Veritas™ Volume Manager Administrator's Guide

Solaris

5.0 Maintenance Pack 3

symantec™

Veritas™ Volume Manager Administrator's Guide

The software described in this book is furnished under a license agreement and may be used only in accordance with the terms of the agreement.

Product Version:5.0 MP3

Document version: 5.0MP3.0

Legal Notice

Copyright © 2008 Symantec Corporation. All rights reserved.

Symantec, the Symantec Logo, Veritas, and Veritas Storage Foundation are trademarks or registered trademarks of Symantec Corporation or its affiliates in the U.S. and other countries. Other names may be trademarks of their respective owners.

This Symantec product may contain third party software for which Symantec is required to provide attribution to the third party (“Third Party Programs”). Some of the Third Party Programs are available under open source or free software licenses. The License Agreement accompanying the Software does not alter any rights or obligations you may have under those open source or free software licenses. Please see the Third Party Legal Notice Appendix to this Documentation or TPIP ReadMe File accompanying this Symantec product for more information on the Third Party Programs.

The product described in this document is distributed under licenses restricting its use, copying, distribution, and decompilation/reverse engineering. No part of this document may be reproduced in any form by any means without prior written authorization of Symantec Corporation and its licensors, if any.

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

The Licensed Software and Documentation are deemed to be commercial computer software as defined in FAR 12.212 and subject to restricted rights as defined in FAR Section 52.227-19 "Commercial Computer Software - Restricted Rights" and DFARS 227.7202, "Rights in Commercial Computer Software or Commercial Computer Software Documentation", as applicable, and any successor regulations. Any use, modification, reproduction release, performance, display or disclosure of the Licensed Software and Documentation by the U.S. Government shall be solely in accordance with the terms of this Agreement.

Symantec Corporation
20330 Stevens Creek Blvd.
Cupertino, CA 95014

http://www.symantec.com

Technical Support

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

Symantec's maintenance offerings include the following:

- A range of support options that give you the flexibility to select the right amount of service for any size organization
- Telephone and Web-based support that provides rapid response and up-to-the-minute information
- Upgrade assurance that delivers automatic software upgrade protection
- Global support that is available 24 hours a day, 7 days a week
- Advanced features, including Account Management Services

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

www.symantec.com/techsupp/

Contacting Technical Support

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

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

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

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

- Product release level
- Hardware information
- Available memory, disk space, and NIC information
- Operating system

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

Licensing and registration

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

www.symantec.com/techsupp/

Customer service

Customer service information is available at the following URL:

www.symantec.com/techsupp/

Customer Service is available to assist with the following types of issues:

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

Maintenance agreement resources

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

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

Additional enterprise services

Symantec offers a comprehensive set of services that allow you to maximize your investment in Symantec products and to develop your knowledge, expertise, and global insight, which enable you to manage your business risks proactively.

Enterprise services that are available include the following:

Symantec Early Warning Solutions	These solutions provide early warning of cyber attacks, comprehensive threat analysis, and countermeasures to prevent attacks before they occur.
Managed Security Services	These services remove the burden of managing and monitoring security devices and events, ensuring rapid response to real threats.
Consulting Services	Symantec Consulting Services provide on-site technical expertise from Symantec and its trusted partners. Symantec Consulting Services offer a variety of prepackaged and customizable options that include assessment, design, implementation, monitoring, and management capabilities. Each is focused on establishing and maintaining the integrity and availability of your IT resources.
Educational Services	Educational Services provide a full array of technical training, security education, security certification, and awareness communication programs.

To access more information about Enterprise services, please visit our Web site at the following URL:

www.symantec.com

Select your country or language from the site index.

Contents

Technical Support 4

Chapter 1 Understanding Veritas Volume Manager 23

About Veritas Volume Manager 23
VxVM and the operating system 25
How data is stored 25
How VxVM handles storage management 26
Physical objects 26
Virtual objects 31
Volume layouts in VxVM 39
Implementation of non-layered volumes 39
Implementation of layered volumes 39
Layout methods 40
Concatenation and spanning 40
Striping (RAID-0) 42
Mirroring (RAID-1) 46
Striping plus mirroring (mirrored-stripe or RAID-0+1) 47
Mirroring plus striping (striped-mirror, RAID-1+0 or RAID-10) 48
RAID-5 (striping with parity) 49
Online relayout 56
How online relayout works 56
Limitations of online relayout 59
Transformation characteristics 60
Transformations and volume length 60
Volume resynchronization 61
Dirty flags 61
Resynchronization process 61
Dirty region logging 62
Dirty region logs 62
Log subdisks and plexes 63
Sequential DRL 63
SmartSync recovery accelerator 63
Volume snapshots 65
Comparison of snapshot features 67
FastResync 68

FastResync enhancements 68
Non-persistent FastResync 69
Persistent FastResync 70
DCO volume versioning 70
FastResync limitations 75
Hot-relocation 76
Volume sets 77
Configuration of volumes on SAN storage 77

Chapter 2 Administering disks 81

About disk management 82
Disk devices 82
Disk device naming in VxVM 83
Private and public disk regions 85
Discovering and configuring newly added disk devices 87
Partial device discovery 88
Discovering disks and dynamically adding disk arrays 89
Third-party driver coexistence 91
How to administer the Device Discovery Layer 93
Disks under VxVM control 105
Changing the disk-naming scheme 106
Displaying the disk-naming scheme 107
Regenerating persistent device names 108
Changing device naming for TPD-controlled enclosures 108
Simple or nopriv disks with enclosure-based naming 110
Discovering the association between enclosure-based disk names and OS-based disk names 112
Disk installation and formatting 112
Displaying or changing default disk layout attributes 113
Adding a disk to VxVM 113
Disk reinitialization 121
Using vxdiskadd to put a disk under VxVM control 122
RAM disk support in VxVM 122
Encapsulating a disk 123
Failure of disk encapsulation 126
Using nopriv disks for encapsulation 127
Rootability 128
Booting root volumes 129
Boot-time volume restrictions 130
Mirroring an encapsulated root disk 131
Booting from alternate boot disks 132
Mirroring other file systems on the root disk 136

Encapsulating SAN disks 136
Unencapsulating the root disk 137
Dynamic LUN expansion 138
Extended Copy Service 139
Enabling a disk for Extended Copy Service operation 140
Removing disks 141
Removing a disk with subdisks 143
Removing a disk with no subdisks 144
Removing a disk from VxVM control 144
Removing and replacing disks 144
Replacing a failed or removed disk 149
Removing and replacing a disk in a Sun StorEdge A5x00 disk array 151
Enabling a disk 152
Taking a disk offline 153
Renaming a disk 153
Reserving disks 154
Displaying disk information 155
Displaying disk information with vxdiskadm 156
Changing host LUN configurations online 157
Dynamic LUN removal from an existing target ID 157
Dynamic new LUN addition to a new target ID 159
Cleaning up the Operating System device tree after removing LUNs - Solaris 10 example 160

Chapter 3 Administering Dynamic Multipathing 163

How DMP works 163
How DMP monitors I/O on paths 166
Load balancing 168
Dynamic Reconfiguration 168
Using MPxIO with DMP 168
Booting from DMP devices 169
DMP in a clustered environment 169
Disabling multipathing and making devices invisible to VxVM 170
Enabling multipathing and making devices visible to VxVM 171
Enabling and disabling I/O for controllers and storage processors 172
Displaying DMP database information 173
Displaying the paths to a disk 173
Setting customized names for DMP nodes 175
Administering DMP using vxdmpadm 176
Retrieving information about a DMP node 177
Displaying consolidated information about the DMP nodes 178

Displaying the members of a LUN group 179
Displaying paths controlled by a DMP node, controller, enclosure, or array port 179
Displaying information about controllers 182
Displaying information about enclosures 183
Displaying information about array ports 183
Displaying information about TPD-controlled devices 184
Displaying HBA details 185
Displaying extended device attributes 185
Suppressing or including devices for VxVM or DMP control 187
Gathering and displaying I/O statistics 188
Setting the attributes of the paths to an enclosure 193
Displaying the redundancy level of a device or enclosure 195
Specifying the minimum number of active paths 196
Displaying the I/O policy 196
Specifying the I/O policy 197
Disabling I/O for paths, controllers or array ports 202
Enabling I/O for paths, controllers or array ports 203
Upgrading disk controller firmware 204
Renaming an enclosure 205
Configuring the response to I/O failures 206
Configuring the I/O throttling mechanism 207
Displaying recovery option values 209
Configuring DMP path restoration policies 210
Stopping the DMP path restoration thread 212
Displaying the status of the DMP path restoration thread 212
Displaying information about the DMP error-handling thread 212
Configuring array policy modules 212

Chapter 4 Creating and administering disk groups 215

About disk groups 216
Specification of disk groups to commands 217
System-wide reserved disk groups 217
Rules for determining the default disk group 218
Displaying disk group information 220
Displaying free space in a disk group 221
Creating a disk group 221
Adding a disk to a disk group 223
Removing a disk from a disk group 223
Deporting a disk group 224
Importing a disk group 226

Handling cloned disks with duplicated identifiers 226
Writing a new UDID to a disk 227
Importing a disk group containing cloned disks 228
Sample cases of operations on cloned disks 230
Renaming a disk group 235
Moving disks between disk groups 237
Moving disk groups between systems 238
Handling errors when importing disks 239
Reserving minor numbers for disk groups 241
Compatibility of disk groups between platforms 243
Handling conflicting configuration copies 244
Example of a serial split brain condition in a cluster 244
Correcting conflicting configuration information 249
Reorganizing the contents of disk groups 250
Limitations of disk group split and join 254
Listing objects potentially affected by a move 255
Moving objects between disk groups 257
Splitting disk groups 260
Joining disk groups 261
Disabling a disk group 263
Destroying a disk group 263
Recovering a destroyed disk group 263
Upgrading a disk group 264
Managing the configuration daemon in VxVM 268
Backing up and restoring disk group configuration data 269
Using vxnotify to monitor configuration changes 270

Chapter 5 Creating and administering subdisks 271

About subdisks 271
Creating subdisks 272
Displaying subdisk information 272
Moving subdisks 273
Splitting subdisks 274
Joining subdisks 274
Associating subdisks with plexes 275
Associating log subdisks 276
Dissociating subdisks from plexes 277
Removing subdisks 278
Changing subdisk attributes 278

Chapter 6 Creating and administering plexes 281

About plexes 281
Creating plexes 282
Creating a striped plex 282
Displaying plex information 283
Plex states 283
Plex condition flags 286
Plex kernel states 287
Attaching and associating plexes 287
Taking plexes offline 288
Detaching plexes 289
Reattaching plexes 289
Moving plexes 290
Copying plexes 291
Dissociating and removing plexes 291
Changing plex attributes 292

Chapter 7 Creating volumes 295

About volume creation 296
Types of volume layouts 296
Supported volume logs and maps 298
Creating a volume 299
Advanced approach 299
Assisted approach 300
Using vxassist 300
Setting default values for vxassist 302
Using the SmartMove™ feature while attaching a plex 303
Discovering the maximum size of a volume 304
Disk group alignment constraints on volumes 304
Creating a volume on any disk 305
Creating a volume on specific disks 305
Specifying ordered allocation of storage to volumes 307
Creating a mirrored volume 310
Creating a mirrored-concatenated volume 311
Creating a concatenated-mirror volume 311
Creating a volume with a version 0 DCO volume 311
Creating a volume with a version 20 DCO volume 315
Creating a volume with dirty region logging enabled 315
Creating a striped volume 316
Creating a mirrored-stripe volume 317
Creating a striped-mirror volume 318
Mirroring across targets, controllers or enclosures 318

Creating a RAID-5 volume 319
Creating tagged volumes 321
Creating a volume using vxmake 322
Creating a volume using a vxmake description file 323
Initializing and starting a volume 324
Initializing and starting a volume created using vxmake 325
Accessing a volume 326

Chapter 8 Administering volumes 327

About volume administration 328
Displaying volume information 328
Volume states 330
Volume kernel states 331
Monitoring and controlling tasks 332
Specifying task tags 332
Managing tasks with vxtask 332
Stopping a volume 334
Putting a volume in maintenance mode 335
Starting a volume 335
Adding a mirror to a volume 336
Mirroring all volumes 336
Mirroring volumes on a VM disk 337
Additional mirroring considerations 338
Removing a mirror 339
Adding logs and maps to volumes 340
Preparing a volume for DRL and instant snapshots 341
Specifying storage for version 20 DCO plexes 342
Using a DCO and DCO volume with a RAID-5 volume 343
Determining the DCO version number 344
Determining if DRL is enabled on a volume 345
Determining if DRL logging is active on a volume 346
Disabling and re-enabling DRL 346
Removing support for DRL and instant snapshots from a volume 346
Upgrading existing volumes to use version 20 DCOs 347
Adding traditional DRL logging to a mirrored volume 349
Removing a traditional DRL log 350
Adding a RAID-5 log 351
Adding a RAID-5 log using vxplex 351
Removing a RAID-5 log 352
Resizing a volume 353
Resizing volumes using vxresize 353

Resizing volumes using vxassist 355
Resizing volumes using vxvol 356
Setting tags on volumes 357
Changing the read policy for mirrored volumes 358
Removing a volume 359
Moving volumes from a VM disk 360
Enabling FastResync on a volume 361
Checking whether FastResync is enabled on a volume 362
Disabling FastResync 363
Performing online relayout 363
Permitted relayout transformations 364
Specifying a non-default layout 367
Specifying a plex for relayout 368
Tagging a relayout operation 368
Viewing the status of a relayout 368
Controlling the progress of a relayout 369
Converting between layered and non-layered volumes 370
Using Thin Provisioning 371
About Thin Provisioning 371
About Thin Reclamation 371
Thin Reclamation of a disk, a disk group, or an enclosure 372

Chapter 9 Administering volume snapshots 373

About volume snapshots 373
Traditional third-mirror break-off snapshots 375
Full-sized instant snapshots 376
Space-optimized instant snapshots 378
Emulation of third-mirror break-off snapshots 379
Linked break-off snapshot volumes 380
Cascaded snapshots 381
Creating a snapshot of a snapshot 383
Creating multiple snapshots 385
Restoring the original volume from a snapshot 386
Creating instant snapshots 387
Preparing to create instant and break-off snapshots 390
Creating and managing space-optimized instant snapshots 394
Creating and managing full-sized instant snapshots 397
Creating and managing third-mirror break-off snapshots 399
Creating and managing linked break-off snapshot volumes 402
Creating multiple instant snapshots 404
Creating instant snapshots of volume sets 405
Adding snapshot mirrors to a volume 407

Removing a snapshot mirror 408
Removing a linked break-off snapshot volume 408
Adding a snapshot to a cascaded snapshot hierarchy 408
Refreshing an instant snapshot 409
Reattaching an instant snapshot 409
Reattaching a linked break-off snapshot volume 410
Restoring a volume from an instant snapshot 411
Dissociating an instant snapshot 412
Removing an instant snapshot 413
Splitting an instant snapshot hierarchy 413
Displaying instant snapshot information 414
Controlling instant snapshot synchronization 416
Listing the snapshots created on a cache 418
Tuning the autogrow attributes of a cache 418
Growing and shrinking a cache 419
Removing a cache 420
Creating traditional third-mirror break-off snapshots 420
Converting a plex into a snapshot plex 424
Creating multiple snapshots 425
Reattaching a snapshot volume 425
Adding plexes to a snapshot volume 427
Dissociating a snapshot volume 427
Displaying snapshot information 428
Adding a version 0 DCO and DCO volume 429
Specifying storage for version 0 DCO plexes 431
Removing a version 0 DCO and DCO volume 432
Reattaching a version 0 DCO and DCO volume 432

Chapter 10 Creating and administering volume sets 435

About volume sets 435
Creating a volume set 436
Adding a volume to a volume set 437
Listing details of volume sets 437
Stopping and starting volume sets 438
Removing a volume from a volume set 439
Raw device node access to component volumes 439
Enabling raw device access when creating a volume set 440
Displaying the raw device access settings for a volume set 441
Controlling raw device access for an existing volume set 441

Chapter 11 Configuring off-host processing 443

About off-host processing solutions 443
Implemention of off-host processing solutions 444
Implementing off-host online backup 445
Implementing decision support 449

Chapter 12 Administering hot-relocation 455

About hot-relocation 455
How hot-relocation works 456
Partial disk failure mail messages 459
Complete disk failure mail messages 460
How space is chosen for relocation 461
Configuring a system for hot-relocation 462
Displaying spare disk information 463
Marking a disk as a hot-relocation spare 463
Removing a disk from use as a hot-relocation spare 464
Excluding a disk from hot-relocation use 465
Making a disk available for hot-relocation use 466
Configuring hot-relocation to use only spare disks 467
Moving relocated subdisks 467
Moving relocated subdisks using vxdiskadm 468
Moving relocated subdisks using vxassist 469
Moving relocated subdisks using vxunreloc 470
Restarting vxunreloc after errors 472
Modifying the behavior of hot-relocation 473

Chapter 13 Administering cluster functionality 475

About the cluster functionality of VxVM 475
Overview of cluster volume management 477
Private and shared disk groups 479
Activation modes of shared disk groups 480
Connectivity policy of shared disk groups 482
Effect of disk connectivity on cluster reconfiguration 487
Limitations of shared disk groups 487
Cluster initialization and configuration 488
Cluster reconfiguration 489
Volume reconfiguration 492
Node shutdown 495
Node abort 496
Cluster shutdown 496
Dirty region logging in cluster environments 496

How DRL works in a cluster environment 497
Multiple host failover configurations 498
Import lock 498
Failover 499
Corruption of disk group configuration 499
Administering VxVM in cluster environments 500
Requesting node status and discovering the master node 500
Determining if a disk is shareable 501
Listing shared disk groups 502
Creating a shared disk group 503
Forcibly adding a disk to a disk group 504
Importing disk groups as shared 504
Converting a disk group from shared to private 505
Moving objects between shared disk groups 505
Splitting shared disk groups 506
Joining shared disk groups 506
Changing the activation mode on a shared disk group 506
Setting the disk detach policy on a shared disk group 507
Setting the disk group failure policy on a shared disk group 507
Creating volumes with exclusive open access by a node 507
Setting exclusive open access to a volume by a node 508
Displaying the cluster protocol version 508
Displaying the supported cluster protocol version range 509
Upgrading the cluster protocol version 509
Recovering volumes in shared disk groups 509
Obtaining cluster performance statistics 510

Chapter 14 Administering sites and remote mirrors 511

About sites and remote mirrors 511
Configuring Remote Mirror sites 514
Configuring sites for hosts 515
Configuring sites for storage 515
Configuring automatic site tagging for a disk group 516
Configuring site tagging for disks or enclosures 516
Changing the site name 517
Configuring site-based allocation on a disk group 517
Configuring site consistency on a disk group 518
Configuring site consistency on a volume 520
Setting the siteread policy on a volume 520
Site-based allocation of storage to volumes 521
Examples of storage allocation by specifying sites 522
Making an existing disk group site consistent 523

Fire drill — testing the configuration 524
Simulating site failure 525
Verifying the secondary site 525
Recovery from simulated site failure 525
Failure scenarios and recovery procedures 525
Recovering from a loss of site connectivity 526
Recovering from host failure 526
Recovering from storage failure 526
Recovering from site failure 527
Automatic site reattachment 527

Chapter 15 Using Storage Expert 529

About Storage Expert 529
How Storage Expert works 530
Before using Storage Expert 530
Running Storage Expert 530
Discovering what a rule does 531
Displaying rule attributes and their default values 531
Running a rule 532
Identifying configuration problems using Storage Expert 533
Recovery time 534
Disk groups 535
Disk striping 537
Disk sparing and relocation management 538
Hardware failures 539
Rootability 539
System name 539
Rule definitions and attributes 539

Chapter 16 Performance monitoring and tuning 545

Performance guidelines 545
Data assignment 545
Striping 546
Mirroring 546
Combining mirroring and striping 547
RAID-5 547
Volume read policies 547
Performance monitoring 548
Setting performance priorities 548
Obtaining performance data 549
Using performance data 550
Tuning VxVM 554

General tuning guidelines 554
Tuning guidelines for large systems 554
Changing the values of tunables 555
Tunable parameters for VxVM 556
DMP tunable parameters 565

Appendix A Using Veritas Volume Manager commands 571

About Veritas Volume Manager commands 571
Online manual pages 593
Section 1M – administrative commands 594
Section 4 – file formats 597
Section 7 – device driver interfaces 597

Appendix B Veritas Storage Foundation support for Solaris Zones 599

About Solaris Zones 599
Adding VxFS file systems to a non-global zone 600
Delegating VxFS file systems to a non-global zone 600
Veritas Storage Foundation Cluster File System mounts 601
Concurrent I/O access in non-global zones 602
Veritas extension for Oracle Disk Manager 602
Exporting VxVM volumes to a non-global zone 603
Volume physical nodes in Solaris 10 604
Removing a VxVM volume from a non-global zone 605
Software limitations of Storage Foundation support of non-global zones 605
Solaris patches required for Veritas Volume Manager support in non-global zone 605
Cannot remove a volume configured in a non-global zone 605
Administration commands are not supported in non-global zone 606
A cluster-mounted VxFS file system is not supported as the root of a non-global zone 606
QIO and CQIO are not supported 606
Package and patch installation in non-global zones 606
Root volume cannot be added to non-global zones 606
Some Veritas Volume Manager operations can cause volume device names to go out of sync 606

Appendix C Migrating arrays 609

Migrating to thin provisioning 609

Appendix D Configuring Veritas Volume Manager 613

Setup tasks after installation 613
Unsupported disk arrays 614
Foreign devices 614
Initialization of disks and creation of disk groups 614
Guidelines for configuring storage 615
Mirroring guidelines 616
Dirty region logging guidelines 616
Striping guidelines 617
RAID-5 guidelines 618
Hot-relocation guidelines 618
Accessing volume devices 620
VxVM's view of multipathed devices 620
Cluster support 620
Configuring shared disk groups 620
Converting existing VxVM disk groups to shared disk groups 622

Appendix E Migrating from Solaris Volume Manager to VxVM 623

About migration from Solaris Volume Manager 623
How Solaris Volume Manager objects are mapped to VxVM objects 625
Conversion of soft partitions 627
Overview of the conversion process 628
Plan and prepare for the conversion 629
Set up the conversion 629
Perform the conversion 630
Perform post-conversion tasks 630
Planning the conversion 630
Scheduling considerations 630
Schedule downtime 631
Check metadevices 632
Identify references by applications 633
Preparing a Solaris Volume Manager configuration for conversion 633
Installing VxVM 633
Setting up a Solaris Volume Manager configuration for conversion 634
Run preconvert 634
Run showconvert 635
Run convertname 635
Make backups 637

Converting from the Solaris Volume Manager software to VxVM 637
Reboot the system 638
Change volume references 638
Post conversion tasks 638
Improve volume layouts 638
Remove the Solaris Volume Manager software 639
Converting a root disk 639

Glossary 641

Index 649

Chapter 1

Understanding Veritas Volume Manager

This chapter includes the following topics:

- About Veritas Volume Manager
- VxVM and the operating system
- How VxVM handles storage management
- Volume layouts in VxVM
- Online relayout
- Volume resynchronization
- Dirty region logging
- Volume snapshots
- FastResync
- Hot-relocation
- Volume sets
- Configuration of volumes on SAN storage

About Veritas Volume Manager

Veritas™ Volume Manager (VxVM) by Symantec is a storage management subsystem that allows you to manage physical disks as logical devices called volumes. A VxVM volume appears to applications and the operating system as a

physical device on which file systems, databases and other managed data objects can be configured.

