**
`vxprint -st [-g` *diskgroup*`] [`*subdisk*`]`	Displays information about subdisks. See "Displaying Subdisk Information" on page 150. Example: `#` **`vxprint -st -g mydg`**
`vxprint -pt [-g` *diskgroup*`] [`*plex*`]`	Displays information about plexes. See "Displaying Plex Information" on page 158. Example: `#` **`vxprint -pt -g mydg`**

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 74. Example: `# `**`vxdiskadd c0t1d0`**

`vxedit [-g `*diskgroup*`] rename `*olddisk*` \`
newdisk
	Renames a disk under control of VxVM. See "Renaming a Disk" on page 91.
	Example:
	`# `**`vxedit -g mydg rename mydg03 \`**
	`mydg02`

`vxedit [-g `*diskgroup*`] set \`
`reserve=on|off `*diskname*
	Sets aside/does not set aside a disk from use in a disk group. See "Reserving Disks" on page 92.
	Examples:
	`# `**`vxedit -g mydg set reserve=on \`**
	`mydg02`
	`# `**`vxedit -g mydg set reserve=off \`**
	`mydg02`

`vxedit [-g `*diskgroup*`] set \`
`nohotuse=on|off `*diskname*
	Does not/does allow free space on a disk to be used for hot-relocation. See "Excluding a Disk from Hot-Relocation Use" on page 251 and "Making a Disk Available for Hot-Relocation Use" on page 252.
	Examples:
	`# `**`vxedit -g mydg set nohotuse=on \`**
	`mydg03`
	`# `**`vxedit -g mydg set nohotuse=off \`**
	`mydg03`

Administering Disks

Command	Description

`vxedit [-g` *diskgroup*`] set \`
 `spare=on|off` *diskname*
	Adds/removes a disk from the pool of hot-relocation spares. See "Marking a Disk as a Hot-Relocation Spare" on page 249 and "Removing a Disk from Use as a Hot-Relocation Spare" on page 250.
	Examples:
	`#` **`vxedit -g mydg set spare=on \`**
	`mydg04`
	`#` **`vxedit -g mydg set spare=off \`**
	`mydg04`

`vxdisk offline` *devicename*	Takes a disk offline. See "Taking a Disk Offline" on page 90. Example: `#` **`vxdisk offline c0t1d0`**
`vxdg -g` *diskgroup* `rmdisk` *diskname*	Removes a disk from its disk group. See "Removing a Disk from a Disk Group" on page 129. Example: `#` **`vxdg -g mydg rmdisk c0t2d0`**
`vxdiskunsetup` *devicename*	Removes a disk from control of VxVM. See "Removing a Disk from a Disk Group" on page 129. Example: `#` **`vxdiskunsetup c0t3d0`**

Command	**Description**
`vxdg init` *diskgroup* \ [*diskname*=] *devicename*	Creates a disk group using a pre-initialized disk. See "Creating a Disk Group" on page 128. Example: `# vxdg init mydg mydg01=c0t1d0`
`vxsplitlines -g` *diskgroup*	Reports conflicting configuration information. See "Handling Conflicting Configuration Copies in a Disk Group" on page 139. Example: `# vxsplitlines -g mydg`
`vxdg [-n` *newname*`] deport` *diskgroup*	Deports a disk group and optionally renames it. See "Deporting a Disk Group" on page 130. Example: `# vxdg -n newdg deport mydg`
`vxdg [-n` *newname*`] import` *diskgroup*	Imports a disk group and optionally renames it. See "Importing a Disk Group" on page 132. Example: `# vxdg -n newdg import mydg`
`vxrecover -g` *diskgroup* `-sb`	Starts all volumes in an imported disk group. See "Moving Disk Groups Between Systems" on page 134 for an example of its use. Example: `# vxrecover -g mydg -sb`
`vxdg destroy` *diskgroup*	Destroys a disk group and releases its disks. See "Destroying a Disk Group" on page 141. Example: `# vxdg destroy mydg`