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

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. You can also use VxVM to dynamically configure disk storage while the system is active.

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

- The *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: The CDS feature requires a Veritas Storage Foundation license.

- The *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.
- The *Veritas FlashSnap Point-In-Time Copy Solutions Administrator's Guide* provides guidelines on using the features of the FlashSnap software to implement various point-in-time copy solutions for backup, and database replication.

  Note: The FlashSnap feature requires a separate license.

- The *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.

- The *Veritas Enterprise Administrator User's Guide* 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

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

`vxconfigd`	The VxVM configuration daemon maintains disk and group configurations and communicates configuration changes to the kernel, and modifies configuration information stored on disks.
`vxiod`	VxVM I/O kernel threads provide extended I/O operations without blocking calling processes. By default, 16 I/O threads are started at boot time, and at least one I/O thread must continue to run at all times.
`vxrelocd`	The hot-relocation daemon monitors VxVM for events that affect redundancy, and performs hot-relocation to restore redundancy.

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.

See "How VxVM handles storage management" on page 26.

How VxVM handles storage management

VxVM uses the following types of objects to handle storage management:

Physical objects	Physical disks, LUNs (virtual disks implemented in hardware), or other hardware with block and raw operating system device interfaces that are used to store data. See "Physical objects" on page 26.
Virtual objects	When one or more physical disks are brought under the control of VxVM, it creates virtual objects called volumes on those physical disks. Each volume records and retrieves data from one or more physical disks. Volumes are accessed by file systems, databases, or other applications in the same way that physical disks are accessed. Volumes are also composed of other virtual objects (plexes and subdisks) that are used in changing the volume configuration. Volumes and their virtual components are called virtual objects or VxVM objects. See "Virtual objects" on page 31.

Physical objects

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

Typical device names are of the form `c#t#d#s#`, where `c#` specifies the controller, `t#` specifies the target ID, `d#` specifies the disk, and s# specifies the partition or slice. 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`.

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

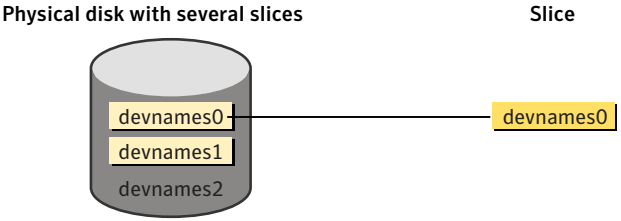
Figure 1-1 Physical disk example

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

Partitions

Figure 1-2 shows how a physical disk can be divided into one or more slices, also known as partitions.

Figure 1-2 Slice example



The slice number is added at the end of the *devname*, and is denoted by `s#`. Note that slice `s2` refers to an entire physical disk for non-EFI disks.

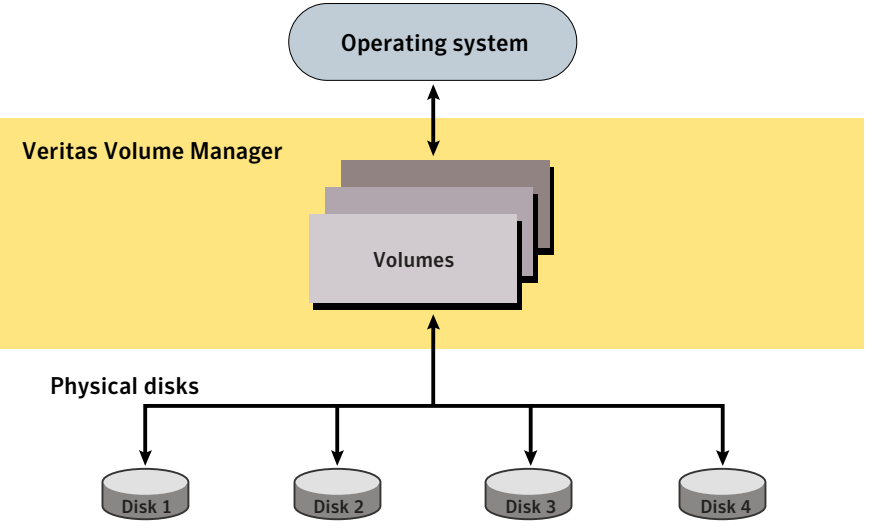
Disk arrays

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

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

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

Figure 1-3 How VxVM presents the disks in a disk array as volumes to the operating system



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

Multiple paths to disk arrays

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

See "How DMP works" on page 163.

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.

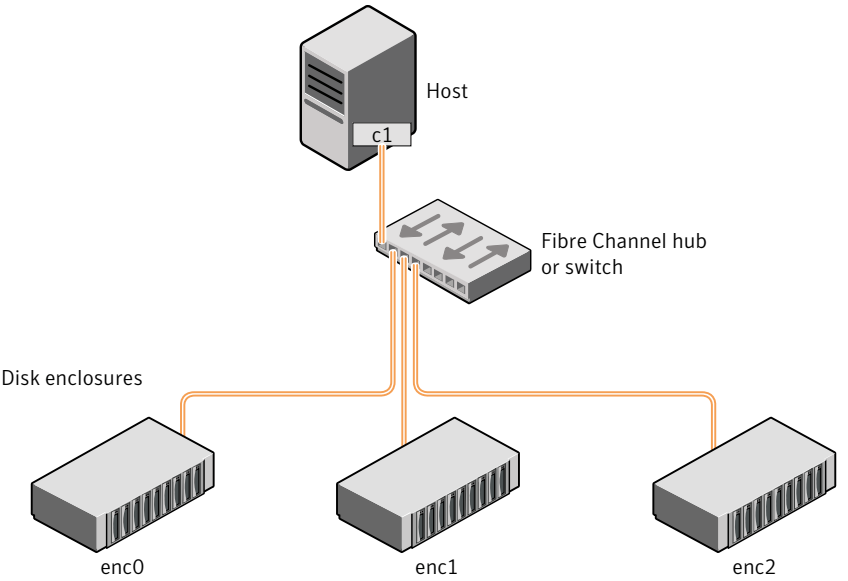
See "How to administer the Device Discovery Layer" on page 93.

Enclosure-based naming

Enclosure-based naming provides an alternative to operating system-based device naming. 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.

Figure 1-4 shows a typical SAN environment where host controllers are connected to multiple enclosures in a daisy chain or through a Fibre Channel hub or fabric switch.

Figure 1-4 Example configuration for disk enclosures connected via a fibre channel hub or switch



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

In 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.

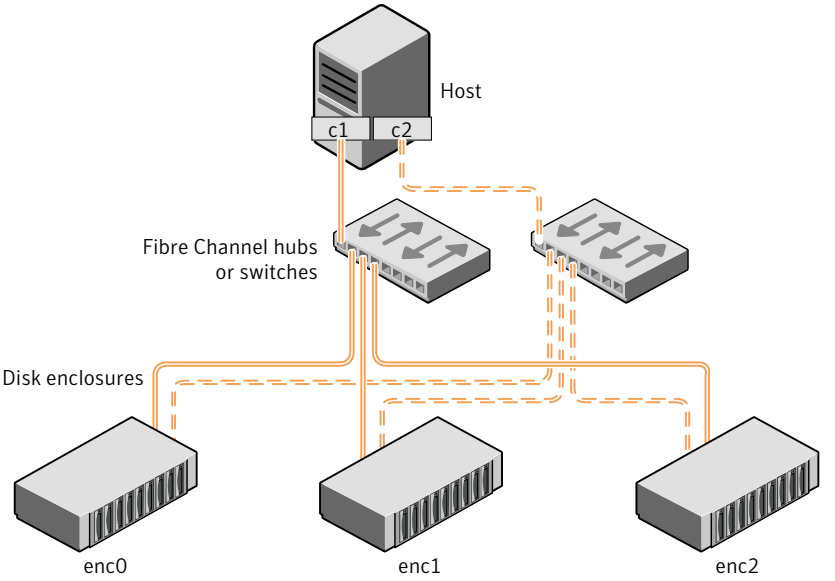
Another important benefit of enclosure-based naming is that it enables VxVM to avoid placing redundant copies of data in the same enclosure. This is a good thing to avoid as each enclosure can be considered to be a separate fault domain. For example, if a mirrored volume were configured only on the disks 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 205.

Figure 1-5 shows a High Availability (HA) configuration where redundant-loop access to storage is implemented by connecting independent controllers on the host to separate hubs with independent paths to the enclosures.

Figure 1-5 Example HA configuration using multiple hubs or switches to provide redundant loop access



Such a configuration protects against the failure of one of the host controllers (`c1` and `c2`), or of the cable between the host and one of the hubs. In this example, each disk is known by the same name to VxVM for all of the paths over which it can be accessed. For example, the disk device `enc0_0` represents a single disk for which two different paths are known to the operating system, such as `c1t99d0` and `c2t99d0`.

See "Disk device naming in VxVM" on page 83.

See "Changing the disk-naming scheme" on page 106.

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

See "Mirroring across targets, controllers or enclosures" on page 318.

Virtual objects

Virtual objects in VxVM include the following:

- Disk groups
  See "Disk groups" on page 33.
- VM disks
  See "VM disks" on page 34.
- Subdisks
  See "Subdisks" on page 35.
- Plexes
  See "Plexes" on page 36.
- Volumes
  See "Volumes" on page 37.

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

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

Bringing the contents of physical disks under VxVM control is accomplished only if VxVM takes control of the physical disks and the disk is not under control of another storage manager such as Sun Microsystems Solaris Volume Manager software.

See "About migration from Solaris Volume Manager" on page 623.

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

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

See "Displaying volume information" on page 328.

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

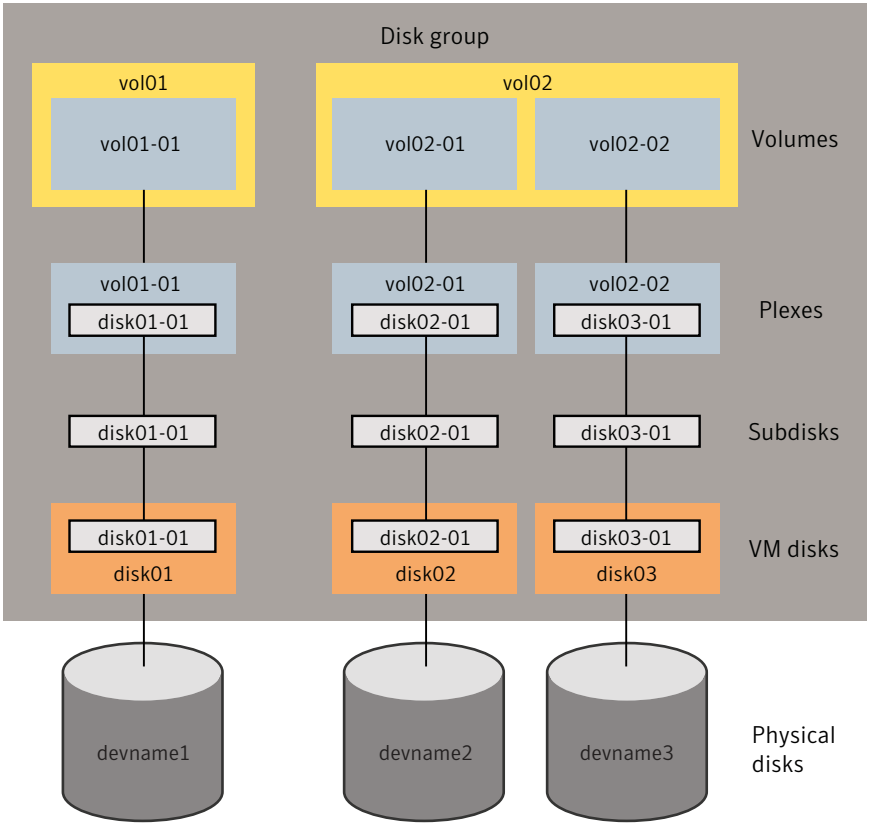
Combining virtual objects in VxVM

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

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

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

Figure 1-6 Connection between objects in VxVM



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

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

Disk groups

A disk group is a collection of disks that share a common configuration, and which are managed by VxVM. A disk group configuration is a set of records with detailed

information about related VxVM objects, their attributes, and their connections. A disk group name can be up to 31 characters long.

See "VM disks" on page 34.

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

See "System-wide reserved disk groups" on page 217.

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

See "Reorganizing the contents of disk groups" on page 250.

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

VM disks

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

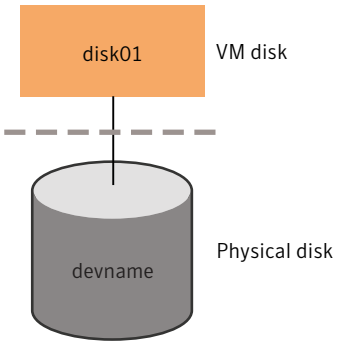
A VM disk typically includes a public region (allocated storage) and a small private region where VxVM internal configuration information is stored.

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

See "Disk groups" on page 33.

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

Figure 1-7 VM disk example



Subdisks

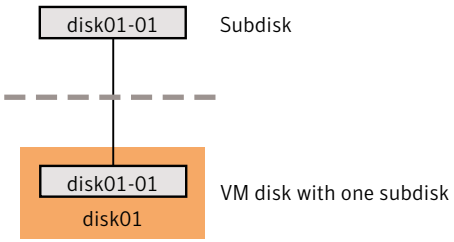
A subdisk is a set of contiguous disk blocks. A block is a unit of space on the disk. VxVM allocates disk space using subdisks. A VM disk can be divided into one or more subdisks. Each subdisk represents a specific portion of a VM disk, which is mapped to a specific region of a physical disk.

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

See "Disk groups" on page 33.

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

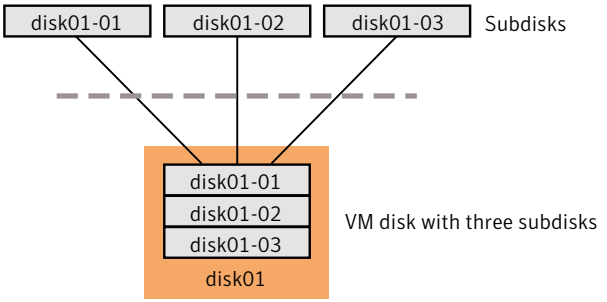
Figure 1-8 Subdisk example



A VM disk can contain multiple subdisks, but subdisks cannot overlap or share the same portions of a VM disk.

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

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



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

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

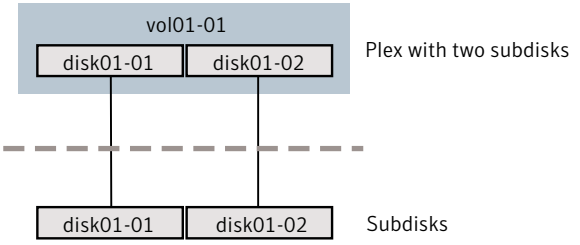
See "Layered volumes" on page 54.

Plexes

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

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

Figure 1-10 Example of a plex with two subdisks



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

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

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

See "Volume layouts in VxVM" on page 39.

Volumes

A volume is a virtual disk device that appears to applications, databases, and file systems like a physical disk device, but does not have the physical limitations of a physical disk device. A volume consists of one or more plexes, each holding a copy of the selected data in the volume. Due to its virtual nature, a volume is not restricted to a particular disk or a specific area of a disk. The configuration of a volume can be changed by using VxVM user interfaces. Configuration changes can be accomplished without causing disruption to applications or file systems that are using the volume. For example, a volume can be mirrored on separate disks or moved to use different disk storage.

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

A volume may be created under the following constraints:

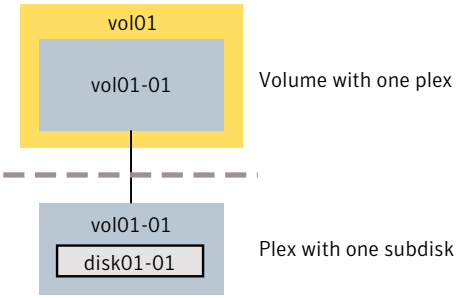
- Its name can contain up to 31 characters.
- It can consist of up to 32 plexes, each of which contains one or more subdisks.
- It must have at least one associated plex that has a complete copy of the data in the volume with at least one associated subdisk.
- All subdisks within a volume must belong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

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.

See the *Veritas Storage Foundation Intelligent Storage Provisioning Administrator's Guide*.

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

Figure 1-11 Example of a volume with one plex

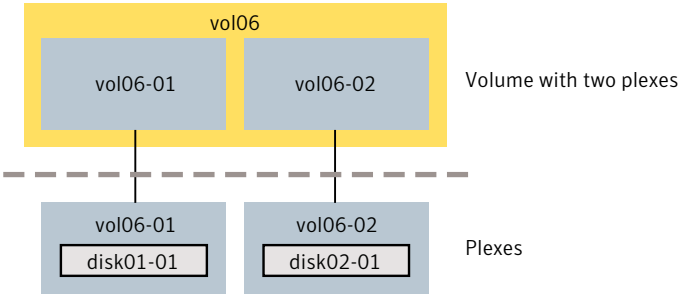


The volume `vol01` has the following characteristics:

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

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

Figure 1-12 Example of a volume with two plexes



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

The volume `vol06` has the following characteristics:

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

See "Mirroring (RAID-1)" on page 46.

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 54.

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
  See "Concatenation and spanning" on page 40.
- Striping (RAID-0)
  See "Striping (RAID-0)" on page 42.
- Mirroring (RAID-1)
  See "Mirroring (RAID-1)" on page 46.
- Striping plus mirroring (mirrored-stripe or RAID-0+1)
  See "Striping plus mirroring (mirrored-stripe or RAID-0+1)" on page 47.
- Mirroring plus striping (striped-mirror, RAID-1+0 or RAID-10)
  See "Mirroring plus striping (striped-mirror, RAID-1+0 or RAID-10)" on page 48.
- RAID-5 (striping with parity)
  See "RAID-5 (striping with parity)" on page 49.

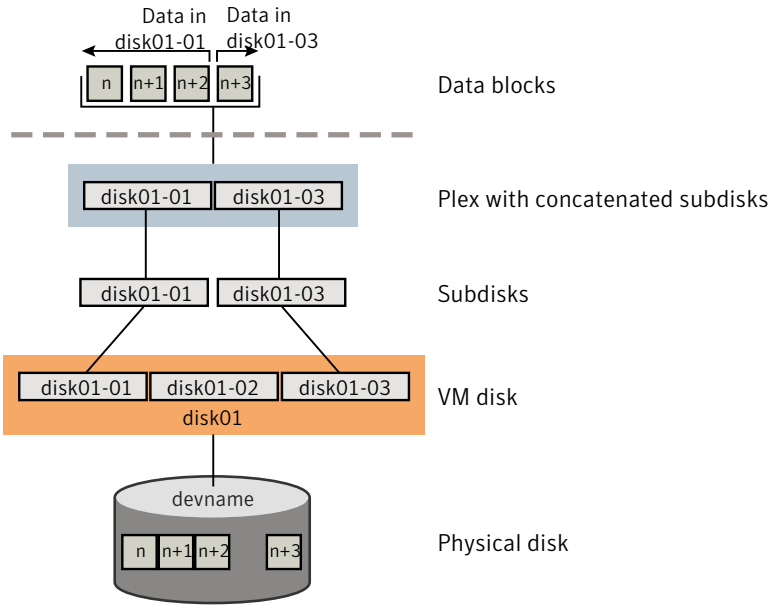
Concatenation and spanning

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.

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

Figure 1-13 Example of concatenation



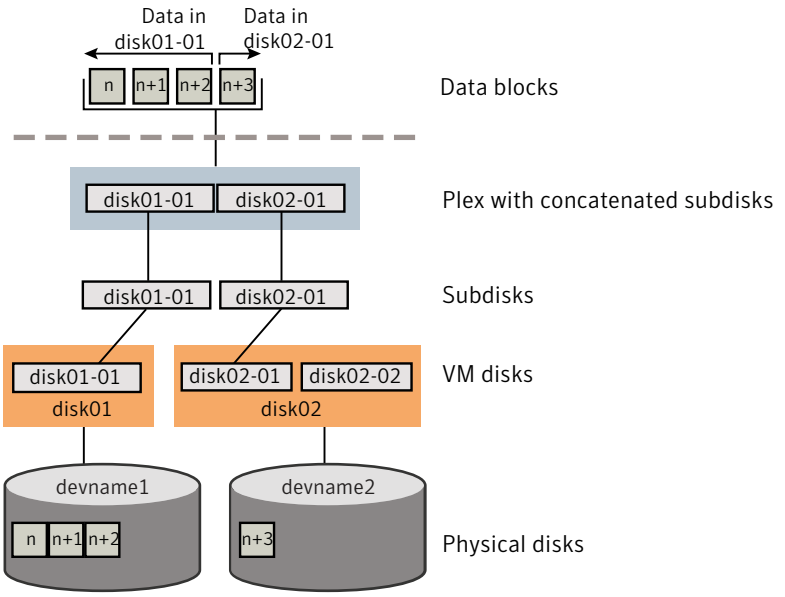
The blocks `n`, `n+1`, `n+2` and `n+3` (numbered relative to the start of the plex) are contiguous on the plex, but actually come from two distinct subdisks on the same physical disk.

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

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

Figure 1-14 shows data spread over two subdisks in a spanned plex.

Figure 1-14 Example of spanning



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

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

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

Striping (RAID-0)

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

Striping maps data so that the data is interleaved among two or more physical disks. A striped plex contains two or more subdisks, spread out over two or more

physical disks. Data is allocated alternately and evenly to the subdisks of a striped plex.

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

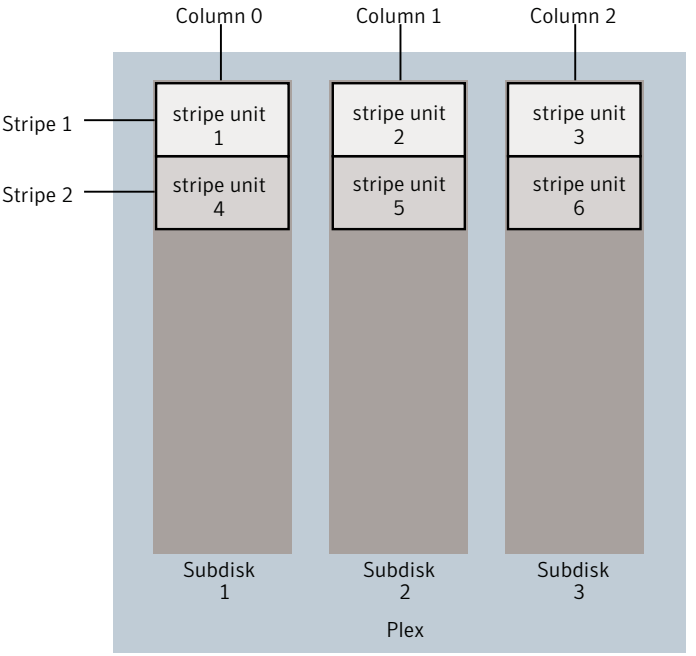
Warning: Striping a volume, or splitting a volume across multiple disks, increases the chance that a disk failure will result in failure of that volume.

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

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

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

Figure 1-15 Striping across three columns



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

Viewed in sequence, the first stripe consists of:

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

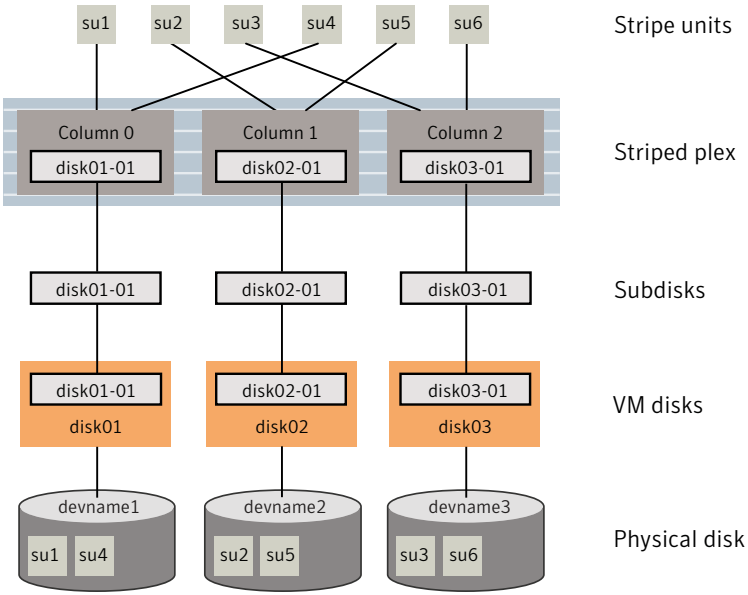
The second stripe consists of:

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

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

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

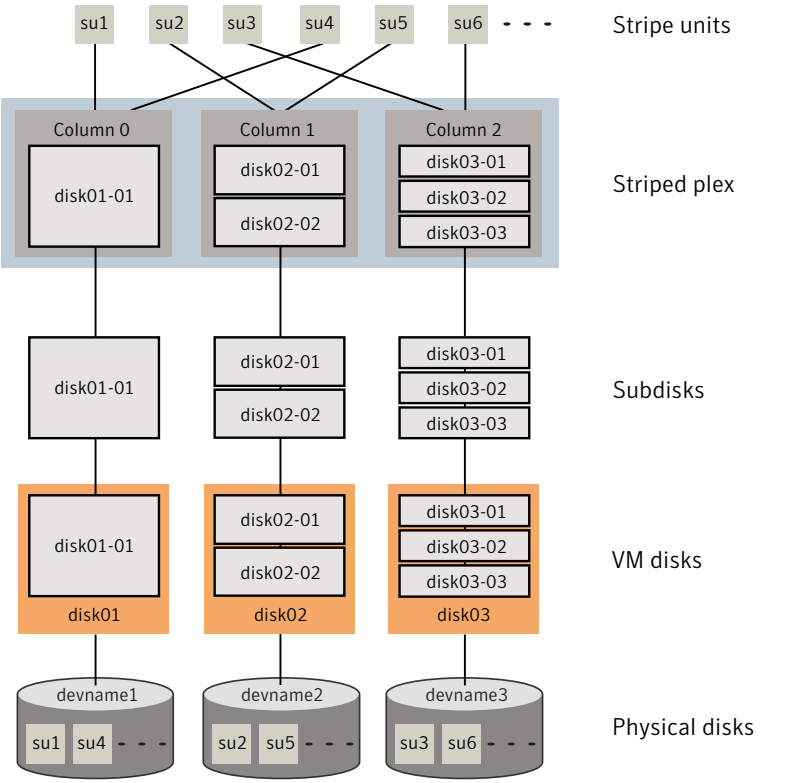
Figure 1-16 Example of a striped plex with one subdisk per column



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

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

Figure 1-17 Example of a striped plex with concatenated subdisks per column



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

See "Creating a striped volume" on page 316.

Mirroring (RAID-1)

Note: You need a full license to use this feature with disks other than the root disk.

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.

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 310.

Disk duplexing, in which each mirror exists on a separate controller, is also supported.

See "Mirroring across targets, controllers or enclosures" on page 318.

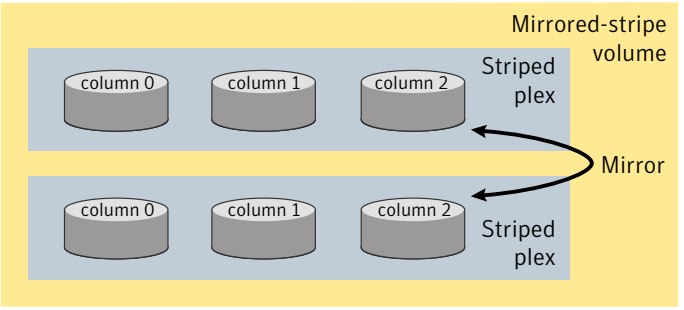
Striping plus mirroring (mirrored-stripe or RAID-0+1)

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

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

Figure 1-18 shows an example where two plexes, each striped across three disks, are attached as mirrors to the same volume to create a mirrored-stripe volume.

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



See "Creating a mirrored-stripe volume" on page 317.

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

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

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

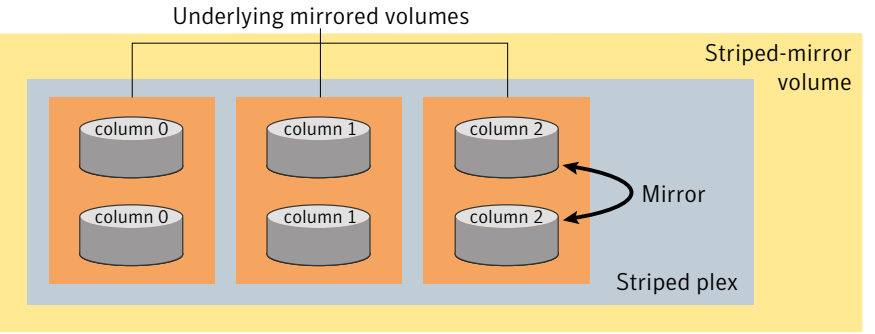
A striped-mirror volume is an example of a layered volume.

See "Layered volumes" on page 54.

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

Figure 1-19 shows an example where a striped-mirror volume is created by using each of three existing 2-disk mirrored volumes to form a separate column within a striped plex.

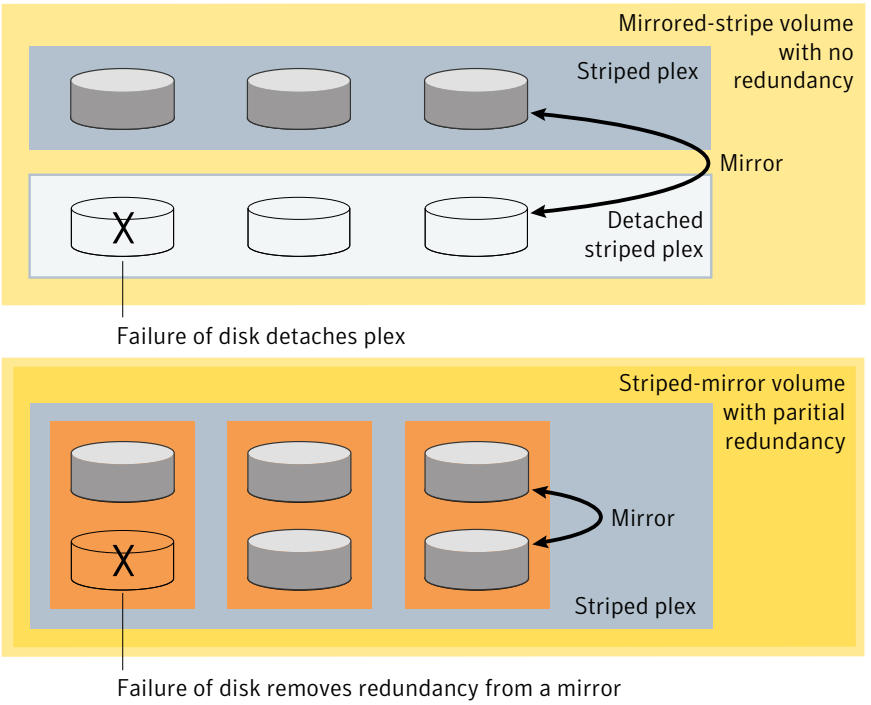
Figure 1-19 Striped-mirror volume laid out on six disks



See "Creating a striped-mirror volume" on page 318.

Figure 1-20 shows that the failure of a disk in a mirrored-stripe layout detaches an entire data plex, thereby losing redundancy on the entire volume.

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



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

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

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

RAID-5 (striping with parity)

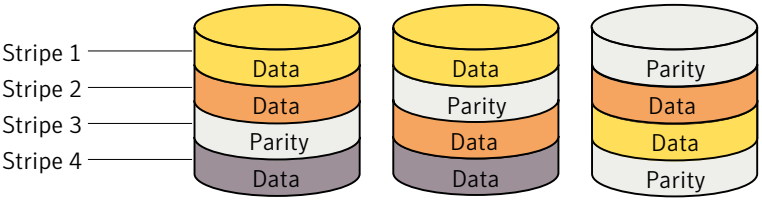
Although both mirroring (RAID-1) and RAID-5 provide redundancy of data, they use different methods. Mirroring provides data redundancy by maintaining multiple complete copies of the data in a volume. Data being written to a mirrored

volume is reflected in all copies. If a portion of a mirrored volume fails, the system continues to use the other copies of the data.

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

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

Figure 1-21 Parity locations in a RAID-5 model



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

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

See "Veritas Volume Manager RAID-5 arrays" on page 51.

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.

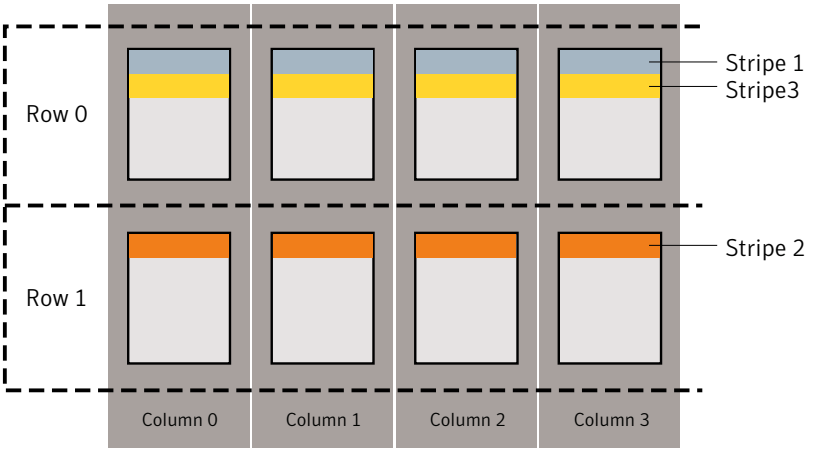
Traditional RAID-5 arrays

A traditional RAID-5 array is several disks organized in rows and columns. A column is a number of disks located in the same ordinal position in the array. A

row is the minimal number of disks necessary to support the full width of a parity stripe.

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

Figure 1-22 Traditional RAID-5 array



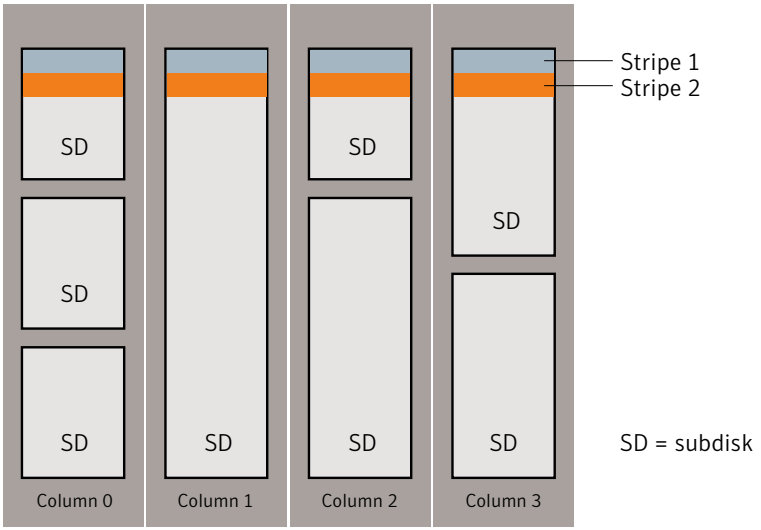
This traditional array structure supports growth by adding more rows per column. Striping is accomplished by applying the first stripe across the disks in Row 0, then the second stripe across the disks in Row 1, then the third stripe across the Row 0 disks, and so on. This type of array requires all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

Veritas Volume Manager RAID-5 arrays

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

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

Figure 1-23 Veritas Volume Manager RAID-5 array



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

See "Striping (RAID-0)" on page 42.

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

See "Creating a RAID-5 volume" on page 319.

Left-symmetric layout

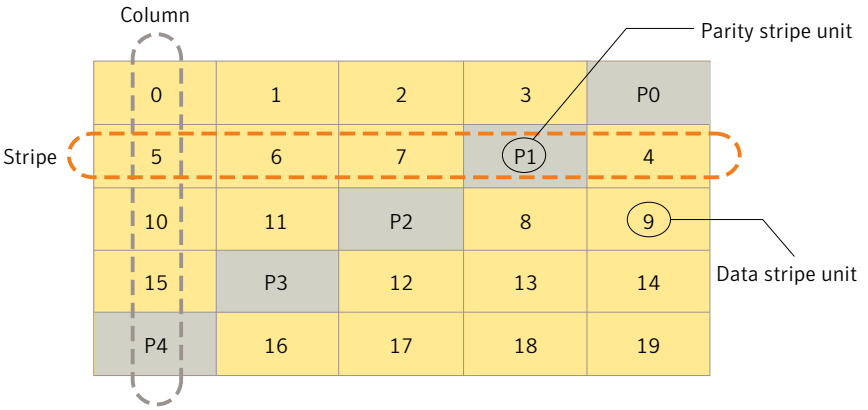
There are several layouts for data and parity that can be used in the setup of a RAID-5 array. The implementation of RAID-5 in VxVM uses a left-symmetric layout. This provides optimal performance for both random I/O operations and large sequential I/O operations. However, the layout selection is not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

Left-symmetric layout stripes both data and parity across columns, placing the parity in a different column for every stripe of data. The first parity stripe unit is located in the rightmost column of the first stripe. Each successive parity stripe

unit is located in the next stripe, shifted left one column from the previous parity stripe unit location. If there are more stripes than columns, the parity stripe unit placement begins in the rightmost column again.

Figure 1-24 shows a left-symmetric parity layout with five disks (one per column).

Figure 1-24 Left-symmetric layout



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

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

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

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

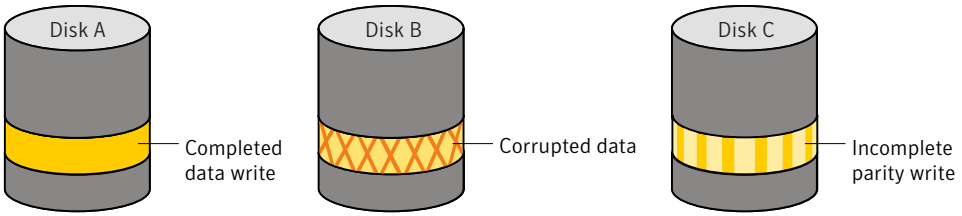
RAID-5 logging

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

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

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

Figure 1-25 Incomplete write to a RAID-5 volume



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

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

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

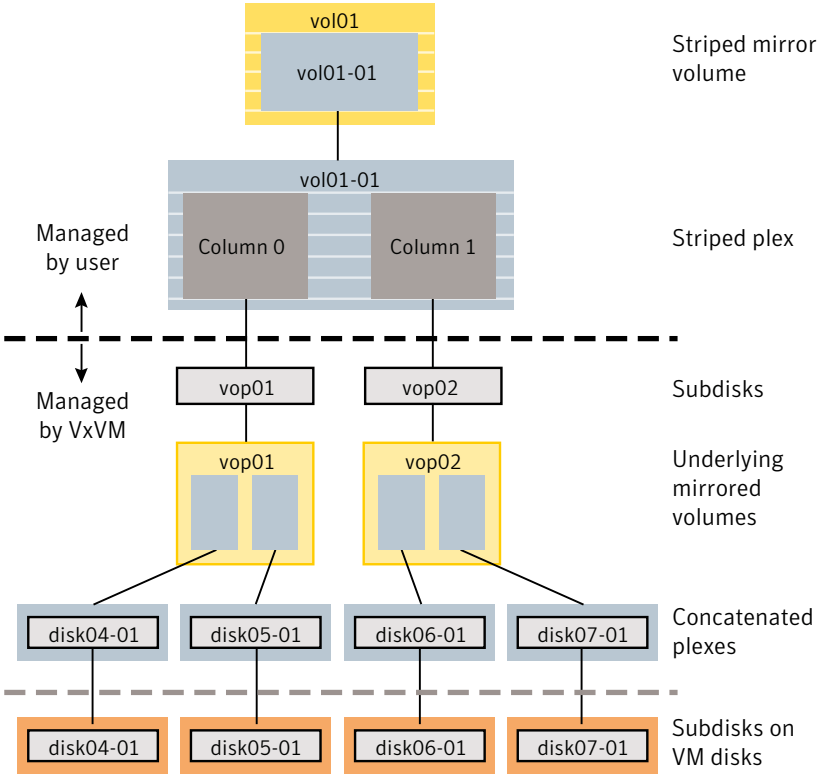
See "Adding a RAID-5 log" on page 351.

Layered volumes

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

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

Figure 1-26 Example of a striped-mirror layered volume



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

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

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

Creating striped-mirrors	See "Creating a striped-mirror volume" on page 318. See the `vxassist`(1M) manual page.
Creating concatenated-mirrors	See "Creating a concatenated-mirror volume" on page 311. See the `vxassist`(1M) manual page.
Online Relayout	See "Online relayout" on page 56. See the `vxassist`(1M) manual page. See the `vxrelayout`(1M) manual page.
Moving RAID-5 subdisks	See the `vxsd`(1M) manual page.
Creating Snapshots	See "About volume snapshots" on page 373. See the `vxassist`(1M) manual page. See the `vxsnap`(1M) manual page.

Online relayout

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

See "Performing online relayout" on page 363.

See "Converting between layered and non-layered volumes" on page 370.

How online relayout works

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

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

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

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

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

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

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

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

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

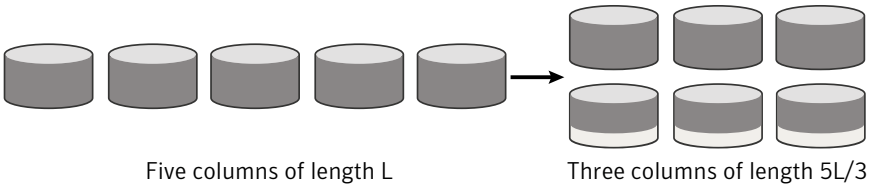
See the `vxassist`(1M) manual page.

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

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

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

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

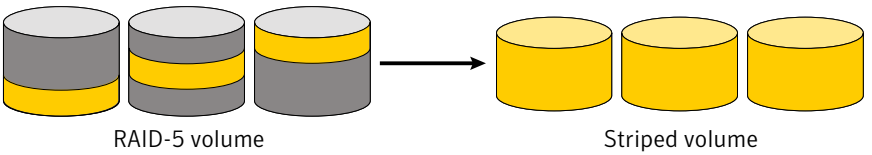


Note that the size of the volume remains the same but an extra disk is needed to extend one of the columns.

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

- Remove parity from a RAID-5 volume to change it to a concatenated, striped, or layered volume.
  Figure 1-28 shows an example of applying relayout a RAID-5 volume.

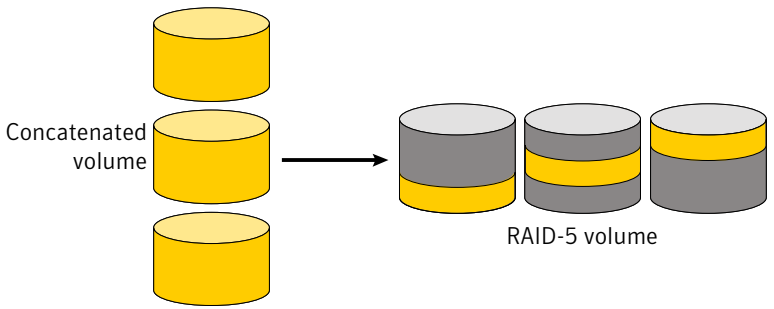
Figure 1-28 Example of relayout of a RAID-5 volume to a striped volume



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

- Add parity to a volume to change it to a RAID-5 volume.
  Figure 1-29 shows an example.

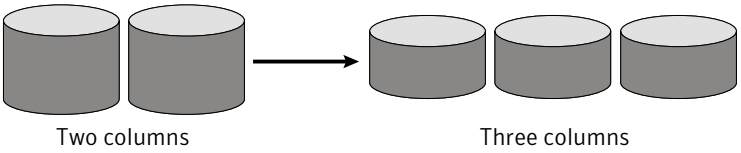
Figure 1-29 Example of relayout of a concatenated volume to a RAID-5 volume



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

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

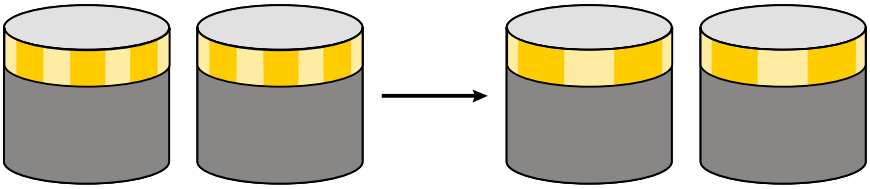
Figure 1-30 Example of increasing the number of columns in a volume



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

- Change the column stripe width in a volume.
  Figure 1-31 shows an example of changing the column stripe width.

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



See "Performing online relayout" on page 363.

See "Permitted relayout transformations" on page 364.

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 370.
- 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 a plex that is not the same size as the volume, or that has regions that are not mapped to any subdisk.)
- The number of mirrors in a mirrored volume cannot be changed using relayout.
- Only one relayout may be applied to a volume at a time.

Transformation characteristics

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

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

You can reverse the layout transformation process at any time, but the data may not be returned to the exact previous storage location. 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` command to shrink or grow a file system.

See “Resizing a volume” on page 353.

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" on page 62.

See "FastResync" on page 68.

For raw volumes used by database applications, the SmartSync feature can be used if this is supported by the database vendor.

See "SmartSync recovery accelerator" on page 63.