Creating and Administering Subdisks

Command	Description
`vxmake [-g` *diskgroup*`] sd` *subdisk* \ *diskname,offset,length*	Creates a subdisk. See "Creating Subdisks" on page 149. Example: `#` **`vxmake -g mydg sd mydg02-01 \`** **`mydg02,0,8000`**
`vxsd [-g` *diskgroup*`] assoc` *plex subdisk*`. . .`	Associates subdisks with an existing plex. See "Associating Subdisks with Plexes" on page 152. Example: `#` **`vxsd -g mydg assoc home-1 \`** **`mydg02-01 mydg02-00 mydg02-01`**
`vxsd [-g` *diskgroup*`] assoc` *plex* \ *subdisk1:0 ... subdiskM:N-1*	Adds subdisks to the ends of the columns in a striped or RAID-5 volume. See "Associating Subdisks with Plexes" on page 152. Example: `#` **`vxsd -g mydg assoc vol01-01 \`** **`mydg10-01:0 mydg11-01:1 mydg12-01:2`**
`vxsd [-g` *diskgroup*`] mv` *oldsubdisk* \ *newsubdisk* `. . .`	Replaces a subdisk. See "Moving Subdisks" on page 150. Example: `#` **`vxsd -g mydg mv mydg01-01 \`** **`mydg02-01`**
`vxsd [-g` *diskgroup*`] -s` *size* `split \` *subdisk sd1 sd2*	Splits a subdisk in two. See "Splitting Subdisks" on page 151. Example: `#` **`vxsd -g mydg -s 1000m split \`** **`mydg03-02 mydg03-02 mydg03-03`**
`vxsd [-g` *diskgroup*`] join` *sd1 sd2* `. . . \` *subdisk*	Joins two or more subdisks. See "Joining Subdisks" on page 152. Example: `#` **`vxsd -g mydg join mydg03-02 \`** **`mydg03-03 mydg03-02`**

Command	Description
`vxassist [-g` *diskgroup*`] move \` *volume* `!`*olddisk* *newdisk*	Relocates subdisks in a volume between disks. See "Moving and Unrelocating Subdisks Using vxassist" on page 255. Example: `#` **`vxassist -g mydg move myvol !mydg02 mydg05`**
`vxunreloc [-g diskgroup]` *original_disk*	Relocates subdisks to their original disks. See "Moving and Unrelocating Subdisks Using vxunreloc" on page 255. Example: `#` **`vxunreloc -g mydg mydg01`**
`vxsd [-g` *diskgroup*`] dis` *subdisk*	Dissociates a subdisk from a plex. See "Dissociating Subdisks from Plexes" on page 154. Example: `#` **`vxsd -g mydg dis mydg02-01`**
`vxedit [-g` *diskgroup*`] rm` *subdisk*	Removes a subdisk. See "Removing Subdisks" on page 155. Example: `#` **`vxedit -g mydg rm mydg02-01`**
`vxsd [-g` *diskgroup*`] -o rm dis` *subdisk*	Dissociates and removes a subdisk from a plex. See "Dissociating Subdisks from Plexes" on page 154. Example: `#` **`vxsd -g mydg -o rm dis mydg02-01`**

Creating and Administering Plexes

Command	Description
`vxmake [-g` *diskgroup*`] plex` *plex* `\` `sd=`*subdisk1*`[,`*subdisk2*`,...]`	Creates a concatenated plex. See "Creating Plexes" on page 157. Example: `# vxmake -g mydg plex vol01-02 \ sd=mydg02-01,mydg02-02`
`vxmake [-g` *diskgroup*`] plex` *plex* `\` `layout=stripe|raid5 stwidth=`*W* `\` `ncolumn=`*N* `sd=`*subdisk1*`[,`*subdisk2*`,...]`	Creates a striped or RAID-5 plex. See "Creating a Striped Plex" on page 158. Example: `# vxmake -g mydg plex pl-01 \ layout=stripe stwidth=32 \ ncolumn=2 sd=mydg01-01,mydg02-01`
`vxplex [-g` *diskgroup*`] att` *volume plex*	Attaches a plex to an existing volume. See "Attaching and Associating Plexes" on page 163 and "Reattaching Plexes" on page 165. Example: `# vxplex -g mydg att vol01 vol01-02`
`vxplex [-g` *diskgroup*`] det` *plex*	Detaches a plex. See "Detaching Plexes" on page 165. Example: `# vxplex -g mydg det vol01-02`
`vxmend [-g` *diskgroup*`] off` *plex*	Takes a plex offline for maintenance. See "Taking Plexes Offline" on page 164. Example: `# vxmend -g mydg off vol02-02`
`vxmend [-g` *diskgroup*`] on` *plex*	Re-enables a plex for use. See "Reattaching Plexes" on page 165. Example: `# vxmend -g mydg on vol02-02`
`vxplex [-g` *diskgroup*`] mv` *oldplex newplex*	Replaces a plex. See "Moving Plexes" on page 166. Example: `# vxplex -g mydg mv vol02-02 \ vol02-03`

Command	Description
`vxplex [-g` *diskgroup*`] cp` *volume newplex*	Copies a volume onto a plex. See "Copying Plexes" on page 167. Example: `#` **`vxplex -g mydg cp vol02 vol03-01`**
`vxmend [-g` *diskgroup*`] fix clean` *plex*	Sets the state of a plex in an unstartable volume to CLEAN. See "Reattaching Plexes" on page 165. Example: `#` **`vxmend -g mydg fix clean vol02-02`**
`vxplex [-g` *diskgroup*`] -o rm dis` *plex*	Dissociates and removes a plex from a volume. See "Dissociating and Removing Plexes" on page 167. Example: `#` **`vxplex -g mydg -o rm dis vol03-01`**