Dirty region logging

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

If a version 20 DCO volume is associated with a volume, a portion of the DCO volume can be used to store the DRL log. There is no need to create a separate DRL log for a volume which has a version 20 DCO volume.

See "DCO volume versioning" on page 70.

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 DMP tunable parameters .

See "Adding traditional DRL logging to a mirrored volume" on page 349.

See "Preparing a volume for DRL and instant snapshots" on page 341.

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: The SmartSync feature of Veritas Volume Manager is only applicable to databases that are configured on raw volumes. You cannot use it with volumes that contain file systems. Use an alternative solution such as the Oracle Resilvering feature of Veritas File System (VxFS).

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

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

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

To enable the use of SmartSync with database volumes in shared disk groups, set the value of the `volcvm_smartsync` tunable to `1`.

See "Tunable parameters for VxVM" on page 556.

Data volume configuration

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

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

Redo log volume configuration

A redo log is a log of changes to the database data. Because the database does not maintain changes to the redo logs, it cannot provide information about which sections require resilvering. Redo logs are also written sequentially, and since traditional dirty region logs are most useful with randomly-written data, they are

of minimal use for reducing recovery time for redo logs. However, VxVM can reduce the number of dirty regions by modifying the behavior of its dirty region logging feature to take advantage of sequential access patterns. Sequential DRL decreases the amount of data needing recovery and reduces recovery time impact on the system.

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

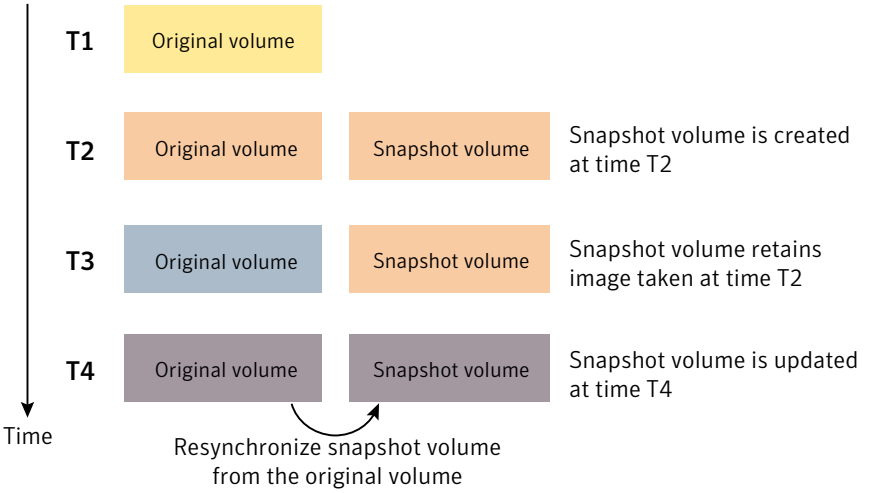
See "Sequential DRL" on page 63.

Volume snapshots

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

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

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



Even though the contents of the original volume can change, the snapshot volume 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.

Warning: 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.

The traditional 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.

See "Traditional third-mirror break-off snapshots" on page 375.

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

See "FastResync" on page 68.

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

See "Full-sized instant snapshots" on page 376.

See "Space-optimized instant snapshots" on page 378.

See "Emulation of third-mirror break-off snapshots" on page 379.

See "Linked break-off snapshot volumes" on page 380.

See "Comparison of snapshot features" on page 67.

See "About volume snapshots" on page 373.

See the vxassist(1M) manual page.

See the vxsnap(1M) manual page.

Comparison of snapshot features

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

Table 1-1 Comparison of snapshot features for supported snapshot types

Snapshot feature	Full-sized instant (vxsnap)	Space-optimized instant (vxsnap)	Break-off (vxassist or vxsnap)
Immediately available for use on creation	Yes	Yes	No
Requires less storage space than original volume	No	Yes	No
Can be reattached to original volume	Yes	No	Yes
Can be used to restore contents of original volume	Yes	Yes	Yes
Can quickly be refreshed without being reattached	Yes	Yes	No
Snapshot hierarchy can be split	Yes	No	No
Can be moved into separate disk group from original volume	Yes	No	Yes
Can be turned into an independent volume	Yes	No	Yes
FastResync ability persists across system reboots or cluster restarts	Yes	Yes	Yes
Synchronization can be controlled	Yes	No	No

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

If storage space is at a premium, space-optimized instant snapshots can be configured with some reduction of supported functionality. For example, space-optimized snapshots cannot be turned into independent volumes, nor can they be moved into a separate disk group for off-host processing.

FastResync

Note: You need a license to use this feature.

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

See "Enabling FastResync on a volume" on page 361.

FastResync enhancements

FastResync provides the following enhancements to VxVM:

Faster mirror resynchronization	FastResync optimizes mirror resynchronization by keeping track of updates to stored data that have been missed by a mirror. (A mirror may be unavailable because it has been detached from its volume, either automatically by VxVM as the result of an error, or directly by an administrator using a utility such as `vxplex` or `vxassist`. A returning mirror is a mirror that was previously detached and is in the process of being re-attached to its original volume as the result of the `vxrecover` or `vxplex att` operation.) When a mirror returns to service, only the updates that it has missed need to be re-applied to resynchronize it. This requires much less effort than the traditional method of copying all the stored data to the returning mirror. Once FastResync has been enabled on a volume, it does not alter how you administer mirrors. The only visible effect is that repair operations conclude more quickly.
Re-use of snapshots	FastResync allows you to refresh and re-use snapshots rather than discard them. You can quickly re-associate (snap back) snapshot plexes with their original volumes. This reduces the system overhead required to perform cyclical operations such as backups that rely on the snapshot functionality of VxVM.

Non-persistent FastResync

Non-persistent FastResync allocates its change maps in memory. If non-persistent FastResync is enabled, a separate FastResync map is kept for the original volume and for each snapshot volume. Unlike a dirty region log (DRL), they do not reside on disk nor in persistent store. This has the advantage that updates to the FastResync map have little impact on I/O performance, as no disk updates needed to be performed. However, if a system is rebooted, the information in the map is lost, so a full resynchronization is required on snapback. This limitation can be overcome for volumes in cluster-shareable disk groups, provided that at least one of the nodes in the cluster remained running to preserve the FastResync map in its memory. However, a node crash in a High Availability (HA) environment requires the full resynchronization of a mirror when it is reattached to its parent volume.

How non-persistent FastResync works with snapshots

The snapshot feature of VxVM takes advantage of FastResync change tracking to record updates to the original volume after a snapshot plex is created. After a snapshot is taken, the `snapback` option is used to reattach the snapshot plex.

Provided that FastResync is enabled on a volume before the snapshot is taken, and that it is not disabled at any time before the snapshot is reattached, the changes that FastResync records are used to resynchronize the volume during the snapback. This considerably reduces the time needed to resynchronize the volume.

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

See "DMP tunable parameters " on page 565.

Persistent FastResync

Unlike non-persistent FastResync, persistent FastResync keeps the FastResync maps on disk so that they can survive system reboots, system crashes and cluster crashes. Persistent FastResync can also track the association between volumes and their snapshot volumes after they are moved into different disk groups. When the disk groups are rejoined, this allows the snapshot plexes to be quickly resynchronized. This ability is not supported by non-persistent FastResync.

See "Reorganizing the contents of disk groups" on page 250.

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

DCO volume versioning

The internal layout of the DCO volume changed in VxVM 4.0 to support new features such as full-sized and space-optimized instant snapshots. Because the DCO volume layout is versioned, VxVM software continues to support the version 0 layout for legacy volumes. However, you must configure a volume to have a version 20 DCO volume if you want to take instant snapshots of the volume. Future releases of Veritas Volume Manager may introduce new versions of the DCO volume layout.

See "Determining the DCO version number" on page 344.

Version 0 DCO volume layout

In earlier releases of VxVM, the DCO object only managed information about the FastResync maps. These maps track writes to the original volume and to each of up to 32 snapshot volumes since the last `snapshot` operation. Each plex of the DCO volume on disk holds 33 maps, each of which is 4 blocks in size by default.

Persistent FastResync uses the maps in a version 0 DCO volume on disk to implement change tracking. As for non-persistent FastResync, each bit in the map represents a region (a contiguous number of blocks) in a volume's address space. The size of each map can be changed by specifying the `dcolen` attribute to the `vxassist` command when the volume is created. The default value of `dcolen` is 132 512-byte blocks (the plex contains 33 maps, each of length 4 blocks). To use a larger map size, multiply the desired map size by 33 to calculate the value of `dcolen` that you need to specify. For example, to use an 8-block map, you would specify `dcolen=264`. The maximum possible map size is 64 blocks, which corresponds to a `dcolen` value of 2112 blocks.

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

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

Version 20 DCO volume layout

In VxVM 4.0 and later releases, the DCO object is used not only to manage the FastResync maps, but also to manage DRL recovery maps and special maps called copymaps that allow instant snapshot operations to resume correctly following a system crash.

See "Dirty region logging" on page 62.

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

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

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

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

where the size of each map in bytes is:

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

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

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

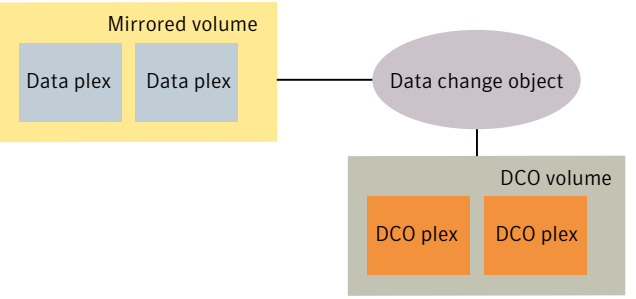
Note: Full-sized and space-optimized instant snapshots, which are administered using the `vxsnap` command, are supported for a version 20 DCO volume layout. The use of the `vxassist` command to administer traditional (third-mirror break-off) snapshots is not supported for a version 20 DCO volume layout.

How persistent FastResync works with snapshots

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

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

Figure 1-33 Mirrored volume with persistent FastResync enabled

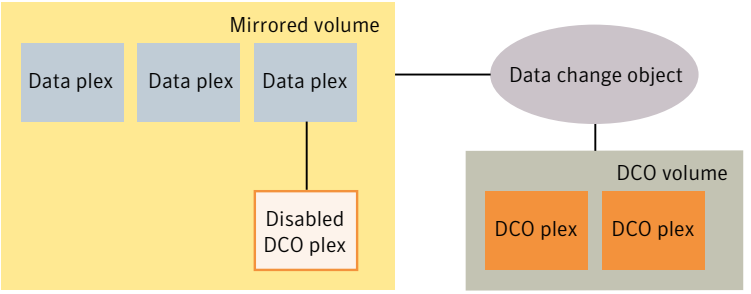


Associated with the volume are a DCO object and a DCO volume with two plexes.

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

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

Figure 1-34 Mirrored volume after completion of a snapstart operation



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

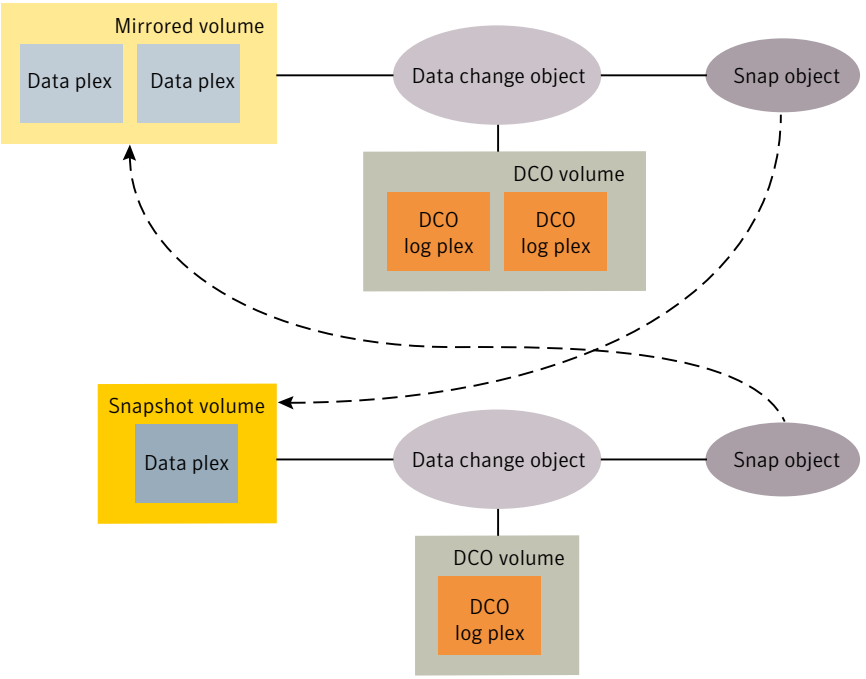
Space-optimized instant snapshots do not require additional full-sized plexes to be created. Instead, they use a storage cache that typically requires only 10% of the storage that is required by full-sized snapshots. There is a trade-off in functionality in using space-optimized snapshots. The storage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

See "Comparison of snapshot features" on page 67.

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

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

Figure 1-35 Mirrored volume and snapshot volume after completion of a snapshot operation



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

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

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

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

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

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

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

See "Space-optimized instant snapshots" on page 378.

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

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

Effect of growing a volume on the FastResync map

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

- For a version 20 DCO volume, the size of the map is increased and the size of the region that is tracked by each bit in the map stays the same.
- For a version 0 DCO volume, the size of the map remains the same and the region size is increased.

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

FastResync limitations

The following limitations apply to FastResync:

- Persistent FastResync is supported for RAID-5 volumes, but this prevents the use of the relayout or resize operations on the volume while a DCO is associated with it.
- Neither non-persistent nor persistent FastResync can be used to resynchronize mirrors after a system crash. Dirty region logging (DRL), which can coexist with FastResync, should be used for this purpose. In VxVM 4.0 and later releases, DRL logs may be stored in a version 20 DCO volume.
- When a subdisk is relocated, the entire plex is marked “dirty” and a full resynchronization becomes necessary.
- If a snapshot volume is split off into another disk group, non-persistent FastResync cannot be used to resynchronize the snapshot plexes with the original volume when the disk group is rejoined with the original volume’s disk group. Persistent FastResync must be used for this purpose.
- If you move or split an original volume (on which persistent FastResync is enabled) into another disk group, and then move or join it to a snapshot volume’s disk group, you cannot use `vxassist snapback` to resynchronize traditional snapshot plexes with the original volume. This restriction arises because a snapshot volume references the original volume by its record ID at the time that the snapshot volume was created. Moving the original volume to a different disk group changes the volume’s record ID, and so breaks the association. However, in such a case, you can use the `vxplex snapback` command with the `-f` (force) option to perform the snapback.

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

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

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

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

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

Hot-relocation

Note: You need a 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 or to free space within the disk group. VxVM then reconstructs the objects that existed before the failure and makes them accessible again.

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

See "How hot-relocation works" on page 456.

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.

See "Creating a volume set" on page 436.

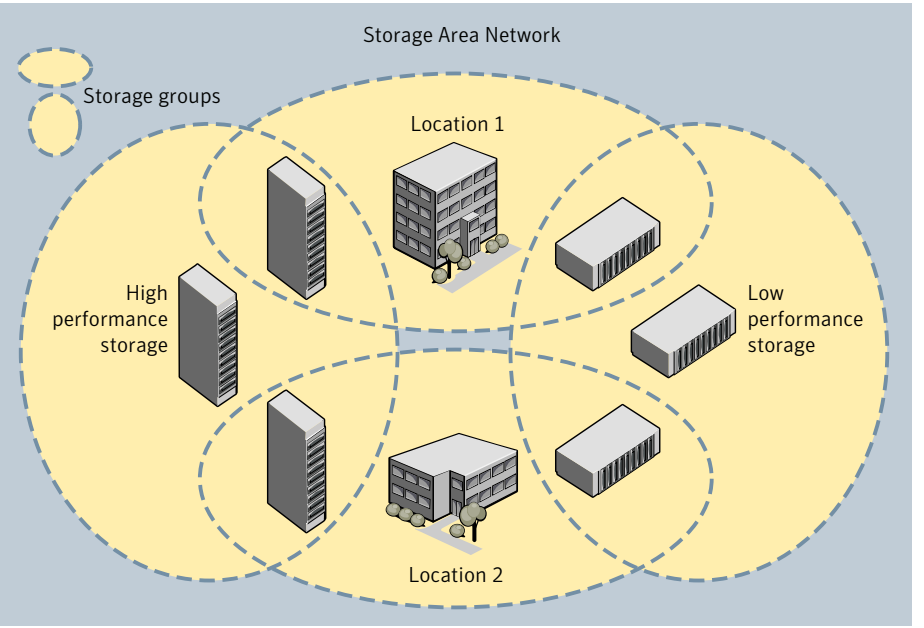
Configuration of volumes on SAN storage

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

Note: This feature of `vxassist` is designed to work in conjunction with SAL (SAN Access Layer) in Veritas CommandCentral Storage. When VxVM with SAN-aware `vxassist` is installed on a host where SAL is also installed, it is recommended that you create a user named `root` under SAL. This allows `vxassist` to use the `root` login to contact the SAL daemon (`sald`) on the primary SAL server without needing to specify the `sal_username` attribute to `vxassist`.

Figure 1-36, shows how you might choose to set up storage groups within a SAN.

Figure 1-36 Dividing a Storage Area Network into storage groups



In this example, the boundaries of the storage groups are based on the performance characteristics of different makes of disk array and on geographic location.

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

To use the CommandCentral Storage storage groups with `vxassist`, perform the following steps in the order listed:

- Use the CommandCentral Storage software to define one or more storage groups. Note that zoning is not an issue as it is completely independent of storage group creation.
- Use the CommandCentral Storage software to attach attribute-value pairs to each storage group's property sheet. Typically, you would assign values for the following attributes: location, storage group, and protection.
- Use the `vxspcshow` command to discover the device names of disks that have a specified set of attributes, or to list the attributes of specified disks.
- Use the `vxdiskadm` command or the VEA to configure the disks that you found in the previous step into VxVM disk groups.
- Use `vxassist` to create volumes on disks that are selected by matching specified criteria for the values of storage group attributes. The usual restriction applies that a volume may only be created using disks from a single disk group.

Chapter 2

Administering disks

This chapter includes the following topics:

- About disk management
- Disk devices
- Discovering and configuring newly added disk devices
- Disks under VxVM control
- Changing the disk-naming scheme
- Discovering the association between enclosure-based disk names and OS-based disk names
- Disk installation and formatting
- Displaying or changing default disk layout attributes
- Adding a disk to VxVM
- RAM disk support in VxVM
- Encapsulating a disk
- Rootability
- Unencapsulating the root disk
- Dynamic LUN expansion
- Extended Copy Service
- Removing disks
- Removing a disk from VxVM control
- Removing and replacing disks

- Enabling a disk
- Taking a disk offline
- Renaming a disk
- Reserving disks
- Displaying disk information
- Changing host LUN configurations online

About disk management

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

Note: Most VxVM commands require superuser or equivalent privileges.

Disks that are controlled by the Sun Microsystems Solaris Volume Manager software cannot be used directly as VxVM disks, but the disks can be converted so that their volumes become VxVM volumes.

See "About migration from Solaris Volume Manager" on page 623.

The dynamic multipathing (DMP) feature of VxVM is used to administer multiported disk arrays.

See "How DMP works" on page 163.

Disk devices

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

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

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

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

VxVM uses the device names to create 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 one or more physical paths, perhaps via different controllers. The number of access paths that are available depends on whether the disk is a single disk, or is part of a multiported disk array that is connected to a system.

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

See "How DMP works" on page 163.

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

See "Disk device naming in VxVM" on page 83.

Disk device naming in VxVM

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

See "Disk categories" on page 90.

Device names can use one of the following naming schemes:

- Operating system-based naming
- Enclosure-based naming

Devices with very long device names (longer than 31 characters) are always represented by enclosure-based names.

You can change the disk-naming scheme if required.

See "Changing the disk-naming scheme" on page 106.

Operating system-based naming

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

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

Note: 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.

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:

- Disks 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.

OS-based names can be made persistent, so that they do not change after reboot. By default, OS-based names are not persistent, and are regenerated if the system configuration changes the device name as recognized by the operating system.

Enclosure-based naming

Enclosure-based naming operates as follows:

- All fabric or non-fabric disks in supported disk arrays are named using the *`enclosure_name_#`* format. For example, disks in the supported disk array, `enggdept` are named `enggdept_0`, `enggdept_1`, `enggdept_2` and so on.
  You can use the `vxdmpadm` command to administer enclosure names.
  See "Renaming an enclosure" on page 205.
  See the `vxdmpadm`(1M) manual page.
- 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.

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

If a CVM cluster is symmetric, each node in the cluster accesses the same set of disks. Enclosure-based names provide a consistent naming system so that the device names are the same on each node.

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

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

See "Renaming an enclosure" on page 205.

See "Disk categories" on page 90.

Private and public disk regions

Most VM disks consist of the following 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 32 megabytes, which is large enough to record the details of several thousand VxVM objects in a disk group.

  Under most circumstances, the default private region size should be sufficient. For administrative purposes, it is usually much simpler to create more disk groups that contain fewer volumes, or to split large disk groups into several smaller ones.

  See "Splitting disk groups" on page 260.

  If required, the value for the private region size may be overridden when you add or replace a disk using the `vxdiskadm` command.

  Each disk that has a private region holds an entire copy of the configuration database for the disk group. The size of the configuration database for a disk group is limited by the size of the smallest copy of the configuration database on any of its member disks.

public region
: An area that covers the remainder of the disk, and which is used for the allocation of storage space to subdisks.

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

The following disk access types are used by VxVM:

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

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

`cdsdisk`	The disk is formatted as a Cross-platform Data Sharing (CDS) disk that is suitable for moving between different operating systems. This is the default format for disks that are not used to boot the system.Typically, most disks on a system are configured as this disk type. However, it is not a suitable format for boot, root or swap disks, for mirrors or hot-relocation spares of such disks, or for Extensible Firmware Interface (EFI) disks.
`simple`	The disk is formatted as a simple disk that can be converted to a CDS disk.
`sliced`	The disk is formatted as a sliced disk. This format can be applied to disks that are used to boot the system. The disk can be converted to a CDS disk if it was not initialized for use as a boot disk.

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

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

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

By default, auto-configured non-EFI 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.

Auto-configured EFI disks are formatted as `sliced` disks by default.

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. An exception is an encapsulated root disk, on which `s3` is usually configured as the public region and `s4` as the private region.

See "Displaying or changing default disk layout attributes" on page 113.

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

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.

If you want VxVM to scan only for new devices that have been added to the system, and not for devices that have been enabled or disabled, specify the `-f` option to either of the commands, as shown here:

```
# vxdctl -f enable
# vxdisk -f scandisks
```

However, a complete scan is initiated if the system configuration has been modified by changes to:

- Installed array support libraries.
- The devices that are listed as being excluded from use by VxVM.
- DISKS (JBOD), SCSI3, or foreign device definitions.

See the vxdctl(1M) manual page.

See the vxdisk(1M) manual page.

Partial device discovery

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

```
# vxdisk scandisks new
```

The next example discovers fabric devices:

```
# vxdisk scandisks fabric
```

The above command discovers devices with the characteristic DDI_NT_FABRIC property set on them.

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/
```

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

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

Discovering disks and dynamically adding disk arrays

DMP uses array support libraries (ASLs) to provide array-specific support for multipathing. 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. DMP provides device discovery layer (DDL) to determine which ASLs should be associated to each disk array.

In some cases, DMP can also provide basic multipathing and failover functionality by treating LUNs as disks (JBODs).

How DMP claims devices

For fully optimized support of any array and for support of more complicated array types, DMP requires the use of array-specific array support libraries (ASLs), possibly coupled with an array policy modules (APMs). ASL and APMs effectively are array-specific plugins that allow close tie-in of DMP with any specific array model.

See the HCL for the complete list of supported arrays.

During device discovery, the DDL checks the installed ASL for each device to find which ASL claims the device. If no ASL is found to claim the device, the DDL checks for a corresponding JBOD definition. You can add JBOD definitions for unsupported arrays to enable DMP to provide multipathing for the array. If a JBOD definition is found, the DDL claims the devices in the DISKS category, which adds the LUNs to the list of JBOD (physical disk) devices used by DMP. If the JBOD definition includes a cabinet number, DDL uses the cabinet number to group the LUNs into enclosures.

See “Adding unsupported disk arrays to the DISKS category” on page 101.

DMP can provide basic multipathing to ALUA-compliant arrays even if there is no ASL or JBOD definition. DDL claims the LUNs as part of the aluadisk enclosure. The array type is shown as ALUA. Adding a JBOD definition also enables you to group the LUNs into enclosures.

Disk categories

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

Disks in JBODs which are capable of being multipathed by DMP, are placed in the `DISKS` category. Disks in unsupported arrays can also be placed in the `DISKS` category.

See "Adding unsupported disk arrays to the DISKS category" on page 101.

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

Adding support for a new disk array

You can dynamically add support for a new type of disk array which has been developed by a third-party vendor. The support comes in the form of vendor-supplied libraries.

Add support to a Solaris system by using the `pkgadd` command.

To add 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.

  This is true only if the newly added devices are claimed by the ssd driver. If not, then device discovery must be triggered manually by running the `vxdisk scandisks` or `vxdctl enable` command.

Enabling discovery of new disk arrays

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.

Warning: 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.

To enable discovery of a new disk array

- Type the following command:

```
# vxdctl enable
```

Removing support for a disk array

Remove support for a disk array by removing the Array Support Library (ASL).

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.

To remove support for a disk array

- Type the following command (in this example, the `vrtsda` ASL is being removed):

```
# pkgrm vrtsda
```

Third-party driver coexistence

The third-party driver (TPD) coexistence feature of VxVM 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 108.

See "Displaying information about TPD-controlled devices" on page 184.

Autodiscovery of EMC Symmetrix arrays

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

- With EMC PowerPath installed, EMC Symmetrix arrays could be configured as foreign devices.
  See "Foreign devices" on page 104.
- Without EMC PowerPath installed, DMP could be used to perform multipathing.

On upgrading a system to VxVM 4.1 or later release, existing EMC PowerPath devices can be discovered by DDL, and configured into DMP as autoconfigured disks with DMP nodes, even if PowerPath is being used to perform multipathing. There is no need to configure such arrays as foreign devices.

Table 2-1 shows the scenarios for using DMP with PowerPath.

Table 2-1 Scenarios for using DMP with PowerPath

PowerPath	DMP	Array mode
Installed	The `libvxemc` ASL handles EMC arrays and DGC claiming internally. PowerPath handles failover. There is no need to install the EMC CLARiiON CX ASL or APM.	Explicit failover
Not installed; the array is not CXn00	DMP handles multipathing. The `libvxemc` ASL handles the EMC Symmetrix array.	Any
Not installed; the array is CXn00	DMP handles multipathing. The EMC CLARiiON CX ASL and APM must be installed.	Any

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

```
# vxddladm rmforeign blockpath=/dev/dsk/emcpower10 \
  charpath=/dev/rdsk/emcpower10
```

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

How to administer 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 hierarchy of all the devices discovered by DDL including iSCSI devices.
- List all the Host Bus Adapters including iSCSI
- List the ports configured on a Host Bus Adapter
- List the targets configured from a Host Bus Adapter
- List the devices configured from a Host Bus Adapter
- Get or set the iSCSI operational parameters
- List the types of arrays that are supported.
- Add support for an array to DDL.
- Remove support for an array from DDL.
- List information about excluded disk arrays.
- List disks that are supported in the `DISKS` (JBOD) category.
- Add disks from different vendors to the `DISKS` category.
- Remove disks from the `DISKS` category.
- Add disks as foreign devices.

The following sections explain these tasks in more detail.

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

Listing all the devices including iSCSI

You can display the hierarchy of all the devices discovered by DDL, including iSCSI devices.

To list all the devices including iSCSI

- Type the following command:

```
# vxddladm list
```

The following is a sample output:

```
HBA c2 (20:00:00:E0:8B:19:77:BE)
     Port c2_p0 (50:0A:09:80:85:84:9D:84)
          Target c2_p0_t0 (50:0A:09:81:85:84:9D:84)
               LUN c2t0d0s2
. . .
HBA c3 (iqn.1986-03.com.sun:01:0003ba8ed1b5.45220f80)
     Port c3_p0 (10.216.130.10:3260)
          Target c3_p0_t0 (iqn.1992-08.com.netapp:sn.84188548)
               LUN c3t0d0s2
               LUN c3t0d1s2
          Target c3_t1 (iqn.1992-08.com.netapp:sn.84190939)
. . .
```

Listing all the Host Bus Adapters including iSCSI

You can obtain information about all the devices discovered by DDL including iSCSI devices. This includes the following information:

Driver	Driver controlling the HBA.
Firmware	Firmware version.
Discovery	The discovery method employed for the targets.
State	Whether the device is Online or Offline.
Address	The hardware address.

To list all the Host Bus Adapters including iSCSI

- Type the following command:

  ```
  # vxddladm list hbas
  ```

  You can use this command to obtain all of the HBAs, including iSCSI devices, configured on the system. The following is a sample output:

```
HBA-ID Driver Firmware     Discovery           State  Address
-------------------------------------------------------------------------------------------------
c2     qlc    v.3.3.20 IPX Fabric              Online 20:00:00:E0:8B:19:77:BE
c3     iscsi  -            iSNS(10.216.130.10) Online iqn.1986-03.com.sun:01:0003ba8ed1b5.4522
```

Listing the ports configured on a Host Bus Adapter

You can obtain information about all the ports configured on an HBA. The display includes the following information:

HBA-ID	The parent HBA.
State	Whether the device is Online or Offline.
Address	The hardware address.

To list the ports configured on a Host Bus Adapter

- Type the following command:

  ```
  # vxddladm list ports
  ```

  You can use this command to obtain the ports configured on an HBA. The following is a sample output:

```
PortID   HBA-ID    State      Address
-------------------------------------------------------------------
c2_p0    c2        Online     50:0A:09:80:85:84:9D:84
c3_p0    c3        Online     10.216.130.10:3260
```

Listing the targets configured from a Host Bus Adapter or port

You can obtain information about all the targets configured from a Host Bus Adapter. This includes the following information:

Alias	The alias name, if available.

HBA-ID	Parent HBA or port.
State	Whether the device is Online or Offline.
Address	The hardware address.

To list the targets

- To list all of the targets, use the following command:

  ```
  # vxddladm list targets
  ```

  The following is a sample output:

  ```
  TgtID      Alias  HBA-ID   State      Address
  -----------------------------------------------------------------
  c2_p0_t0   -      c2       Online     50:0A:09:80:85:84:9D:84
  c3_p0_t1   -      c3       Online     iqn.1992-08.com.netapp:sn.84190939
  ```

To list the targets configured from a Host Bus Adapter or port

- You can filter based on a HBA or port, using the following command:

  ```
  # vxddladm list targets [hba=hba_name|port=port_name]
  ```

  For example, to obtain the targets configured from the specified HBA:

  ```
  # vxddladm list targets hba=c2
  ```

  ```
  TgtID      Alias  HBA-ID   State      Address
  -----------------------------------------------------------------
  c2_p0_t0   -      c2       Online     50:0A:09:80:85:84:9D:84
  ```

Listing the devices configured from a Host Bus Adapter and target

You can obtain information about all the devices configured from a Host Bus Adapter. This includes the following information:

Target-ID	The parent target.
State	Whether the device is Online or Offline.
DDL status	Whether the device is claimed by DDL. If claimed, the output also displays the ASL name.

To list the devices configured from a Host Bus Adapter

- To obtain the devices configured, use the following command:

```
# vxddladm list devices

Device    Target-ID    State    DDL status (ASL)
-------------------------------------------------------------
c2t0d2s2  c2_p0_t0     Online   CLAIMED (libvxemc.so)
c3t1d2s2  c3_p0_t1     Online   SKIPPED
c4t1d2s2  c4_p0_t1     Offline  ERROR
c4t1d2s2  c4_p0_t2     Online   EXCLUDED
c4t5d2s2  c4_p0_t5     Offline  MASKED
```

To list the devices configured from a Host Bus Adapter and target

- To obtain the devices configured from a particular HBA and target, use the following command:

```
# vxddladm list devices target=target_name
```

Getting or setting the iSCSI operational parameters

DDL provides an interface to set and display certain parameters that affect the performance of the iSCSI device path. However, the underlying OS framework must support the ability to set these values. The `vxddladm set` command returns an error if the OS support is not available.

Table 2-2 Parameters for iSCSI devices

Parameter	Default value	Minimum value	Maximum value
DataPDUInOrder	yes	no	yes
DataSequenceInOrder	yes	no	yes
DefaultTime2Retain	20	0	3600
DefaultTime2Wait	2	0	3600
ErrorRecoveryLevel	0	0	2
FirstBurstLength	65535	512	16777215
InitialR2T	yes	no	yes
ImmediateData	yes	no	yes

Table 2-2 Parameters for iSCSI devices *(continued)*

Parameter	Default value	Minimum value	Maximum value
MaxBurstLength	262144	512	16777215
MaxConnections	1	1	65535
MaxOutStandingR2T	1	1	65535
MaxRecvDataSegmentLength	8182	512	16777215

To get the iSCSI operational parameters on the initiator for a specific iSCSI target

- Type the following commands:

  ```
  # vxddladm getiscsi target=tgt-id {all | parameter}
  ```

  You can use this command to obtain all the iSCSI operational parameters. The following is a sample output:

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

```
PARAMETER                 CURRENT        DEFAULT         MIN           MAX
--------------------------------------------------------------------------------
DataPDUInOrder            yes            yes             no            yes
DataSequenceInOrder       yes            yes             no            yes
DefaultTime2Retain        20               20             0           3600
DefaultTime2Wait           2                2             0           3600
ErrorRecoveryLevel         0                0             0              2
FirstBurstLength       65535            65535           512       16777215
InitialR2T                yes              yes           no            yes
ImmediateData             yes              yes           no            yes
MaxBurstLength         262144           262144          512       16777215
MaxConnections             1                1             1          65535
MaxOutStandingR2T          1                1             1          65535
MaxRecvDataSegmentLength 8192            8182          512       16777215
```

To set the iSCSI operational parameters on the initiator for a specific iSCSI target

- Type the following command:

  ```
  # vxddladm setiscsi target=tgt-id
     parameter=value
  ```

Displaying details about a supported array library

To display details about a supported array library

- Type the following command:

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

To display details about a supported array library

- Type the following command:

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

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

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

Listing all supported disk arrays

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

To list all supported disk arrays

- Type the following command:

  ```
  # vxddladm listsupport all
  ```

Excluding support for a disk array library

To exclude support for a disk array library

- Type the following command:

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

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

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

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

Re-including support for an excluded disk array library

To re-include support for an excluded disk array library

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

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

  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

- Type the following command:

  ```
  # vxddladm listexclude
  ```

Listing supported disks in the DISKS category

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

- Type the following command:

  ```
  # vxddladm listjbod
  ```

Adding unsupported disk arrays to the DISKS category

Disk arrays should be added as JBOD devices if no ASL is available for the array.

JBODs are assumed to be Active/Active (A/A) unless otherwise specified. If a suitable ASL is not available, an A/A-A, A/P or A/PF array must be claimed as an Active/Passive (A/P) JBOD to prevent path delays and I/O failures. If a JBOD is ALUA-compliant, it is added as an ALUA array.

See "How DMP works" on page 163.

Warning: This procedure 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 to the DISKS category

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

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

where *device_name* is the device name of one of the disks in the array (for example, `/dev/hdisk10`). 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 Stop all applications, such as databases, from accessing VxVM volumes that are configured on the array, and unmount all file systems and checkpoints that are configured on the array.

3 If the array is of type A/A-A, A/P or A/PF, configure it in autotrespass mode.

4 Enter the following command to add a new JBOD category:

```
# vxddladm addjbod vid=vendorid [pid=productid] \
[serialnum=opcode/pagecode/offset/length]
[cabinetnum=opcode/pagecode/offset/length] policy={aa|ap}]
```

where *vendorid* and *productid* are the VID and PID values that you found from the previous step. For example, *vendorid* might be `FUJITSU`, `IBM`, or `SEAGATE`. For Fujitsu devices, you must also specify the number of characters in the serial number as the argument to the `length` argument (for example, 10). If the array is of type A/A-A, A/P or A/PF, you must also specify the `policy=ap` attribute.

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

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

5 Use the `vxdctl enable` command to bring the array under VxVM control.

```
# vxdctl enable
```

See "Enabling discovery of new disk arrays" on page 91.

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

```
# vxddladm listjbod
```

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

```
VID      PID                SerialNum               CabinetNum        Policy
                          (Cmd/PageCode/off/len)  (Cmd/PageCode/off/len)
==============================================================================
SEAGATE  ALL PIDs       18/-1/36/12            18/-1/10/11             Disk
SUN      SESS01         18/-1/36/12            18/-1/12/11             Disk
```

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

```
# vxdmpadm listenclosure
ENCLR_NAME    ENCLR_TYPE    ENCLR_SNO    STATUS
============================================================
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
...
```

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

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

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

For more information, enter the command `vxddladm help addjbod`.

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

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

Removing disks from the DISKS category

To remove disks from the `DISKS` category

- Use the `vxddladm` command with the `rmjbod` keyword. The following example illustrates the command for removing disks supplied by the vendor, Seagate:

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

Foreign devices

DDL may not be able to discover some devices that are controlled by third-party drivers, such as those that provide 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 feature was introduced in VxVM 4.0 to support non-standard devices such as RAM disks, some solid state disks, and pseudo-devices such as EMC PowerPath.

Foreign device support has the following limitations:

- A foreign device is always considered as a disk with a single path. Unlike an autodiscovered disk, it does not have a DMP node.
- It is not supported for shared disk groups in a clustered environment. Only standalone host systems are supported.
- It is not supported for Persistent Group Reservation (PGR) operations.
- It is not under the control of DMP, so enabling of a failed disk cannot be automatic, and DMP administrative commands are not applicable.
- Enclosure information is not available to VxVM. This can reduce the availability of any disk groups that are created using such devices.
- The I/O Fencing and Cluster File System features are not supported for foreign devices.

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

See "Third-party driver coexistence" on page 91.

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 for determining the default disk group.

See "Rules for determining the default disk group" on page 218.

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

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

  Warning: Initialization destroys existing data on disks.

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

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

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

See "Disabling multipathing and making devices invisible to VxVM" on page 170.

Changing the disk-naming scheme

You can either use enclosure-based naming for disks or the operating system's naming scheme.

Note: Devices with very long device names (longer than 31 characters) are represented by enclosure-based names regardless of the naming scheme. If the OS-based names include WWN identifiers, the device name displays with the WWN identifier as long as the device name is less than 31 characters. If any device name is longer than 31 characters, that device name displays with an enclosure name.

To change the disk-naming scheme

- Select `Change the disk naming scheme` from the `vxdiskadm` main menu to change the disk-naming scheme that you want VxVM to use. When prompted, enter `y` to change the naming scheme. This restarts the `vxconfigd` daemon to bring the new disk naming scheme into effect.

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

  ```
  # vxddladm set namingscheme=ebn [persistence={yes|no}] \
  [use_avid=yes|no] [lowercase=yes|no]
  ```

  Use the following command to select operating system-based naming:

  ```
  # vxddladm set namingscheme=osn [persistence={yes|no}] \
  [lowercase=yes|no]
  ```

  The optional `persistence` argument allows you to select whether the names of disk devices that are displayed by VxVM remain unchanged after disk hardware has been reconfigured and the system rebooted. By default, enclosure-based naming is persistent. Operating system-based naming is not persistent by default.

  By default, the names of the enclosure are converted to lowercase, regardless of the case of the name specified by the ASL. The enclosure-based device names are therefore in lower case. Set the `lowercase=no` option to suppress the conversion to lowercase.

  For enclosure-based naming, the `use_avid` option specifies whether the Array Volume ID is used for the index number in the device name. By default, `use_avid=yes`, indicating the devices are named as *enclosure_avid*. If `use_avid` is set to `no`, DMP devices are named as *enclosure_index*. The index number is assigned after the devices are sorted by LUN serial number.

  The change is immediate whichever method you use.

  See "Regenerating persistent device names" on page 108.

Displaying the disk-naming scheme

VxVM disk naming can be operating-system based naming or enclosure-based naming. This command displays whether the VxVM disk naming scheme is currently set. It also displays the attributes for the disk naming scheme, such as whether persistence is enabled.

To display the current disk-naming scheme and its mode of operations, use the following command:

```
# vxddladm get namingscheme
```

See "Disk device naming in VxVM" on page 83.

Regenerating persistent device names

The persistent device naming feature makes the names of disk devices persistent across system reboots. DDL assigns device names acording to the persistent device name database.

If operating system-based naming is selected, each disk name is usually set to the name of one of the paths to the disk. After hardware reconfiguration and a subsequent reboot, the operating system may generate different names for the paths to the disks. Therefore, the persistent device names may no longer correspond to the actual paths. This does not prevent the disks from being used, but the association between the disk name and one of its paths is lost.

Similarly, if enclosure-based naming is selected, the device name depends on the name of the enclosure and an index number. If a hardware configuration changes the order of the LUNs exposed by the array, the persistent device name may not reflect the current index.

To regenerate persistent device names

- To regenerate the persistent names repository, use the following command:

  ```
  # vxddladm [-c] assign names
  ```

  The `-c` option clears all user-specified names and replaces them with autogenerated names.

  If the `-c` option is not specified, existing user-specified names are maintained, but OS-based and enclosure-based names are regenerated.

  The disk names now correspond to the new path names.

Changing device naming for TPD-controlled enclosures

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

To change device naming for TPD-controlled enclosures

- For disk enclosures that are controlled by third-party drivers (TPD) whose coexistence is supported by an appropriate ASL, the default behavior is to assign device names that are based on the TPD-assigned node names. You

can use the `vxdmpadm` command to switch between these names and the device names that are known to the operating system:

```
# vxdmpadm setattr enclosure enclosure 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.

Simple or nopriv disks with enclosure-based naming

If you change from OS-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.

You can use the `vxdarestore` command to handle simple and nopriv disk failures that arise from changing to the enclosure-based naming scheme. You do not need to use this command 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 OS-based naming is in use. Additionally, `vxdarestore` does not handle failures on simple or 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.

See "Removing the error state for simple or nopriv disks in the boot disk group" on page 110.

See "Removing the error state for simple or nopriv disks in non-boot disk groups" on page 111.

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

Removing the error state for simple or 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.

To remove the error state for simple or nopriv disks in the boot disk group

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
   ```

Removing the error state for simple or 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.

To remove the error state for simple or nopriv disks in non-boot disk groups

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
   ```

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

To discover 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 OS-based names. To discover the `hdisk#` names that are associated with a given enclosure-based disk name, use either of the following commands:

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

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

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

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

Disk installation and formatting

Depending on the hardware capabilities of your disks and of your system, you may either need to shut down and power off your system before installing the disks, or you may be able to hot-insert the disks into the live system. Many operating systems can detect the presence of the new disks on being rebooted. If the disks are inserted while the system is live, you may need to enter an operating system-specific command to notify the system.

If the disks require low or intermediate-level formatting before use, use the operating system-specific formatting command to do this.

For Solaris SPARC systems, if a disk is not formatted, the status field in the `vxdisk list` output shows as `nolabel`. The disk must be formatted before you add it to VxVM control.

Note: SCSI disks are usually preformatted. Reformatting is needed only if the existing formatting has become damaged.

See "Displaying or changing default disk layout attributes" on page 113.

See "Adding a disk to VxVM" on page 113.

Displaying or changing default disk layout attributes

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

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

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

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

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

Adding a disk to VxVM

Formatted disks being placed under VxVM control may be new or previously used outside VxVM. The set of disks can consist of all disks on 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 disks may be initialized, while others may be encapsulated to preserve existing data on the disks.

When initializing multiple disks at one time, it is possible to exclude certain disks or certain controllers.

You can also exclude certain disks or certain controllers when encapsulating multiple disks at one time.

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

Warning: Initialization does not preserve the existing data on the disks.

See "Disabling multipathing and making devices invisible to VxVM" on page 170.

To initialize disks for VxVM use

1. Select `Add or initialize one or more disks` from the `vxdiskadm` main menu.
2. At the following prompt, enter the disk device name of the disk to be added to VxVM control (or enter `list` for a list of disks):

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

The <pattern-list> can be a single disk, or a series of disks and/or controllers (with optional targets). If <pattern-list> consists of multiple items, separate them using white space. For example, specify four disks at separate target IDs on controller 3 as follows:

```
c3t0d0 c3t1d0 c3t2d0 c3t3d0
```

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
```

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

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

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

```
Here are the disks selected. Output format: [Device]
list of device names

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

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

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

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

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

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

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

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

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

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

6 At the following prompt, either press Return to accept the default disk name or enter `n` to allow you to define your own disk names:

```
Use default disk names for the disks? [y,n,q,?] (default: y)
```

7 When prompted whether the disks should become hot-relocation spares, enter `n` (or press Return):

```
Add disks as spare disks for disk group name? [y,n,q,?]
(default: n) n
```

8 When prompted whether to exclude the disks from hot-relocation use, enter `n` (or press Return).

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

9 You are next prompted to choose whether you want to add a site tag to the disks:

```
Add site tag to disks? [y,n,q,?] (default: n)
```

A site tag is usually applied to disk arrays or enclosures, and is not required unless you want to use the Remote Mirror feature.

If you enter `y` to choose to add a site tag, you are prompted to the site name at step 11.

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

```
The selected disks will be added to the disk group
disk group name with default disk names.
list of device names
Continue with operation? [y,n,q,?] (default: y) y
```

11 If you chose to tag the disks with a site in step 9, you are now prompted to enter the site name that should be applied to the disks in each enclosure:

```
The following disk(s):
list of device names

belong to enclosure(s):
list of enclosure names

Enter site tag for disks on enclosure enclosure name
[<name>,q,?] site_name
```

12 If you see the following prompt, it lists any disks that have already been initialized for use by VxVM:

```
The following disk devices appear to have been initialized
already.
The disks are currently available as replacement disks.
Output format: [Device]

list of device names

Use these devices? [Y,N,S(elect),q,?] (default: Y) Y
```

This prompt allows you to indicate "yes" or "no" for all of these disks (`Y` or `N`) or to select how to process each of these disks on an individual basis (`S`).