Creating Volumes

Command	Description
`vxassist [-g` *diskgroup*`] maxsize \` `layout=`*layout* `[`*attributes*`]`	Displays the maximum size of volume that can be created. See "Discovering the Maximum Size of a Volume" on page 176. Example: `#` **`vxassist -g mydg maxsize \ layout=raid5 nlog=2`**
`vxassist [-g` *diskgroup*`] -b 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 177. Example: `#` **`vxassist -b -g mydg make myvol \ 20g layout=concat mydg01 mydg02`**
`vxassist [-g` *diskgroup*`] -b make \` *volume length* `layout=mirror \` `[nmirror=`*N*`] [`*attributes*`]`	Creates a mirrored volume. See "Creating a Mirrored Volume" on page 183. Example: `#` **`vxassist -b -g mydg make mymvol \ 20g layout=mirror nmirror=2`**
`vxassist [-g` *diskgroup*`] -b 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 185 and "Creating a RAID-5 Volume" on page 188. Example: `#` **`vxassist -b -g mydg make mysvol \ 20g layout=stripe stripeunit=32 ncol=4`**
`vxassist [-g` *diskgroup*`] -b make \` *volume* *`length`* `layout=mirror \` `mirror=ctlr [`*attributes*`]`	Creates a volume with mirrored data plexes on separate controllers. See "Mirroring across Targets, Controllers or Enclosures" on page 187. Example: `#` **`vxassist -b -g mydg make mymcvol \ 20g layout=mirror mirror=ctlr`**
`vxmake [-g` *diskgroup*`] -b -U`*usage_type* `\` `vol` *volume* `[len=`*length*`] plex=`*plex,...*	Creates a volume from existing plexes. See "Creating a Volume Using vxmake" on page 189. Example: `#` **`vxmake -g mydg -Uraid5 vol r5vol \ plex=raidplex,raidlog1,raidlog2`**

Command	Description
`vxvol [-g` *diskgroup*`] start` *volume*	Initializes and starts a volume for use. See "Initializing and Starting a Volume" on page 192 and "Starting a Volume" on page 203. Example: `# `**`vxvol -g mydg start r5vol`**
`vxvol [-g` *diskgroup*`] init zero` *volume*	Initializes and zeros out a volume for use. See "Initializing and Starting a Volume" on page 192. Example: `# `**`vxvol -g mydg init zero myvol`**

Administering Volumes

Command	Description
`vxassist [-g` *diskgroup*`] mirror` *volume* `\` [*attributes*]	Adds a mirror to a volume. See "Adding a Mirror to a Volume" on page 203. Example: `# vxassist -g mydg mirror myvol \ mydg10`
`vxassist [-g` *diskgroup*`] remove mirror \` *volume* [*attributes*]	Removes a mirror from a volume. See "Removing a Mirror" on page 205. Example: `# vxassist -g mydg remove \ mirror myvol !mydg11`
`vxassist [-g` *diskgroup*`] {growto|growby}\` *volume length*	Grows a volume to a specified size or by a specified amount. See "Resizing Volumes using vxassist" on page 213. Example: `# vxassist -g mydg growby myvol \ 10g`
`vxassist [-g` *diskgroup*`] \` `{shrinkto|shrinkby}` *volume length*	Shrinks a volume to a specified size or by a specified amount. See "Resizing Volumes using vxassist" on page 213. Example: `# vxassist -g mydg shrinkto myvol \ 20g`
`vxresize -b -F xvfs [-g` *diskgroup*`] \` *volume length diskname* . . .	Resizes a volume and the underlying VERITAS File System. See "Resizing Volumes using vxresize" on page 212. Example: `# vxassist -b -F vxfs -g mydg \ myvol 20g mydg10 mydg11`
`vxassist [-g` *diskgroup*`] relayout` *volume* `\` `[layout=`*layout*`] [`*relayout_options*`]`	Performs online relayout of a volume. See "Performing Online Relayout" on page 219. Example: `# vxassist -g mydg relayout \ vol2 layout=stripe`