If you are sure that you want to reinitialize all of these disks, enter `Y` at the following prompt:

```
VxVM NOTICE V-5-2-366 The following disks you selected for use
appear to already have been initialized for the Volume
Manager. If you are certain the disks already have been
initialized for the Volume Manager, then you do not need to
reinitialize these disk devices.
Output format: [Device]

list of device names

Reinitialize these devices? [Y,N,S(elect),q,?] (default: Y) Y
```

13 `vxdiskadm` may now indicate that one or more disks is a candidate for encapsulation. Encapsulation allows you to add an active disk to VxVM control and preserve the data on that disk.If you want to preserve the data on the disk, enter `y`. If you are sure that there is no data on the disk that you want to preserve, enter `n` to avoid encapsulation.

```
VxVM NOTICE V-5-2-355 The following disk device has a valid
partition table, but does not appear to have been initialized
for the Volume Manager. If there is data on the disk that
should NOT be destroyed you should encapsulate the existing
disk partitions as volumes instead of adding the disk as a new
disk.
Output format: [Device]

device name

Encapsulate this device? [y,n,q,?] (default: y)
```

14 If you choose to encapsulate the disk `vxdiskadm` confirms its device name and prompts you for permission to proceed. Enter `y` (or press Return) to continue encapsulation:

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

device name

Continue with encapsulation? [y,n,q,?] (default: y) y
vxdiskadm now displays an encapsulation status and informs you
that you must perform a shutdown and reboot as soon as
possible:
VxVM INFO V-5-2-333 The disk device device name will be
encapsulated and added to the disk group disk group name with the
disk name disk name.
```

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

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

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

At the following prompt, `vxdiskadm` asks if you want to use the default private region size of 65536 blocks (32MB). Press Return to confirm that you want to use the default value, or enter a different value. (The maximum value that you can specify is 524288 blocks.)

```
Enter desired private region length [<privlen>,q,?]
(default: 65536)
```

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

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

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

`vxdiskadm` then proceeds to encapsulate the disks. You should now reboot your system at the earliest possible opportunity, for example by running this command:

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

The `/etc/vfstab` file is updated to include the volume devices that are used to mount any encapsulated file systems. You may need to update any other references in backup scripts, databases, or manually created swap devices. The original `/etc/vfstab` file is saved as `/etc/vfstab.prevm`.

15 If you choose not to encapsulate the disk `vxdiskadm` asks if you want to initialize the disk instead. Enter `y` to confirm this:

Instead of encapsulating, initialize? [y,n,q,?] (default: n) **y**`vxdiskadm` now confirms those disks that are being initialized and added to VxVM control with messages similar to the following. In addition, you may be prompted to perform surface analysis.

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

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

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

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

17 At the following prompt, `vxdiskadm` asks if you want to use the default private region size of 65536 blocks (32MB). Press Return to confirm that you want to use the default value, or enter a different value. (The maximum value that you can specify is 524288 blocks.)

`vxdiskadm` then proceeds to add the disks.

```
VxVM INFO V-5-2-88 Adding disk device device name to disk group
disk group name with disk name disk name.
  .
  .
  .
```

18 At the following prompt, indicate whether you want to continue to initialize more disks (`y`) or return to the `vxdiskadm` main menu (`n`):

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

The default layout for disks can be changed.

See "Displaying or changing default disk layout attributes" on page 113.

Disk reinitialization

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

See "Adding a disk to VxVM" on page 113.

Warning: Reinitialization does not preserve data on the disk. If you want to reinitialize the disk, make sure that it does not contain data that should be preserved.

If the disk you want to add has been used before, but not with VxVM, you can encapsulate the disk to preserve its information. If the disk you want to add has previously been under the control of Solaris Volume Manager, you can preserve the data it contains on a VxVM disk by the process of conversion.

See "About migration from Solaris Volume Manager" on page 623.

Using vxdiskadd to put a disk under VxVM control

To use the vxdiskadd command to put a disk under VxVM control.

- Type the following command:

  ```
  # vxdiskadd disk
  ```

  For example, to initialize the second disk on the first controller:

  ```
  # vxdiskadd c0t1d0
  ```

  The `vxdiskadd` command examines your disk to determine whether it has been initialized and also checks for disks that have been added to VxVM, and for other conditions.

  The `vxdiskadd` command also checks for disks that can be encapsulated.

  See "Encapsulating a disk" on page 123.

  If you are adding an uninitialized disk, warning and error messages are displayed on the console by the `vxdiskadd` command. Ignore these messages. These messages should not appear after the disk has been fully initialized; the `vxdiskadd` command displays a success message when the initialization completes.

  The interactive dialog for adding a disk using `vxdiskadd` is similar to that for `vxdiskadm`.

  See "Adding a disk to VxVM" on page 113.

RAM disk support in VxVM

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

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

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

Encapsulating a disk

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

This section describes how to encapsulate a disk for use in VxVM. Encapsulation preserves any existing data on the disk when the disk is placed under VxVM control.

To prevent the encapsulation from failing, make sure that the following conditions apply:

- The disk has two free partitions for the public and private regions.
- The disk has an `s2` slice.
- 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. If the disk being encapsulated is the root disk, and this does not have sufficient free space available, a similar sized portion of the swap partition is used instead.

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

See "Rootability" on page 128.

Use the `format` or `fdisk` commands to obtain a printout of the root disk partition table before you encapsulate a root disk. For more information, see the appropriate manual pages. You may need this information should you subsequently need to recreate the original root disk.

You cannot grow or shrink any volume (`rootvol`, `usrvol`, `varvol`, `optvol`, `swapvol`, and so on) that is associated with an encapsulated root disk. This is because these volumes map to physical partitions on the disk, and these partitions must be contiguous..

Warning: If the root disk is encapsulated and the dump device is covered by the swap volume, it is not safe to use the `savecore -L` operation because this overwrites the swap area. Configure a dedicated dump device on a partition other than the swap area.

To encapsulate a disk for use in VxVM

1 Select `Encapsulate one or more disks` from the `vxdiskadm` main menu.

Your system may use device names that differ from the examples shown here.

At the following prompt, enter the disk device name for the disks to be encapsulated:

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

The <pattern-list> can be a single disk, or a series of disks 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 Select the disk group to which the disk it to be added at the following prompt:

```
You can choose to add this disk to an existing disk group or to
a new disk group. To create a new disk group, select a disk
group name that does not yet exist.

Which disk group [<group>,list,q,?]
```

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
disk group name 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 diskgroup01.
```

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

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

Enter the format that is appropriate for your needs. In most cases, this is the default format, `cdsdisk`. Note that only the `sliced` format is suitable for use with root, boot or swap disks.

8 At the following prompt, `vxdiskadm` asks if you want to use the default private region size of 65536 blocks (32MB). Press Return to confirm that you want to use the default value, or enter a different value. (The maximum value that you can specify is 524288 blocks.)

```
Enter desired private region length [<privlen>,q,?]
(default: 65536)
```

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. You should now reboot your system at the earliest possible opportunity, for example by running this command:

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

The `/etc/vfstab` file is updated to include the volume devices that are used to mount any encapsulated file systems. You may need to update any other references in backup scripts, databases, or manually created swap devices. The original `/etc/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
```

The default layout that is used to encapsulate disks can be changed.

See "Displaying or changing default disk layout attributes" on page 113.

Failure of disk encapsulation

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

```
VxVM ERROR V-5-2-338 The encapsulation operation failed with the
following error:
It is not possible to encapsulate device, for the following
```

```
reason:
<VxVM vxslicer ERROR V-5-1-1108 Unsupported disk layout.>
```

One solution is to configure the disk with the `nopriv` format.

See "Using nopriv disks for encapsulation" on page 127.

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, 32 megabytes) for storing the private region that VxVM uses for disk identification and configuration information. This free space cannot be included in any other partitions.

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

You can encapsulate a disk that does not have space available for the VxVM private region partition by using the `vxdisk` utility. To do this, configure the disk as a `nopriv` device that does not have a private region.

The drawback with using `nopriv` devices is that VxVM cannot track changes in the address or controller of the disk. Normally, VxVM uses identifying information stored in the private region on the physical disk to track changes in the location of a physical disk. Because `nopriv` devices do not have private regions and have no identifying information stored on the physical disk, tracking cannot occur.

One use of `nopriv` devices is to encapsulate a disk so that you can use VxVM to move data off the disk. When space has been made available on the disk, remove the `nopriv` device, and encapsulate the disk as a standard disk device.

A disk group cannot be formed entirely from `nopriv` devices. This is because `nopriv` devices do not provide space for storing disk group configuration information. Configuration information must be stored on at least one disk in the disk group.

Creating a nopriv disk for encapsulation

Warning: Do not use nopriv disks to encapsulate a root disk. If insufficient free space exists on the root disk for the private region, part of the swap area can be used instead.

To create a nopriv disk for encapsulation

1. If it does not exist already, set up a partition on the disk for the area that you want to access using VxVM.
2. Use the following command to map a VM disk to the partition:

   ```
   # vxdisk define partition-device type=nopriv
   ```

   where *partition-device* is the basename of the device in the `/dev/dsk` directory.

   For example, to map partition `3` of disk device `c0t4d0`, use the following command:

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

Creating volumes for other partitions on a nopriv disk

To create volumes for other partitions on a nopriv disk

1. Add the partition to a disk group.
2. Determine where the partition resides within the encapsulated partition.
3. If no data is to be preserved on the partition, use `vxassist` to create a volume with the required length.

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

Rootability

VxVM can place various files from the root file system, `swap` device, and other file systems on the root disk under VxVM control. This is called rootability. The root disk (that is, the disk containing the root file system) can be put under VxVM control through the process of encapsulation.

The root disk can be encapsulated using the `vxdiskadm` command.

See "Encapsulating a disk" on page 123.

Once encapsulated, the root disk can also be mirrored by using the `vxdiskadm`. command.

See "Mirroring an encapsulated root disk" on page 131.

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

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

It is possible to mirror the `rootvol`, and `swapvol` volumes, as well as other parts of the root disk that are required for a successful boot of the system (for example, `/usr`). This provides complete redundancy and recovery capability in the event of disk failure. Without VxVM rootability, the loss of the `root`, `swap`, or `usr` partition prevents the system from being booted from surviving disks.

Mirroring disk drives that are critical to booting ensures that no single disk failure renders the system unusable. A suggested configuration is to mirror the critical disk onto another available disk (using the `vxdiskadm` command). If the disk containing `root` and `swap` partitions fails, the system can be rebooted from a disk containing mirrors of these partitions.

Recovering a system after the failure of an encapsulated root disk requires the application of special procedures.

See the *Veritas Volume Manager Troubleshooting Guide*.

Booting root volumes

When the operating system is booted, the `root` file system and `swap` area must be available for use before the `vxconfigd` daemon can load the VxVM configuration or start any volumes. During system startup, the operating system must see the `rootvol` and `swapvol` volumes as regular partitions so that it can access them as ordinary disk partitions.

Due to this restriction, each of the `rootvol` and `swapvol` plexes must be created from contiguous space on a disk that is mapped to a single partition. It is not possible to stripe, concatenate or span the plex of a `rootvol` or `swapvol` volume that is used for booting. Any mirrors of these plexes that are potentially bootable also cannot be striped, concatenated or spanned.

For information on how to configure your system BIOS to boot from a disk other than the default boot disk, refer to the documentation from your hardware vendor.

Boot-time volume restrictions

When the operating system is booted, the `root` file system and `swap` area must be available for use before the `vxconfigd` daemon can load the VxVM configuration or start any volumes. During system startup, the operating system must see the `rootvol` and `swapvol` volumes as regular partitions so that it can access them as ordinary disk partitions.

Due to this restriction, each of the `rootvol` and `swapvol` plexes must be created from contiguous space on a disk that is mapped to a single partition. It is not possible to stripe, concatenate or span the plex of a `rootvol` or `swapvol` volume that is used for booting. Any mirrors of these plexes that are potentially bootable also cannot be striped, concatenated or spanned.

Volumes on the root disk have the following restrictions on their configuration:

- For the x64 platform, root encapsulation is supported for Update 1 and later releases of the Solaris 10 OS that include the GRUB boat loader.
- The root volume (`rootvol`) must exist in the disk group that is chosen to be the boot disk group, `bootdg`. Although other volumes named `rootvol` can be created in other disk groups, only the `rootvol` in `bootdg` can be used to boot the system.
- If the volumes that are required to boot the system span multiple disks, all these disks must be in the boot disk group.
- The `rootvol` and `swapvol` volumes always have minor device numbers 0 and 1 respectively. Other volumes on the root disk do not have specific minor device numbers.
- Restricted mirrors of volumes on the root disk have overlay partitions created for them. An overlay partition is one that exactly includes the disk space occupied by the restricted mirror. During boot, before the `rootvol`, `varvol`, `usrvol` and `swapvol` volumes are fully configured, the default volume configuration uses the overlay partition to access the data on the disk.
- Although it is possible to add a striped mirror to a `rootvol` device for performance reasons, you cannot stripe the primary plex or any mirrors of `rootvol` that may be needed for system recovery or booting purposes if the primary plex fails.
- `rootvol` and `swapvol` cannot be spanned or contain a primary plex with multiple noncontiguous subdisks. You cannot grow or shrink any volume associated with an encapsulated boot disk (`rootvol`, `usrvol`, `varvol`, `optvol`, `swapvol`, and so on) because these map to a physical underlying partition on the disk and must be contiguous. A workaround is to unencapsulate the boot

disk, repartition the boot disk as desired (growing or shrinking partitions as needed), and then re-encapsulating.

- When mirroring parts of the boot disk, the disk being mirrored to must be large enough to hold the data on the original plex, or mirroring may not work.
- The volumes on the root disk cannot use dirty region logging (DRL).

In addition to these requirements, it is a good idea to have at least one contiguous, (cylinder-aligned if appropriate) mirror for each of the volumes for `root`, `usr`, `var`, `opt` and `swap`. This makes it easier to convert these from volumes back to regular disk partitions (during an operating system upgrade, for example).

Mirroring an encapsulated root disk

VxVM allows you to mirror the root volume and other areas needed for booting onto another disk. Mirroring the root volume enables you to recover from a failure of your root disk by replacing it with one of its mirrors.

For Sun x64 systems, mirroring a root disk creates a GRUB boot menu entry for the Primary and Alternate (mirror) Boot disk.

For Sun SPARC systems, after mirroring the root disk, you can configure the system to boot from the alternate boot drive to recover from a primary boot drive failure.

See the *Veritas Volume Manager Troubleshooting Guide* for more information about recovering from boot drive failure.

To mirror your root disk onto another disk

1. Choose a disk that is at least as large as the existing `root` disk.
2. If the selected disk is not already under VxVM control, use the `vxdiskadd` or `vxdiskadm` command, or the Veritas Enterprise Administrator (VEA) to add it to the `bootdg` disk group. Ensure that you specify the `sliced` format for the disk.

3 Select **Mirror Volumes on a Disk** from the `vxdiskadm` main menu, or use the VEA to create a mirror of the root disk. Doing so automatically invokes the `vxrootmir` command if the mirroring operation is performed on the `root` disk.

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

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

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

4 Monitor the progress of the mirroring operation with the `vxtask list` command.

```
# vxtask list
   TASKID PTID TYPE/STATE PCT PROGRESS
   161 PARENT/R 0.00% 3/0(1) VXRECOVER dg01 dg
   162 162 ATCOPY/R 04.77% 0/41945715/2000896 PLXATT home home-01 dg
```

Booting from alternate boot disks

If the root disk is encapsulated and mirrored, you can use one of its mirrors to boot the system if the primary boot disk fails. This procedure differs between Sun SPARC systems and Sun x64 systems.

Booting from an alternate (mirror) boot disk on Sun x64 systems

On a Sun x64 system, the alternate boot disk is added to the GRUB boot menu when a boot disk is mirrored. If one root disk fails, the system stays up and lets you replace the disk. No reboot is required to perform this maintenance with internal SAS controllers and other CRU-type drives that are hot swappable. Replace the disk, then rescan with the `vxdctl enable` command to discover the replacement.

Alternatively, the `bootpath` can be redefined in the EEPROM without changing the GRUB configuration.

See “The boot process on x64 systems ” on page 133.

Console access and the ability to select from the GRUB menu is required for the following procedure. The system should not have rebooted because of plex failure, but may have rebooted for other reasons.

To boot from an alternate boot disk on an x64 system

1 Select the "Alternate" GRUB menu entry:

```
title Solaris 10 11/06 s10x_u3wos_10 x64  <VxVM: Alternate Boot Disk>
       root (hd0,0,a)
       kernel /platform/i64pc/multiboot
       module /platform/i64pc/boot_archive.alt
```

2 After the system has booted, see the *Veritas Volume Manager Troubleshooting Guide* for information on replacing the failed drive.

The boot process on x64 systems

From Update 1 of the Solaris 10 OS, x64 systems are configured to use the GRUB boot loader. The devices from which a system may be booted are defined in the GRUB configuration file, `/boot/grub/menu.lst`. From the GRUB menu, you can select from the available bootable partitions that are known to the system. By default, the system will boot from the device that is defined by the `bootpath` variable in the EEPROM.

Defining root disk mirrors as bootable

After creating a root disk mirror, you can make it available for booting.

On Sun x64 systems, VxVM automatically creates a GRUB menu entry for the alternate boot disk when the boot disk is mirrored. During the booting process, select the alternate GRUB menu entry from the system console.

An alternate method is to change the 'default' GRUB menu setting in the /boot/grub/menu.lst file to select this entry automatically during the booting process.

Booting from an alternate boot disk on Sun SPARC systems

If the root disk is encapsulated and mirrored, you can use one of its mirrors to boot the system if the primary boot disk fails.

On a Sun SPARC system, booting from an alternate boot disk requires that some EEPROM settings are changed.

See "The boot process on SPARC systems" on page 134.

The boot process on SPARC systems

A Sun SPARC® system prompts for a boot command unless the `autoboot` flag has been set in the nonvolatile storage area used by the firmware. Machines with older PROMs have different prompts than that for the newer V2 and V3 versions. These newer versions of PROM are also known as OpenBoot PROMs (OBP). The `boot` command syntax for the newer types of PROMs is:

```
ok boot [OBP names] [filename] [boot-flags]
```

The OBP names specify the OpenBoot PROM designations. For example, on Desktop SPARC systems, the designation `sbus/esp@0,800000/sd@3,0:a` indicates a SCSI disk (`sd`) at target 3, lun 0 on the SCSI bus, with the `esp` host bus adapter plugged into slot 0.

You can use Veritas Volume Manager boot disk alias names instead of OBP names. Example aliases are `vx-rootdisk` or `vx-disk01`. To list the available boot devices, use the `devalias` command at the OpenBoot prompt.

The filename argument is the name of a file that contains the kernel. The default is `/kernel/unix` in the `root` partition. If necessary, you can specify another program (such as `/stand/diag`) by specifying the `-a` flag. (Some versions of the firmware allow the default filename to be saved in the nonvolatile storage area of the system.)

Warning: Do not boot a system running VxVM with rootability enabled using all the defaults presented by the `-a` flag.

Boot flags are not interpreted by the `boot` program. The `boot` program passes all boot-flags to the file identified by filename.

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

See the `kernel` (1) manual page.

Defining a root disk mirror as bootable

After creating a root disk mirror, you can make it available for booting.

To define a root disk mirror as bootable on a SPARC system

1 Check that the EEPROM variable `use-nvramrc?` is set to `true`. Enter the following command at the boot prompt:

```
ok printenv use-nvramrc?
```

If the system is up and running, enter the following command:

```
# eeprom use-nvramrc?
```

If `use-nvramrc?` is set to `true`, this allows the use of alternate boot disks.

If `use-nvramrc?` is set to `false`, the system fails to boot from the `devalias` and displays an error message such as the following:

```
Rebooting with command: boot vx-mirdisk
Boot device: /pci@1f,4000/scsi@3/disk@0,0 File and args:vx-
mirdisk
boot: cannot open vx-mirdisk
Enter filename [vx-mirdisk]:
```

2 Set the value of `use-nvramrc?` to true.

At the `ok` boot prompt, enter:

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

If the system is up and running, use the following command:

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

3 Define an alternate boot disk by entering the following command at the `ok` boot prompt:

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

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

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

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

4 Use the `devalias` command at the boot prompt to discover the alternate disks from which the system may be booted:

```
ok devalias
```

Suitable mirrors of the root disk are listed with names of the form `vx-diskname`.

5 You should now be able to boot the system from an alternate boot disk, `vx-altboot_disk`, by entering the following command at the `ok` boot prompt:

```
ok boot vx-altboot_disk
```

If a selected disk contains a root mirror that is stale, `vxconfigd` displays an error stating that the mirror is unusable and lists any non-stale alternate bootable disks.

Mirroring other file systems on the root disk

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

```
# vxassist mirror homevol alternate_disk
```

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

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

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

Encapsulating SAN disks

A Solaris system may be booted from a SAN disk under the following conditions:

- For Solaris 9, the operating system must first be installed on an internal boot disk. The `ufsdump` and `ufsrestore` commands can then be used to create a bootable SAN disk. For more information, refer to the Sun document *Automating the Installation of an FC-Fabric SAN-Booted System* at http://www.sun.com/bigadmin/features/articles/fc_fabric_san.html.
- For Solaris 10, the operating system can also be installed directly onto a fabric disk in a SAN environment.

Veritas Volume Manager can encapsulate a bootable SAN disk provided that the disk is listed as being supported for this purpose in the Hardware Compatibility List (HCL) on the support site at http://support.veritas.com.

For some disk arrays, special hardware configuration may be required to allow a system to be booted from one of the LUNs in the array. Refer to the documentation supplied by the array vendor for more information. Having configured the disk array so that you can boot your system from it, you can proceed to encapsulate it using VxVM.

To migrate from an internal boot disk to a SAN boot disk:

1. Verify that the HCL lists the target SAN disk as being supported for SAN booting.
2. Use `Add or initialize one or more disks` from the `vxdiskadm` main menu, or the VEA to add the target SAN disk to the boot disk group (aliased as bootdg).
3. Use `Mirror Volumes on a Disk` from the `vxdiskadm` main menu, or the VEA to create a mirror of the root disk on the target disk.
4. Boot from the mirror disk to verify that the system is still bootable.

   Once you have booted the system from the SAN disk, you can mirror it to another SAN disk that has been added to the boot disk group.

   If required, you can remove the plexes of the original boot disk by using the `vxplex` command. For example, the following command removes the plexes `rootvol-01`, `swapvol-01`, and `home-01` that are configured on the boot disk:

   ```
   # vxplex -o rm dis rootvol-01 swapvol-01 home-01
   ```

Unencapsulating the root disk

You can use the `vxunroot` utility to remove rootability support from a system. This makes `root`, `swap`, `home` and other file systems on the root disk directly accessible through disk partitions, instead of through volume devices.

The `vxunroot` utility also makes the necessary configuration changes to allow the system to boot without any dependency on VxVM.

Only the volumes that were present on the root disk when it was encapsulated can be unencapsulated using `vxunroot`. Before running `vxunroot`, evacuate all other volumes that were created on the root disk after it was encapsulated.

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

Warning: This procedure requires a reboot of the system.

To remove rootability from a system

1. Use the `vxplex` command to remove all the plexes of the volumes `rootvol`, `swapvol`, `usr`, `var`, `opt` and `home` on the disks other than the root disk.

   For example, the following command removes the plexes `rootvol-02`, `swapvol-02`, and `home-02` that are configured on the boot disk mirror:

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

2. Run the `vxunroot` utility:

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

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

Dynamic LUN expansion

Note: A Storage Foundation license is required to use the dynamic LUN expansion feature.

The following form of the `vxdisk` command can be used to make VxVM aware of the new size of a virtual disk device that has been resized:

```
# vxdisk [-f] [-g diskgroup] resize {accessname|medianame} \
  [length=value]
```

The device must have a SCSI interface that is presented by a smart switch, smart array or RAID controller. Following a resize operation to increase the length that is defined for a device, additional disk space on the device is available for allocation. You can optionally specify the new size by using the `length` attribute.

If a disk media name rather than a disk access name is specified, the disk group must either be specified using the `-g` option or the default disk group will be used. If the default disk group has not been set up, an error message will be generated.

This facility is provided to support dynamic LUN expansion by updating disk headers and other VxVM structures to match a new LUN size. It does not resize the LUN itself.

Any volumes on the device should only be grown after the device itself has first been grown. Otherwise, storage other than the device may be used to grow the volumes, or the volume resize may fail if no free storage is available.

Resizing should only be performed on devices that preserve data. Consult the array documentation to verify that data preservation is supported and has been qualified. The operation also requires that only storage at the end of the LUN is affected. Data at the beginning of the LUN must not be altered. No attempt is made to verify the validity of pre-existing data on the LUN. The operation should be performed on the host where the disk group is imported (or on the master node for a cluster-shared disk group).

Resizing of LUNs that are not part of a disk group is not supported. It is not possible to resize LUNs that are in the boot disk group (aliased as `bootdg`), in a deported disk group, or that are offline, uninitialized, being reinitialized, or in an error state.

Warning: Do not perform this operation when replacing a physical disk with a disk of a different size as data is not preserved.

Before reducing the size of a device, any volumes on the device should first be reduced in size or moved off the device. By default, the resize fails if any subdisks would be disabled as a result of their being removed in whole or in part during a shrink operation.

If the device that is being resized has the only valid configuration copy for a disk group, the `-f` option may be specified to forcibly resize the device.

Resizing a device that contains the only valid configuration copy for a disk group can result in data loss if a system crash occurs during the resize.

Resizing a virtual disk device is a non-transactional operation outside the control of VxVM. This means that the resize command may have to be re-issued following a system crash. In addition, a system crash may leave the private region on the device in an unusable state. If this occurs, the disk must be reinitialized, reattached to the disk group, and its data resynchronized or recovered from a backup.

Extended Copy Service

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

The benefits of the Extended Copy Service are:

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

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

```
# vxprint -l disk03
Disk group: rootdg

Disk:        disk03
info:        diskid=1234567890.59.vm250e1.veritas.com
assoc:       device=c2t2d0s2 type=auto
flags:       autoconfig ecopy_enabled
device:      path=/dev/vx/dmp/c2t2d0s4
devinfo:     publen=35354136 privlen=9167
```

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

Enabling a disk for Extended Copy Service operation

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

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

Enabling Extended Copy Service for Hitachi arrays

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

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

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

  This command must be executed as `root`.

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

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

  ```
  c1t22d0      50060e800404040b
  c1t23d0      50060e800404040b
  c1t24d0      50060e800404040b
  ```

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

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

  ```
  c1t22d0      1
  c1t23d0      2
  c1t24d0      1
  ```

Removing disks

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

See "Disabling a disk group" on page 263.

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

See "Destroying a disk group" on page 263.

You can remove a disk from a system and move it to another system if the disk is failing or has failed.

To remove a disk

1. Stop all activity by applications to volumes that are configured on the disk that is to be removed. Unmount file systems and shut down databases that are configured on the volumes.
2. Use the following command to stop the volumes:

   ```
   # vxvol [-g diskgroup] stop vol1 vol2 ...
   ```

3. Move the volumes to other disks or back up the volumes. To move a volume, use `vxdiskadm` to mirror the volume on one or more disks, then remove the original copy of the volume. If the volumes are no longer needed, they can be removed instead of moved.
4. Check that any data on the disk has either been moved to other disks or is no longer needed.
5. Select `Remove a disk` from the `vxdiskadm` main menu.
6. At the following prompt, enter the disk name of the disk to be removed:

   ```
   Enter disk name [<disk>,list,q,?] mydg01
   ```

7. If there are any volumes on the disk, VxVM asks you whether they should be evacuated from the disk. If you wish to keep the volumes, answer `y`. Otherwise, answer `n`.
8. At the following verification prompt, press Return to continue:

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

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

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

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

   You can now remove the disk or leave it on your system as a replacement.
9. At the following prompt, indicate whether you want to remove other disks (`y`) or return to the `vxdiskadm` main menu (`n`):

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

Removing a disk with subdisks

You can remove a disk on which some subdisks are defined. For example, you can consolidate all the volumes onto one disk. If you use the `vxdiskadm` program to remove a disk, you can choose to move volumes off that disk.

Some subdisks are not movable. A subdisk may not be movable for one of the following reasons:

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

If the `vxdiskadm` program cannot move some subdisks, remove some plexes from some disks to free more space before proceeding with the disk removal operation.

See "Removing a volume" on page 359.

See "Taking plexes offline" on page 288.

To remove a disk with subdisks

1. Run the `vxdiskadm` program and select `Remove a disk` from the main menu.

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

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

   home usrvol

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

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

2. Choose `y` to move all subdisks off the disk, if possible.

Removing a disk with no subdisks

To remove a disk that contains no subdisks from its disk group

- Run the `vxdiskadm` program and select `Remove a disk` from the main menu, and respond to the prompts as shown in this example to remove `mydg02`:

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

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

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

  Enter `y` to remove the disk completely from VxVM control. If you do not want to remove the disk completely from VxVM control, enter `n`.

Removing a disk from VxVM control

After removing a disk from a disk group, you can permanently remove it from Veritas Volume Manager control.

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

To remove a disk from VxVM control

- Type the following command:

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

Removing and replacing disks

A replacement disk should have the same disk geometry as the disk that failed. That is, the replacement disk should have the same bytes per sector, sectors per track, tracks per cylinder and sectors per cylinder, same number of cylinders, and the same number of accessible cylinders.

You can use the `prtvtoc` command to obtain disk information.

Note: You may need to run commands that are specific to the operating system or disk array before removing a physical disk.

The removal and replacement of a disk in a Sun StorEdgeTM A5x00 or similar type of array, requires a different procedure.

See "Removing and replacing a disk in a Sun StorEdge A5x00 disk array" on page 151.

If failures are starting to occur on a disk, but the disk has not yet failed completely, you can replace the disk. This involves detaching the failed or failing disk from its disk group, followed by replacing the failed or failing disk with a new one. Replacing the disk can be postponed until a later date if necessary.

If removing a disk causes a volume to be disabled, you can restart the volume so that you can restore its data from a backup.

See the *Veritas Volume Manager Troubleshooting Guide*.

To replace a disk

1. Select `Remove a disk for replacement` from the `vxdiskadm` main menu.
2. At the following prompt, enter the name of the disk to be replaced (or enter `list` for a list of disks):

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

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

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

home src

No data on these volumes will be lost.

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

mkting

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

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

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

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

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

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

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

4 At the following prompt, either select the device name of the replacement disk (from the list provided), press Return to choose the default disk, or enter `none` if you are going to replace the physical disk:

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

Do not choose the old disk drive as a replacement even though it appears in the selection list. If necessary, you can choose to initialize a new disk.

You can enter `none` if you intend to replace the physical disk.

See "Replacing a failed or removed disk" on page 149.

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.
```

6 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
```

Warning: 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.

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 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`.

8 At the following prompt, `vxdiskadm` asks if you want to use the default private region size of 65536 blocks (32 MB). Press Return to confirm that you want to use the default value, or enter a different value. (The maximum value that you can specify is 524288 blocks.)

```
Enter desired private region length [<privlen>,q,?]
(default: 65536)
```

9 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.
```

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

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

It is possible to move hot-relocate subdisks back to a replacement disk.

See "Configuring hot-relocation to use only spare disks" on page 467.

Replacing a failed or removed disk

The following procedure describes how to replace a failed or removed disk.

A different procedure is required to remove and replace a disk in a Sun StorEdge A5x00 or similar type of array.

See "Removing and replacing a disk in a Sun StorEdge A5x00 disk array" on page 151.

To specify a disk that has replaced a failed or removed disk

1 Select `Replace a failed or removed disk` from the `vxdiskadm` main menu.

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

```
Select a removed or failed disk [<disk>,list,q,?] mydg02
```

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

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

Warning: 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 65536 blocks (32 MB). Press Return to confirm that you want to use the default value, or enter a different value. (The maximum value that you can specify is 524288 blocks.)

```
Enter desired private region length [<privlen>,q,?]
(default: 65536)
```

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

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

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

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

Removing and replacing a disk in a Sun StorEdge A5x00 disk array

Note: The following procedure is suitable for use with any array that is administered by using the Solaris `luxadm` command.

To replace a disk in a Sun StorEdge A5x00 disk array

1 Run the `vxdiskadm` command, and select `Remove a disk for replacement` from the main men. Enter `none` when prompted to name a replacement disk.

2 Use the following command to remove the disk from VxVM:

```
# vxdisk rm daname
```

where *daname* is the disk access name of the device (for example, `c1t5d0s2`).

3 Use the Solaris `luxadm` command to obtain the array name and slot number of the disk, and then use these values with `luxadm` to remove the disk:

```
# luxadm disp /dev/rdsk/daname
# luxadm remove_device array_name,slot_number
```

Follow the `luxadm` prompts, and pull out the disk when instructed.

4 Run the following `luxadm` command when you are ready to insert the replacement disk:

```
# luxadm insert_device array_name,slot_number
```

Follow the `luxadm` prompts, and insert the replacement disk when instructed.

5 Run the following command to scan for the new disk and update the system:

```
# vxdiskconfig
```

6 Run the `vxdiskadm` command, select `Replace a failed or removed disk` from the main menu, and follow the instructions.

See "Replacing a failed or removed disk" on page 149.

Enabling a disk

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

To enable a disk

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

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

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

Taking a disk offline

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

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

To take a disk offline

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

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

- Type the following command:

  ```
  # vxedit [-g diskgroup] rename old_diskname new_diskname
  ```

  By default, VxVM names subdisk objects after the VM disk on which they are located. Renaming a VM disk does not automatically rename the subdisks on that disk.

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

  ```
  # vxdisk list
  DEVICE       TYPE            DISK         GROUP        STATUS
  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
  ```

Reserving disks

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

To reserve a disk

- Type the following command:

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

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

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

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

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

To turn off reservation of a disk

- Type the following command:

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

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

Displaying disk information

Before you use a disk, you need to know if it has been initialized and placed under VxVM control. You also need to know if the disk is part of a disk group, because you cannot create volumes on a disk that is not part of a disk group. The `vxdisk list` command displays device names for all recognized disks, the disk names, the disk group names associated with each disk, and the status of each disk.

To display information on all disks that are known to VxVM

- Type the following command:

```
# vxdisk list
```

VxVM returns a display similar to the following:

```
DEVICE       TYPE            DISK         GROUP    STATUS
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
```

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

To display information about an individual disk

- Type the following command:

```
# vxdisk [-v] list diskname
```

The `-v` option causes the command to additionally list all tags and tag values that are defined for the disk. Without this option, no tags are displayed.

Displaying disk information with vxdiskadm

Displaying disk information shows you which disks are initialized, to which disk groups they belong, and the disk status. The `list` command displays device names for all recognized disks, the disk names, the disk group names associated with each disk, and the status of each disk.

To display disk information

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

```
List disk information
Menu: VolumeManager/Disk/ListDisk
```

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

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

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

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

Changing host LUN configurations online

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

Dynamic LUN reconfigurations require Array Configuration commands, operating system commands and Veritas Volume Manager commands. To complete the operations correctly, the commands must be issued in the proper sequence on the host.

The operations can broadly be classified into the following categories:

- Dynamic LUN removal from an existing target ID.
  See "Dynamic LUN removal from an existing target ID" on page 157.
- Dynamic new LUN addition to new target ID.
  See "Dynamic new LUN addition to a new target ID" on page 159.

Dynamic LUN removal from an existing target ID

In this case, a group of LUNs is unmapped from the host HBA ports and an OS device scan is issued. The device scan recognizes the LUNs and adds the LUNs to DMP control. In order for subsequent LUNs to be added seamlessly, perform additional steps to clean up the OS device tree.

The high-level procedure and the VxVM commands are generic. However, the OS commands may vary for Solaris versions. This procedure uses Solaris 10 with Leadville stack as an example.

To perform online LUN removal

1. Identify the LUNs to be removed from the host.

   Use Storage Array Management to identify the IDs for the above LUNs.

2. Evacuate the data from the LUNs from Volume Manager.

3. Remove the LUNs from the disk group.

   ```
   # vxdg -g diskgroup rmdisk da-name
   ```

4. Unmap/Unmask the concerned LUNs from the host using Storage Array Management.

5. Remove the LUNs from `vxdisk list`:

   ```
   # vxdisk rm da-name
   ```

   Note: This step is required. If you do not perform this step, the DMP device tree will show ghost paths.

6. Clean up the Solaris device tree.

   See "Cleaning up the Operating System device tree after removing LUNs - Solaris 10 example" on page 160.

   Note: This step is required. You must clean up the device tree to release the SCSI Target ID for reuse if a new LUN is added to the host later.

7. Perform a device scan with Volume Manager:

   ```
   # vxdctl enable
   ```

   or

   ```
   # vxdisk scandisks
   ```

8. Verify that the removal completed cleanly by answering the following questions:

   - Is the device tree clean?
     Check that the metanodes are removed from the `/dev` directory.
   - Were all the appropriate LUNs removed?
     Use DMP disk reporting tools such as the `vxdisk list` command output to determine whether the LUNs have been cleaned up successfully.

- Is the `vxdisk list` output correct?
  Verify that the `vxdisk list` output shows the correct number of paths.

If the answer to any of these questions is no, return to step 4 and perform the required steps.

If the answer to all of these questions is yes, the LUN remove operation is successful.

Dynamic new LUN addition to a new target ID

In this case, a new group of LUNS is mapped to the host by multiple HBA ports. An OS device scan is issued for the LUNs to be recognized and added to DMP control.

The high-level procedure and the VxVM commands are generic. However, the OS commands may vary for Solaris versions. This procedure uses Solaris 10 with Leadville stack as an example.

To perform online LUN addition

1. Identify the LUNs to be added.

   Use Storage Array Management to identify the IDs for the above LUNs.

2. Map/Mask the concerned LUNs to the new target IDs on multiple host ports.

3. Perform a Solaris device scan. For example:

   ```
   #cfgadm -c configure c2
   #cfgadm -c configure c3
   #devfsadm -Cv
   ```

   Repeat step 1 to step 2 until the required # of LUNs are visible.

4. If required, write an OS Label on the LUNs using the `fmthard` command.

5. Perform a device scan with Volume Manager:

   ```
   # vxdctl enable
   ```

6. Verify that the devices are correctly added by answering the following questions:

   - Do the correct number of LUNs appear in `vxdisk list`?
   - Are there multiple paths for each LUN?

   If the answer to any of these questions is no, you need to clean up the device tree.

   You also need to clean up the device tree if the following message displays:

```
V-5-1-13788 Data Corruption Protection Activated - User Corrective Action Needed
vxvm:vxconfigd: [ID 702911 daemon.notice] V-5-1-13789 LUN serial number 6005076306FFC5280000000000
```

The above message indicates that a new LUN is attempting to reuse the target ID of an older LUN. The device entries have not been cleaned, so the new LUN cannot use the target ID. DMP prevents this operation until the OS device tree is cleaned up.

See "Cleaning up the Operating System device tree after removing LUNs - Solaris 10 example" on page 160.

Then return to step 3.

If the answer to all of these questions is yes, the LUN has been successfully added. You can now add the LUNs to a disk group, create new volumes, or grow existing volumes.

Cleaning up the Operating System device tree after removing LUNs - Solaris 10 example

You must clean up the device tree after removing LUNs. The OS commands may vary for Solaris versions. This procedure uses Solaris 10 with Leadville stack as an example.

Contact Sun Support if any of the steps in this section fail to produce the wanted results.

To clean up the device tree after you remove LUNs

1. The removed devices show up as `drive not available` in the output of the `format` command:

```
413. c3t5006048ACAFE4A7Cd252 <drive not available>
        /pci@1d,700000/SUNW,qlc@1,1/fp@0,0/ssd@w5006048acafe4a7c,fc
```

2. After the LUNs are unmapped using Array management or the command line, Solaris also displays the devices as either `unusable` or `failing`.

```
bash-3.00# cfgadm -al -o show_SCSI_LUN | grep -i unusable
  c2::5006048acafe4a73,256   disk  connected  configured unusable
  c3::5006048acafe4a7c,255   disk  connected  configured unusable

bash-3.00# cfgadm -al -o show_SCSI_LUN | grep -i failing
  c2::5006048acafe4a73,71    disk  connected configured  failing
  c3::5006048acafe4a7c,252   disk  connected configured  failing
```

3 If the removed LUNs show up as `failing`, you need to force a LIP on the HBA. This operation probes the targets again, so that the device shows up as `unusable`. Unless the device shows up as `unusable`, it cannot be removed from the device tree.

```
luxadm -e forcelip /devices/pci@1d,700000/SUNW,qlc@1,1/fp@0,0:devctl
```

4 To remove the device from the `cfgadm` database, run the following commands on the HBA:

```
cfgadm -c unconfigure -o unusable_SCSI_LUN c2::5006048acafe4a73
cfgadm -c unconfigure -o unusable_SCSI_LUN c3::5006048acafe4a7c
```

5 Repeat step 2 to verify that the LUNs have been removed.

6 Clean up the device tree. The following command removes the `/dev/rdsk...` links to `/devices`.

```
$devfsadm -Cv
```

Chapter 3

Administering Dynamic Multipathing

This chapter includes the following topics:

- How DMP works
- Disabling multipathing and making devices invisible to VxVM
- Enabling multipathing and making devices visible to VxVM
- Enabling and disabling I/O for controllers and storage processors
- Displaying DMP database information
- Displaying the paths to a disk
- Setting customized names for DMP nodes
- Administering DMP using vxdmpadm

How DMP works

The Dynamic Multipathing (DMP) feature of Veritas Volume Manager (VxVM) provides greater availability, reliability and performance by using path failover and load balancing. This feature is available for multiported disk arrays from various vendors.

Multiported disk arrays can be connected to host systems through multiple paths. To detect the various paths to a disk, DMP uses a mechanism that is specific to each supported array 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 87.

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

DMP supports the following standard array types:

Active/Active (A/A)	Allows several paths to be used concurrently for I/O. Such arrays allow DMP to provide greater I/O throughput by balancing the I/O load uniformly across the multiple paths to the LUNs. In the event that one path fails, DMP automatically routes I/O over the other available paths.
Asymmetric Active/Active (A/A-A)	A/A-A or Asymmetric Active/Active arrays can be accessed through secondary storage paths with little performance degradation. Usually an A/A-A array behaves like an A/P array rather than an A/A array. However, during failover, an A/A-A array behaves like an A/A array.
Active/Passive (A/P)	Allows access to its LUNs (logical units; real disks or virtual disks created using hardware) via the primary (active) path on a single controller (also known as an access port or a storage processor) during normal operation. In implicit failover mode (or autotrespass mode), an A/P array automatically fails over by scheduling I/O to the secondary (passive) path on a separate controller if the primary path fails. This passive port is not used for I/O until the active port fails. In A/P arrays, path failover can occur for a single LUN if I/O fails on the primary path.
Active/Passive in explicit failover mode or non-autotrespass mode (A/P-F)	The appropriate command must be issued to the array to make the LUNs fail over to the secondary path.
Active/Passive with LUN group failover (A/P-G)	For Active/Passive arrays with LUN group failover (A/PG arrays), a group of LUNs that are connected through a controller is treated as a single failover entity. Unlike A/P arrays, failover occurs at the controller level, and not for individual LUNs. The primary 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.

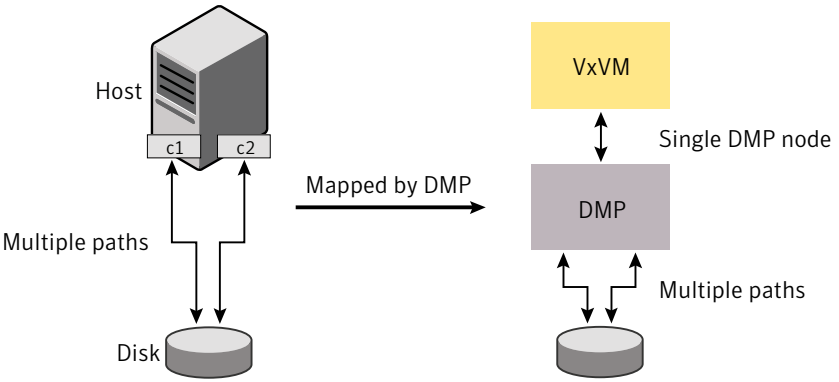
Concurrent Active/Passive (A/P-C) Concurrent Active/Passive in explicit failover mode or non-autotrespass mode (A/PF-C) Concurrent Active/Passive with LUN group failover (A/PG-C)	Variants of the A/P, AP/F and A/PG array types that support concurrent I/O and load balancing by having multiple primary paths 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 array support library (ASL) may define array types to DMP in addition to the standard types for the arrays that it supports.

VxVM uses DMP metanodes (DMP nodes) to access disk devices connected to the system. For each disk in a supported array, DMP maps one node to the set of paths that are connected to the disk. Additionally, DMP associates the appropriate 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.

Figure 3-1 shows how DMP sets up a node for a disk in a supported disk array.

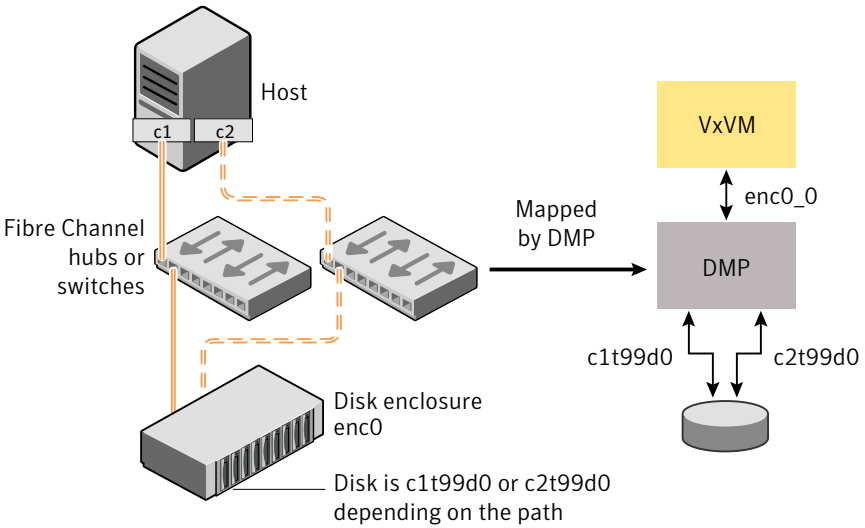
Figure 3-1 How DMP represents multiple physical paths to a disk as one node



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

Figure 3-2 shows an example where 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.