Command	Description
`vxassist [-g` *diskgroup*`] relayout` *volume* `\` `layout=raid5 stripeunit=`*W* `ncol=`*N*	Relays out a volume as a RAID-5 volume with stripe width *W* and *N* columns. See “Performing Online Relayout” on page 219. Example: `#` **`vxassist -g mydg relayout \`** **`vol3 layout=raid5 \`** **`stripeunit=16 ncol=4`**
`vxrelayout [-g` *diskgroup*`] -o bg reverse \` *volume*	Reverses the direction of a paused volume relayout. See “Controlling the Progress of a Relayout” on page 224. Example: `#` **`vxrelayout -g mydg -o bg \`** **`reverse vol3`**
`vxassist [-g` *diskgroup*`] convert` *volume* `\` `[layout=`*layout*`] [`*convert_options*`]`	Converts between a layered volume and a non-layered volume layout. See “Converting Between Layered and Non-Layered Volumes” on page 225. Example: `#` **`vxassist -g mydg convert vol3 \`** **`layout=stripe-mirror`**
`vxassist [-g` *diskgroup*`] remove volume \` *volume*	Removes a volume. See “Removing a Volume” on page 216. Example: `#` **`vxassist -g mydg remove myvol`**

Monitoring and Controlling Tasks

Command	Description
`command [-g` *diskgroup*`] -t tasktag [options] [arguments]`	Specifies a task tag to a VxVM command. See "Specifying Task Tags" on page 199. Example: `#` **`vxrecover -g mydg -t mytask -b \ mydg05`**
`vxtask [-h] [-g` *diskgroup*`] list`	Lists tasks running on a system. See "`vxtask` Usage" on page 201. Example: `#` **`vxtask -h -g mydg list`**
`vxtask monitor` *task*	Monitors the progress of a task. See "`vxtask` Usage" on page 201. Example: `#` **`vxtask monitor mytask`**
`vxtask pause` *task*	Suspends operation of a task. See "`vxtask` Usage" on page 201. Example: `#` **`vxtask pause mytask`**
`vxtask -p [-g` *diskgroup*`] list`	Lists all paused tasks. See "`vxtask` Usage" on page 201. Example: `#` **`vxtask -p -g mydg list`**
`vxtask resume` *task*	Resumes a paused task. See "`vxtask` Usage" on page 201. Example: `#` **`vxtask resume mytask`**
`vxtask abort` *task*	Cancels a task and attempts to reverse its effects. See "`vxtask` Usage" on page 201. Example: `#` **`vxtask abort mytask`**

Online Manual Pages

Manual pages are organized into three sections:

- Section 1M — Administrative Commands
- Section 4 — File Formats
- Section 7 — Device Driver Interfaces

Section 1M — Administrative Commands

Manual pages in section 1M describe commands that are used to administer VERITAS Volume Manager.

Section 1M Manual Pages

Name	Description
`vxassist`	Create, relayout, convert, mirror, backup, grow, shrink, delete, and move volumes.
`vxcdsconvert`	Make disks and disk groups portable between systems.
`vxcmdlog`	Administer command logging.
`vxconfigbackup`	Back up disk group configuration.
`vxconfigbackupd`	Disk group configuration backup daemon.
`vxconfigd`	VERITAS Volume Manager configuration daemon
`vxconfigrestore`	Restore disk group configuration.
`vxdarestore`	Restore simple or nopriv disk access records.
`vxdctl`	Control the volume configuration daemon.
`vxddladm`	Device Discovery Layer subsystem administration.
`vxdg`	Manage VERITAS Volume Manager disk groups.
`vxdisk`	Define and manage VERITAS Volume Manager disks.
`vxdiskadd`	Add one or more disks for use with VERITAS Volume Manager.

Section 1M Manual Pages

Name	Description
`vxdiskadm`	Menu-driven VERITAS Volume Manager disk administration.
`vxdiskconfig`	Configure disk devices and bring them under VxVM control.
`vxdisksetup`	Configure a disk for use with VERITAS Volume Manager.
`vxdiskunsetup`	Deconfigure a disk from use with VERITAS Volume Manager.
`vxdmpadm`	DMP subsystem administration.
`vxdmpinq`	Display SCSI inquiry data.
`vxedit`	Create, remove, and modify VERITAS Volume Manager records.
`vxencap`	Encapsulate partitions on a new disk.
`vxevac`	Evacuate all volumes from a disk.
`vxinfo`	Print accessibility and usability of volumes.
`vxinstall`	Menu-driven VERITAS Volume Manager initial configuration.
`vxintro`	Introduction to the VERITAS Volume Manager utilities.
`vxiod`	Start, stop, and report on VERITAS Volume Manager kernel daemons.
`vxlufinish`	Finish a live upgrade of VERITAS Volume Manager.
`vxlustart`	Start a live upgrade of VERITAS Volume Manager.
`vxmake`	Create VERITAS Volume Manager configuration records.
`vxmemstat`	Display memory statistics for VERITAS Volume Manager.
`vxmend`	Mend simple problems in configuration records.
`vxmirror`	Mirror volumes on a disk or control default mirroring.
`vxnotify`	Display VERITAS Volume Manager configuration events.
`vxplex`	Perform VERITAS Volume Manager operations on plexes.

Section 1M Manual Pages

Name	Description
`vxpool`	Create and administer ISP storage pools.
`vxprint`	Display records from the VERITAS Volume Manager configuration.
`vxr5check`	Verify RAID-5 volume parity.
`vxreattach`	Reattach disk drives that have become accessible again.
`vxrecover`	Perform volume recovery operations.
`vxrelayout`	Convert online storage from one layout to another.
`vxrelocd`	Monitor VERITAS Volume Manager for failure events and relocate failed subdisks.
`vxresize`	Change the length of a volume containing a file system.
`vxsd`	Perform VERITAS Volume Manager operations on subdisks.
`vxse`	Storage Expert rules.
`vxsparecheck`	Monitor VERITAS Volume Manager for failure events and replace failed disks.
`vxstat`	VERITAS Volume Manager statistics management utility.
`vxtask`	List and administer VERITAS Volume Manager tasks.
`vxtemplate`	Install and administer ISP volume templates and template sets.
`vxtrace`	Trace operations on volumes.
`vxtranslog`	Administer transaction logging.
`vxtune`	Adjust VERITAS Volume Replicator and VERITAS Volume Manager tunables.
`vxunreloc`	Move a hot-relocated subdisk back to its original disk.
`vxusertemplate`	Create and administer ISP user templates.
`vxvol`	Perform VERITAS Volume Manager operations on volumes.

Section 1M Manual Pages

Name	Description
`vxvoladm`	Create and administer ISP application volumes on allocated storage.
`vxvoladmtask`	Administer ISP tasks.
`vxvset`	Create and administer volume sets.

Section 4 — File Formats

Manual pages in section 4 describe the format of files that are used by VERITAS Volume Manager.

Section 4 Manual Pages

Name	Description
`vol_pattern`	Disk group search specifications.
`vxmake`	`vxmake` description file.

Section 7 — Device Driver Interfaces

Manual pages in section 7 describe the interfaces to VERITAS Volume Manager devices.

Configuring VERITAS Volume Manager B

This appendix provides guidelines for setting up efficient storage management after installing the VERITAS Volume Manager software.

This chapter describes:

Setup Tasks After Installation

The setup sequence listed below is a typical example. Your system requirements may differ.

▼ Initial Setup Tasks

1. Create disk groups by placing disks under VERITAS Volume Manager control.
2. If you intend to use the Intelligent Storage Provisioning (ISP) feature, create storage pools within the disk groups.
3. Create volumes in the disk groups.
4. Configure file systems on the volumes.

▼ Optional Setup Tasks

- ◆ Designate hot-relocation spare disks in each disk group.
- ◆ Add mirrors to volumes.
- ◆ Configure DRL on volumes.

▼ Maintenance Tasks

- ◆ Resize volumes and file systems.
- ◆ Add more disks, create new disk groups, and create new volumes.
- ◆ Create and maintain snapshots.

Adding Unsupported Disk Arrays as JBODS

After installation, add any disk arrays that are unsupported by VERITAS to the DISKS (JBOD) category as described in "Administering the Device Discovery Layer" on page 57.

Adding Foreign Devices

The device discovery feature of VxVM can discover some devices that are controlled by third-party drivers, such as for EMC PowerPath. For these devices it may be preferable to use the multipathing capability that is provided by the third-party drivers rather than using the Dynamic Multipathing (DMP) feature. Provided that a suitable array support library is available, DMP can co-exist with such drivers. Other foreign devices, for which a compatible ASL does not exist, can be made available to VERITAS Volume Manager as simple disks by using the `vxddladm addforeign` command. This also has the effect of bypassing DMP. Refer to "Administering the Device Discovery Layer" on page 57 for more information.

Adding Disks to Disk Groups

To place disks in disk groups, use VEA or the `vxdiskadm` program after completing the installation. Refer to "Adding Disks to VxVM" on page 68 and the VEA online help for information on how to add your disks to new disk groups.

See the *VERITAS Storage Foundation Intelligent Storage Provisioning Administrator's Guide* for information about creating storage pools within disk groups. Storage pools are only required if you intend using the ISP feature of VxVM.

Guidelines for Configuring Storage

The following general guidelines help you understand and plan an efficient storage management system.

Guidelines for Protecting Your System and Data

A disk failure can cause loss of data on the failed disk and loss of access to your system. Loss of access is due to the failure of a key disk used for system operations. VERITAS Volume Manager can protect your system from these problems.

The following are suggestions for protecting your data:

- Perform regular backups to protect your data. Backups are necessary if all copies of a volume are lost or corrupted. Power surges can damage several (or all) disks on your system. Also, typing a command in error can remove critical files or damage a file system directly. Performing regular backups ensures that lost or corrupted data is available to be retrieved.
- Use mirroring to protect data against loss from a disk failure. See "Mirroring Guidelines" on page 324 for details.
- Use the DRL feature to speed up recovery of mirrored volumes after a system crash. See "Dirty Region Logging (DRL) Guidelines" on page 324 for details.
- Use striping to improve the I/O performance of volumes. See "Striping Guidelines" on page 325 for details.
- Make sure enough disks are available for a combined striped and mirrored configuration. At least two disks are required for the striped plex, and one or more additional disks are needed for the mirror.
- When combining striping and mirroring, never place subdisks from one plex on the same physical disk as subdisks from the other plex.
- Use logging to prevent corruption of recovery data in RAID-5 volumes. Make sure that each RAID-5 volume has at least one log plex. See "RAID-5 Guidelines" on page 326 for details.
- Leave the VERITAS Volume Manager hot-relocation feature enabled. See "Hot-Relocation Guidelines" on page 326 for details.

Mirroring Guidelines

Refer to the following guidelines when using mirroring.

- Do not place subdisks from different plexes of a mirrored volume on the same physical disk. This action compromises the availability benefits of mirroring and degrades performance. Using the `vxassist` or `vxdiskadm` commands precludes this from happening.
- To provide optimum performance improvements through the use of mirroring, at least 70 percent of physical I/O operations should be read operations. A higher percentage of read operations results in even better performance. Mirroring may not provide a performance increase or may even result in performance decrease in a write-intensive workload environment.

Note The operating system implements a file system cache. Read requests can frequently be satisfied from the cache. This can cause the read/write ratio for physical I/O operations through the file system to be biased toward writing (when compared to the read/write ratio at the application level).

- Where possible, use disks attached to different controllers when mirroring or striping. Most disk controllers support overlapped seeks. This allows seeks to begin on two disks at once. Do not configure two plexes of the same volume on disks that are attached to a controller that does not support overlapped seeks. This is important for older controllers or SCSI disks that do not cache on the drive. It is less important for modern SCSI disks and controllers. Mirroring across controllers allows the system to survive a failure of one of the controllers. Another controller can continue to provide data from a mirror.
- A plex exhibits superior performance when striped or concatenated across multiple disks, or when located on a much faster device. Set the read policy to prefer the faster plex. By default, a volume with one striped plex is configured to prefer reading from the striped plex.

For more information, see “Mirroring (RAID-1)” on page 26.

Dirty Region Logging (DRL) Guidelines

Dirty Region Logging (DRL) can speed up recovery of mirrored volumes following a system crash. When DRL is enabled, VERITAS Volume Manager keeps track of the regions within a volume that have changed as a result of writes to a plex.

Note Using Dirty Region Logging can impact system performance in a write-intensive environment.

For more information, see “Dirty Region Logging (DRL)” on page 43.

Striping Guidelines

Refer to the following guidelines when using striping.

- Do not place more than one column of a striped plex on the same physical disk.
- Calculate stripe-unit sizes carefully. In general, a moderate stripe-unit size (for example, 64 kilobytes, which is also the default used by `vxassist`) is recommended.
- If it is not feasible to set the stripe-unit size to the track size, and you do not know the application I/O pattern, use the default stripe-unit size.

Note Many modern disk drives have *variable geometry*. This means that the track size differs between cylinders, so that outer disk tracks have more sectors than inner tracks. It is therefore not always appropriate to use the track size as the stripe-unit size. For these drives, use a moderate stripe-unit size (such as 64 kilobytes), unless you know the I/O pattern of the application.

- Volumes with small stripe-unit sizes can exhibit poor sequential I/O latency if the disks do not have synchronized spindles. Generally, striping over disks without synchronized spindles yields better performance when used with larger stripe-unit sizes and multi-threaded, or largely asynchronous, random I/O streams.
- Typically, the greater the number of physical disks in the stripe, the greater the improvement in I/O performance; however, this reduces the effective mean time between failures of the volume. If this is an issue, combine striping with mirroring to combine high-performance with improved reliability.
- If only one plex of a mirrored volume is striped, set the policy of the volume to `prefer` for the striped plex. (The default read policy, `select`, does this automatically.)
- If more than one plex of a mirrored volume is striped, configure the same stripe-unit size for each striped plex.
- Where possible, distribute the subdisks of a striped volume across drives connected to different controllers and buses.
- Avoid the use of controllers that do not support overlapped seeks. (Such controllers are rare.)

The `vxassist` command automatically applies and enforces many of these rules when it allocates space for striped plexes in a volume.

For more information, see "Striping (RAID-0)" on page 22.

RAID-5 Guidelines

Refer to the following guidelines when using RAID-5.

In general, the guidelines for mirroring and striping together also apply to RAID-5. The following guidelines should also be observed with RAID-5:

- Only one RAID-5 plex can exist per RAID-5 volume (but there can be multiple log plexes).
- The RAID-5 plex must be derived from at least three subdisks on three or more physical disks. If any log plexes exist, they must belong to disks other than those used for the RAID-5 plex.
- RAID-5 logs can be mirrored and striped.
- If the volume length is not explicitly specified, it is set to the length of any RAID-5 plex associated with the volume; otherwise, it is set to zero. If you specify the volume length, it must be a multiple of the stripe-unit size of the associated RAID-5 plex, if any.
- If the log length is not explicitly specified, it is set to the length of the smallest RAID-5 log plex that is associated, if any. If no RAID-5 log plexes are associated, it is set to zero.
- Sparse RAID-5 log plexes are not valid.
- RAID-5 volumes are not supported for sharing in a cluster.

For more information, see "RAID-5 (Striping with Parity)" on page 30.

Hot-Relocation Guidelines

Hot-relocation automatically restores redundancy and access to mirrored and RAID-5 volumes when a disk fails. This is done by relocating the affected subdisks to disks designated as spares and/or free space in the same disk group.

The hot-relocation feature is enabled by default. The associated daemon, `vxrelocd`, is automatically started during system startup.

Refer to the following guidelines when using hot-relocation.

- The hot-relocation feature is enabled by default. Although it is possible to disable hot-relocation, it is advisable to leave it enabled. It will notify you of the nature of the failure, attempt to relocate any affected subdisks that are redundant, and initiate recovery procedures.

- Although hot-relocation does not require you to designate disks as spares, designate at least one disk as a spare within each disk group. This gives you some control over which disks are used for relocation. If no spares exist, VERITAS Volume Manager uses any available free space within the disk group. When free space is used for relocation purposes, it is possible to have performance degradation after the relocation.
- After hot-relocation occurs, designate one or more additional disks as spares to augment the spare space. Some of the original spare space may be occupied by relocated subdisks.
- If a given disk group spans multiple controllers and has more than one spare disk, set up the spare disks on different controllers (in case one of the controllers fails).
- For a mirrored volume, configure the disk group so that there is at least one disk that does not already contain a mirror of the volume. This disk should either be a spare disk with some available space or a regular disk with some free space and the disk is not excluded from hot-relocation use.
- For a mirrored and striped volume, configure the disk group so that at least one disk does not already contain one of the mirrors of the volume or another subdisk in the striped plex. This disk should either be a spare disk with some available space or a regular disk with some free space and the disk is not excluded from hot-relocation use.
- For a RAID-5 volume, configure the disk group so that at least one disk does not already contain the RAID-5 plex (or one of its log plexes) of the volume. This disk should either be a spare disk with some available space or a regular disk with some free space and the disk is not excluded from hot-relocation use.
- If a mirrored volume has a DRL log subdisk as part of its data plex, you cannot relocate the data plex. Instead, place log subdisks in log plexes that contain no data.
- Hot-relocation does not guarantee to preserve the original performance characteristics or data layout. Examine the locations of newly-relocated subdisks to determine whether they should be relocated to more suitable disks to regain the original performance benefits.
- Although it is possible to build VERITAS Volume Manager objects on spare disks (using `vxmake` or the VEA interface), it is recommended that you use spare disks for hot-relocation only.

See “Administering Hot-Relocation” on page 241 for more information.

Accessing Volume Devices

As soon as a volume has been created and initialized, it is available for use as a virtual disk partition by the operating system for the creation of a file system, or by application programs such as relational databases and other data management software.

Creating a volume in a disk group sets up block and character (raw) device files that can be used to access the volume:

`/dev/vx/dsk/`*diskgroup*`/`*volume*	block device file for *volume*
`/dev/vx/rdsk/`*diskgroup*`/`*volume*	character device file for *volume*

The pathnames include a directory named for the disk group. Use the appropriate device node to create, mount and repair file systems, and to lay out databases that require raw partitions.

Controlling VxVM's View of Multipathed Devices

To control how a device is treated by the Dynamic Multipathing (DMP) feature of VxVM, use the `vxdiskadm` command as described in "Disabling and Enabling Multipathing for Specific Devices" on page 99.

Reconfiguration Tasks

The following sections describe tasks that allow you to make changes to the configuration that you specified during installation.

Changing the Name of the Default Disk Group

If you use the VERITAS installer to install the VERITAS Volume Manager software, you can enter the name of the default disk group. This disk group will be used by commands if you do not specify the -g option, or the `VXVM_DEFAULTDG` environment variable is not set. If required, you can use the `vxdctl defaultdg` command to change the default disk group. See "Displaying and Specifying the System-Wide Default Disk Group" on page 125 for details.

Enabling or Disabling Enclosure-Based Naming

If you use the VERITAS installer to install the VERITAS Volume Manager software, you can choose whether the displayed disk access names are based on device names or on names that you assign to disk enclosures. If required, you can use the `vxdiskadm` or `vxddladm` commands to change the naming convention that is used. See "Changing the Disk-Naming Scheme" on page 63 for more information.

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.

boot disk group

A private disk group that contains the disks from which the system may be booted.

bootdg

A reserved disk group name that is an alias for the name of the boot disk group.

clean node shutdown

The ability of a node to leave a cluster gracefully when all access to shared volumes has ceased.

cluster

A set of hosts (each termed a node) that share a set of disks.

cluster manager

An externally-provided daemon that runs on each node in a cluster. The cluster managers on each node communicate with each other and inform VxVM of changes in cluster membership.

cluster-shareable disk group

A disk group in which access to the disks is shared by multiple hosts (also referred to as a shared disk group). 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, and dirty region logs (see dirty region logging).

detached

A state in which a VxVM object is associated with another object, but not enabled for use.

device name

The device name or address used to access a physical disk, such as `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 method by which the VxVM monitors and logs modifications to a plex as a bitmap of changed regions. For a volumes with a new-style DCO volume, the dirty region log (DRL) is maintained in the DCO volume. Otherwise, the DRL is allocated to an associated subdisk called a *log subdisk*.

disabled path

A path to a disk that is not available for I/O. A path can be *disabled* due to real hardware failures or if the user has used the `vxdmpadm disable` command on that controller.

disk

A collection of read/write data blocks that are indexed and can be accessed fairly quickly. Each disk has a universally unique identifier.

disk access name

An alternative term for a device name.

disk access records

Configuration records used to specify the access path to particular disks. Each disk access record contains a name, a type, and possibly some type-specific information, which is used by VxVM in deciding how to access and manipulate the disk that is defined by the disk access record.

disk array

A collection of disks logically arranged into an object. Arrays tend to provide benefits such as redundancy or improved performance. 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 disk group names `bootdg` (an alias for the boot disk group), `defaultdg` (an alias for the default disk group) and `nodg` (represents no disk group) are reserved.

disk group ID

A unique identifier used to identify a disk group.

disk ID

A universally unique identifier that is given to each disk and can be used to identify the disk, even if it is moved.

disk media name

An alternative term for a disk name.

disk media record

A configuration record that identifies a particular disk, by disk ID, and gives that disk a logical (or administrative) name.

disk name

A logical or administrative name chosen for a disk that is under the control of VxVM, such as `disk03`. The term disk media name is also used to refer to a disk name.

dissociate

The process by which any link that exists between two VxVM objects is removed. For example, dissociating a subdisk from a plex removes the subdisk from the plex and adds the subdisk to the free space pool.

dissociated plex

A plex dissociated from a volume.

dissociated subdisk

A subdisk dissociated from a plex.

distributed lock manager

A lock manager that runs on different systems in a cluster, and ensures consistent access to distributed resources.

enabled path

A path to a disk that is available for I/O.

encapsulation

A process that converts existing partitions on a specified disk to volumes. If any partitions contain file systems, `/etc/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 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.

snapshot

A point-in-time copy of a volume (volume snapshot) or a file system (file system snapshot).

spanning

A layout technique that permits a volume (and its file system or database) that is too large to fit on a single disk to be configured across multiple physical disks.

sparse plex

A plex that is not as long as the volume or that has holes (regions of the plex that do not have a backing subdisk).

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 64KB. The stripe unit size is sometimes also referred to as the *stripe width*.

striping

A layout technique that spreads data across several physical disks using stripes. The data is allocated alternately to the stripes within the subdisks of each plex.

subdisk

A consecutive set of contiguous disk blocks that form a logical disk segment. Subdisks can be associated with plexes to form volumes.

swap area

A disk region used to hold copies of memory pages swapped out by the system pager process.

swap volume

A VxVM volume that is configured for use as a swap area.

transaction

A set of configuration changes that succeed or fail as a group, rather than individually. Transactions are used internally to maintain consistent configurations.

volboot file

A small file that is used to locate copies of the boot disk group configuration. The file may list disks that contain configuration copies in standard locations, and can also contain direct pointers to configuration copy locations. The `volboot` file is stored in a system-dependent location.

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