Figure 3-2 Example of multipathing for a disk enclosure in a SAN environment



See "Enclosure-based naming" on page 29.

See "Changing the disk-naming scheme" on page 106.

See "Discovering and configuring newly added disk devices" on page 87.

How DMP monitors I/O on paths

In older releases of VxVM, DMP had one kernel daemon (`errord`) that performed error processing, and another (`restored`) that performed path restoration activities.

From release 5.0, DMP maintains a pool of kernel threads that are used to perform such tasks as error processing, path restoration, statistics collection, and SCSI request callbacks. The `vxdmpadm stat` command can be used to provide information about the threads. The names `errord` and `restored` have been retained for backward compatibility.

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

The restore kernel thread is woken periodically (typically every 5 minutes) to check the health of the paths, and to resume I/O on paths that have been restored. As some paths may suffer from intermittent failure, I/O is only resumed on a path

if has remained healthy for a given period of time (by default, 5 minutes). DMP can be configured with different policies for checking the paths.

See "Configuring DMP path restoration policies" on page 210.

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

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

See "Path failover mechanism" on page 167.

See "I/O throttling" on page 167.

Path failover mechanism

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

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

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

See "Configuring the response to I/O failures" on page 206.

I/O throttling

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

See "Configuring the I/O throttling mechanism" on page 207.

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.

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).

For A/P-C, A/PF-C and A/PG-C arrays, load balancing is performed across all the currently active paths as is done for Active/Active arrays.

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

See "Specifying the I/O policy" on page 197.

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.

See "Enabling and disabling I/O for controllers and storage processors" on page 172.

Using MPxIO with DMP

Either the Sun StorEdge Traffic Manager (SSTM) driver or DMP can be used to handle multipathing for a single host in a non-clustered environment.

In multihost configurations, the Sun StorEdge T3 must be configured to MPxIO mode (`explicit failover` or `nonauto_trespass`). In this mode, the STMS driver on the host handles multipathing. VxVM will not see multiple paths to any device on the array.

You must upgrade the Sun StorEdge T3 firmware to 1.17B. The patch for this upgrade is `109115-09`. The latest firmware for StorEdge T3 and T3+ Arrays is available in patches `109115-xx` and `112276-xx`.

MPxIO mode is enabled by setting `mpxio-disable="no"` in the `/kernel/drv/scsi_vhci.conf` file on Solaris 8 or 9, or in the `/kernel/drv/fp.conf` file on Solaris 10.

Use the command `sys mp_support=mpxio` to enable the MPxIO mode on the Sun StorEdge T3 array.

Note: All hosts in a clustered environment must use the MPxIO mode.

Booting from DMP devices

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

DMP in a clustered environment

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

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

Prior to release 4.1 of VxVM, the clustering and DMP features could not handle automatic failback in A/P arrays when a path was restored, and did not support failback for explicit failover mode arrays. Failback could only be implemented manually by running the `vxdctl enable` command on each cluster node after the path failure had been corrected. From release 4.1, failback is now an automatic cluster-wide operation that is coordinated by the master node. Automatic failback in explicit failover mode arrays is also handled by issuing the appropriate low-level command. If required, this feature can be disabled by selecting the “no failback” option that is defined in the array policy module (APM) for an array.

Note: Support for automatic failback of an A/P array requires that an appropriate ASL (and APM, if required) is available for the array, and has been installed on the system.

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

See "How to administer the Device Discovery Layer" on page 93.

See "Configuring array policy modules" on page 212.

Enabling or disabling controllers with shared disk groups

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

Disabling multipathing and making devices invisible to VxVM

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

Warning: A reboot of the system may be required.

To disable multipathing and make devices invisible to VxVM

1. Run the `vxdiskadm` command, and select `Prevent multipathing/Suppress devices from VxVM's view` from the main menu. You are prompted to confirm whether you want to continue.
2. Select the operation you want to perform from the following options:

Option 1	Excludes 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.
Option 2	Excludes specified paths from the view of VxVM.
Option 3	Excludes disks from the view of VxVM that match a specified Vendor ID and Product ID.
Option 4	Defines 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.
Option 5	Disables multipathing for all disks on a specified controller.
Option 6	Disables multipathing for specified paths. The disks that correspond to a specified path are claimed in the OTHER_DISKS category and are not multipathed.
Option 7	Disables 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.
Option 8	Lists the devices that are currently suppressed or not multipathed.

Enabling multipathing and making devices visible to VxVM

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

Warning: A reboot of the system may be required.

To enable multipathing and make devices visible to VxVM

1. Run the `vxdiskadm` command, and select `Allow multipathing/Unsuppress devices from VxVM's view` from the main menu. You are prompted to confirm whether you want to continue.
2. Select the operation you want to perform from the following options:

Option 1	Makes all paths through a specified controller visible to VxVM.
Option 2	Makes specified paths visible to VxVM.
Option 3	Makes disks visible to VxVM that match a specified Vendor ID and Product ID.
Option 4	Removes a pathgroup definition. (A pathgroup explicitly defines alternate paths to the same disk.) Once a pathgroup has been removed, all paths that were defined in that pathgroup become visible again.
Option 5	Enables multipathing for all disks that have paths through the specified controller.
Option 6	Enables multipathing for specified paths.
Option 7	Enables multipathing for disks that match a specified Vendor ID and Product ID.
Option 8	Lists the devices that are currently suppressed or not multipathed.

Enabling and disabling I/O for controllers and storage processors

DMP allows you to turn off I/O for a controller or the array port of a storage processor so that you can perform administrative operations. This feature can be used for maintenance of HBA controllers on the host, or array ports that are attached to disk arrays supported by VxVM. I/O operations to the controller or array port can be turned back on after the maintenance task is completed. You can accomplish these operations using the `vxdmpadm` command 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 controllers that are connected to the interface board. Disable the 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 the interface card for an array port or an HBA controller card on the host (if supported by the hardware) that is connected to the disk array, disable I/O operations to the array port or to the HBA controller. This shifts all I/O over to an active secondary path or to an active primary path on another 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.

See "Disabling I/O for paths, controllers or array ports" on page 202.

See "Enabling I/O for paths, controllers or array ports" on page 203.

See "Upgrading disk controller firmware" on page 204.

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

Displaying DMP database information

You can use the `vxdmpadm` command to list DMP database information and perform other administrative tasks. This command allows you to list all controllers that are connected to disks, and other related information that is stored in the DMP database. You can use this information to locate system hardware, and to help you decide which controllers need to be enabled or disabled.

The `vxdmpadm` command also provides useful information such as disk array serial numbers, which DMP devices (disks) are connected to the disk array, and which paths are connected to a particular controller, enclosure or array port.

See "Administering DMP using vxdmpadm" on page 176.

Displaying the paths to a 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 one of the system's HBA

controllers. In VxVM, all the physical disks in the system are represented as metadevices with one or more physical paths.

To display the multipathing information on a system

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

```
# vxdisk path

SUBPATH      DANAME     DMNAME     GROUP    STATE
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 one path is active (`state=enabled`) and that the other path has failed (`state=disabled`).

The `type` field is shown for disks on Active/Passive type disk arrays such as the EMC CLARiiON, Hitachi HDS 9200 and 9500, Sun StorEdge 6xxx, and Sun StorEdge T3 array. This field indicates the primary and secondary paths to the disk.

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

Setting customized names for DMP nodes

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

See "Disk device naming in VxVM" on page 83.

You can specify a customized name for a DMP node. User-specified names are persistent even if names persistence is turned off.

To specify a custom name for a DMP node

- Use the following command:

```
# vxdmpadm setattr dmpnode dmpnodename name=name
```

You can also assign names from an input file. This enables you to customize the DMP nodes on the system with meaningful names.

To assign DMP nodes from a file

1. Use the script `vxgetdmpnames` to get a sample file populated from the devices in your configuration. The sample file shows the format required and serves as a template to specify your customized names.
2. To assign the names, use the following command:

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

Administering DMP using vxdmpadm

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

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

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

- Rename an enclosure.
- Configure how DMP responds to I/O request failures.
- Configure the I/O throttling mechanism.
- Control the operation of the DMP path restoration thread.

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

The `vxdmpadm` command can also be used to change the value of various DMP tunables.

See "Changing the values of tunables" on page 555.

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 command displays output similar to the following:

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

Use the `-v` option to display the LUN serial number and the array volume ID.

```
# vxdmpadm -v getdmpnode nodename=c2t16d11s2
```

```
NAME          STATE     PATHS  ENBL  DSBL  ENCLR-NAME    SERIAL-NO        ARRAY_VOL_ID
=============================================================================================
c2t16d11s2    ENABLED  4       4     0     HDS9500-ALUA0 D600172E015B     E01
```

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
```

Use the `dmpnodename`attribute with `getdmpnode` to display the DMP information for a given DMP node.

```
# vxdmpadm getdmpnode dmpnodename=emc_clariion0_158

NAME               STATE    ENCLR-TYPE    PATHS  ENBL  DSBL  ENCLR-NAME
===================================================================
emc_clariion0_158 ENABLED  EMC_CLARiiON  1      1     0     emc_clariion0
```

Displaying consolidated information about the DMP nodes

The `vxdmpadm list dmpnode` command displays the detail information of a DMP node. The information includes the enclosure name, LUN serial number, port id information, device attributes etc.

The following command displays the consolidated information for all of the DMP nodes in the system:

```
# vxdmpadm list dmpnode all
```

The following command displays the consolidated information for all of the DMP nodes in a given enclosure:

```
# vxdmpadm list dmpnode enclosure=enclosure name
```

The following command displays the detailed information for a specified DMP node. The DMP node can be specified by name or by specifying a path name.

```
# vxdmpadm list dmpnode dmpnodename=dmpnodename
```

The command also displays path information for each subpath of the listed dmpnode. The path state differentiates between a path that is disabled due to a failure and a path that has been manually disabled for administrative purposes. A path that has been manually disabled using the `vxdmpadm disable` command is listed as `disabled(m)`.

Use the `enclosure` attribute with `list dmpnode` to obtain a list of all DMP nodes for the specified enclosure.

```
# vxdmpadm list dmpnode enclosure=enc0
```

Use the `dmpnodename`attribute with `list dmpnode` to display the DMP information for a given DMP node.

```
# vxdmpadm list dmpnode dmpnodename=emc_clariion0_158
```

```
dmpdev      = emc_clariion0_19
state       = enabled
enclosure   = emc_clariion0
cab-sno     = APM00042102192
asl         = libvxCLARiiON.so
vid         = DGC
pid         = CLARiiON
array-name  = EMC_CLARiiON
array-type  = CLR-A/P
iopolicy    = MinimumQ
avid        = -
lun-sno     = 6006016070071100F6BF98A778EDD811
udid        = DGC%5FCLARiiON%5FAPM00042102192%5F6006016070071100F6BF98A778EDD811
dev-attr    = -
###path     = name state type transport ctlr hwpath aportID aportWWN attr
path        = hdisk11 enabled(a) primary FC fscsi0 07-08-02 B0APM00042102192 50:06:01:68:10:21
path        = hdisk31 disabled(m) secondary FC fscsi1 08-08-02 A0APM00042102192 50:06:01:60:10
```

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 paths controlled by a DMP node, controller, enclosure, or array port

The `vxdmpadm getsubpaths` command lists all of the paths known to DMP. The `vxdmpadm getsubpaths` command also provides options to list the subpaths through a particular DMP node, controller, enclosure, or array port. To list the paths through an array port, specify either a combination of enclosure name and array port id, or array port WWN.

To list all subpaths known to DMP:

```
# vxdmpadm getsubpaths

NAME   STATE[A] PATH-TYPE[M] DMPNODENAME  ENCLR-NAME     CTLR   ATTRS
=============================================================================
c1t65d0s2 ENABLED(A)  -  Disk_1 Disk  c1   -
c1t66d0s2 ENABLED(A)  -  Disk_2 Disk  c1   -
c2t65d0s2 ENABLED(A)  -  Disk_1 Disk  c2   -
c2t66d0s2 ENABLED(A)  -  Disk_2 Disk  c2   -
c3t2d0s2  ENABLED(A)  -  EMC0_1 EMC0  c3   -
c3t2d1s2  ENABLED(A)  -  EMC0_2 EMC0  c3   -
c4t2d0s2  ENABLED(A)  -  EMC0_1 EMC0  c4   -
c4t2d1s2  ENABLED(A)  -  EMC0_2 EMC0  c4   -
```

The `vxdmpadm getsubpaths` command combined with the `dmpnodename` attribute displays all the paths to a LUN that are controlled by the specified DMP node name from the `/dev/vx/rdmp` directory:

```
# vxdmpadm getsubpaths dmpnodename=c2t66d0s2

NAME    STATE[A]   PATH-TYPE[M]CTLR-NAMEENCLR-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 197.

Paths that are in the DISABLED state are not available for I/O operations.

You can use `getsubpaths` to obtain information about all the paths that are connected to a particular HBA controller:

```
# vxdmpadm getsubpaths ctlr=c2

NAME     STATE[-] PATH-TYPE[-] CTLR-NAME  ENCLR-TYPE ENCLR-NAME ATTRS
=============================================================================
c2t1d0s2 ENABLED  PRIMARY      c2t1d0s2   ACME       enc0       -
c2t2d0s2 ENABLED  PRIMARY      c2t2d0s2   ACME       enc0       -
```

```
c2t3d0s2 ENABLED  SECONDARY    c2t3d0s2   ACME       enc0       -
c2t4d0s2 ENABLED  SECONDARY    c2t4d0s2   ACME       enc0       -
```

You can also use `getsubpaths` to obtain information about all the paths that are connected to a port on an array. The array port can be specified by the name of the enclosure and the array port ID, or by the worldwide name (WWN) identifier of the array port:

```
# vxdmpadm getsubpaths enclosure=HDS9500V0 portid=1A
# vxdmpadm getsubpaths pwwn=20:00:00:E0:8B:06:5F:19
```

For example, to list subpaths through an array port:

```
# vxdmpadm getsubpaths enclosure=HDS9500-ALUA0 portid=1A

NAME  STATE[A] PATH-TYPE[M] DMPNODENAME  ENCLR-NAME CTLR   ATTRS
==========================================================================
c1t65d0s2 ENABLED(A)    PRIMARY   c1t65d0s2    HDS9500-ALUA0 c1      -
c1t66d0s2 ENABLED(A)    PRIMARY   c1t66d0s2    HDS9500-ALUA0 c1      -


-
```

You can use `getsubpaths` to obtain information about all the subpaths of an enclosure.

```
# vxdmpadm getsubpaths enclosure=enclosure_name [ctlr=ctlrname]
```

To list all subpaths of an enclosure:

```
# vxdmpadm getsubpaths enclosure=Disk

NAME      STATE[A]    PATH-TYPE[M] DMPNODENAME  ENCLR-NAME  CTLR   ATTRS
==========================================================================
c1t65d0s2 ENABLED(A)  -            Disk_1       Disk        c1     -
c1t66d0s2 ENABLED(A)  -            Disk_2       Disk        c1     -
c2t65d0s2 ENABLED(A)  -            Disk_1       Disk        c2     -
c2t66d0s2 ENABLED(A)  -            Disk_2       Disk        c2     -
```

To list all subpaths of a controller on an enclosure:

```
# vxdmpadm getsubpaths enclosure=Disk ctlr=c1

NAME      STATE[A]    PATH-TYPE[M] DMPNODENAME  ENCLR-NAME     CTLR   ATTRS
==========================================================================
c1t65d0s2 ENABLED(A)  -            Disk_1       Disk           c1     -
c1t66d0s2 ENABLED(A)  -            Disk_2       Disk           c1     -
```

By default, the output of the `vxdmpadm getsubpaths` command is sorted by enclosure name, DMP node name, and within that, path name. To sort the output based on the pathname, the DMP node name, the enclosure name, or the host controller name, use the `-s` option.

To sort subpaths information, use the following command:

```
# vxdmpadm -s {path | dmpnode | enclosure | ctlr} getsubpaths <clauses>
```

Displaying information about controllers

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

```
# vxdmpadm listctlr all

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

This output shows that the controller `c1` is connected to disks that are not in any recognized DMP category as the enclosure type is `OTHER`.

The other controllers are connected to disks that are in recognized DMP categories.

All the controllers are in the ENABLED state which indicates that they are available for I/O operations.

The state DISABLED is used to indicate that controllers are unavailable for I/O operations. The unavailability can be due to a hardware failure or due to I/O operations being disabled on that controller by using the `vxdmpadm disable` command.

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
```

Displaying information about enclosures

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

```
# vxdmpadm listenclosure enc0
```

```
ENCLR_NAME ENCLR_TYPE  ENCLR_SNO            STATUS  ARRAY_TYPE    LUN_COUNT
==========================================================================
enc0       T3          60020f20000001a90000 CONNECTED  A/P        30
```

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

```
# vxdmpadm listenclosure all
```

```
ENCLR_NAME ENCLR_TYPE  ENCLR_SNO            STATUS  ARRAY_TYPE    LUN_COUNT
==========================================================================
Disk       Disk        DISKS                CONNECTED   Disk       6
SENA0      SENA        508002000001d660     CONNECTED   A/A        57
enc0       T3          60020f20000001a90000 CONNECTED   A/P        30
```

Displaying information about array ports

To display the attributes of an array port that is accessible via a path, DMP node or HBA controller, use one of the following commands:

```
# vxdmpadm getportids path=path-name
# vxdmpadm getportids dmpnodename=dmpnode-name
# vxdmpadm getportids ctlr=ctlr-name
```

The information displayed for an array port includes the name of its enclosure, and its ID and worldwide name (WWN) identifier.

The following form of the command displays information about all of the array ports within the specified enclosure:

```
# vxdmpadm getportids enclosure=enclr-name
```

The following example shows information about the array port that is accessible via DMP node `c2t66d0s2`:

```
# vxdmpadm getportids dmpnodename=c2t66d0s2

NAME       ENCLR-NAME  ARRAY-PORT-ID pWWN
==============================================================
c2t66d0s2  HDS9500V0   1A            20:00:00:E0:8B:06:5F:19
```

Displaying information about TPD-controlled devices

The third-party driver (TPD) coexistence feature allows I/O that is controlled by third-party 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 108.

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
=================================================================
c7t0d10s2emcpower10s2-           emcpower10s2  EMC        EMC0
c6t0d10s2emcpower10s2-           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
```

```
==============================================================
emcpower10s2 ENABLED  2      EMC           EMC0
```

Displaying HBA details

The `vxdmpadm getctlr` command displays HBA vendor details and the Controller ID. For iSCSI devices, the Controller ID is the IQN or IEEE-format based name. For FC devices, the Controller ID is the WWN. Because the WWN is obtained from ESD, this field is blank if ESD is not running. ESD is a daemon process used to notify DDL about occurance of events. The WWN shown as 'Controller ID' maps to the WWN of the HBA port associated with the host controller.

```
# vxdmpadm getctlr c5
```

```
LNAME     PNAME HBA-VENDOR  CTLR-ID
==============================================================================
c5        c5   qlogic 20:07:00:a0:b8:17:e1:37
c6        c6 qlogic iqn.1986-03.com.sun:01:0003ba8ed1b5.45220f80
```

Displaying extended device attributes

The `vxdisk -p list` command displays DDL extended attributes, such as hardware mirrors, in the status field.

The `vxdisk -p list` command displays extended attributes for the following arrays:

- EMC Symmetric
- DS8k
- Hitachi/HDS
- XP12k

For example:

```
# vxdisk -p list
```

```
.............
LUN_SERIAL_NO  : 00AA
LUN_OWNER      : Y
LIBNAME        : libvxhdsalua.sl
HARDWARE_MIRROR: /dev/rdsk/c4t0d7
DDL_DEVICE_ATTR: SVOL
DMP_DEVICE     : c4t0d7
CAB_SERIAL_NO  : D6000180
ATYPE          : ALUA
```

```
ARRAY_VOLUME_ID: 00AA
ARRAY_CTLR_ID  : 1
ANAME          : HDS9500-ALUA
TRANSPORT      : SCSI
…….
```

ASLs furnish this information to DDL through the property DDL_DEVICE_ATTR.

The `vxdisk -x attribute -p list` command displays the 1-line listing for the property list and the attributes. You can specify multiple `-x` options in the same command to display multiple entries. For example:

```
# vxdisk -x DDL_DEVICE_ATTR -x SCSI_VERSION -x VID -p list

DEVICE       SCSI_VERSION VID      DDL_DEVICE_ATTR
disk_1       3            IBM      NULL
c2t0d0       3            Hitachi  STD
c2t1d0       3            Hitachi  STD
c4t0d0       3            Hitachi  STD
c4t0d2       3            Hitachi  STD
c4t0d3       3            Hitachi  STD
c4t0d4       3            Hitachi  STD
c4t0d5       3            Hitachi  STD
c4t0d6       3            Hitachi  STD
c4t0d7       3            Hitachi  STD
c4t1d9       3            Hitachi  STD
c4t1d10      3            Hitachi  TC-PVOL
c4t1d11      3            Hitachi  PVOL
c4t1d12      3            Hitachi  SVOL
```

Also, `vxdisk -e list` prints the EXT_ATTR as the last column.

```
# vxdisk -e list


DEVICE        TYPE    DISK   GROUP  STATUS    OS_NATIVE_NAME  EXT_ATTR
Disk_01       auto    -      -      LVM       hdisk1          -
Disk_02       auto    -      -      LVM       hdisk0          -
Hitachi_03    auto    -      -      online    hdisk73         PVOL
Hitachi_04    auto    -      -      online    hdisk40         SVOL
```

The list of modified ASL's which now discover the following attributes.

a. EMC Symmetric

The following are the list of attributes recognized by EMC symmetric array.

- STD This is the standard device and is not involved in any special operat
- BCV Mirror device created by Timefinder operation
- BCV-NR BCV device in Not ready state.
- MIRROR This is EMC Mirror device.
- SRDF-R1 Primary/source device involved in SRDF operation
- SRDF-R2 Secondary/Target device involved in SRDF operation.

b. DS8k

The following are the list of attributes recognized by DS8000 arrays.

- STD This is the standard device and is not involved in any special opera
- FlashCopy Disk is involved in Point in time copy operation and it is the

c. Hitachi/HDS

The following are the list of attributes recognized by Hitachi arrays

- STD This is the standard HDS device and is not involved in any spec
- PVOL Shadow image primary/original device
- SVOL Shadow image secondary/clone device
- TC-PVOL Truecopy primary/original device
- TC-SVOL Truecopy secondary/clone device
- VVOL Quick shadow device
- HDP Hitachi Dynamic provisioning volume

d. XP12k

- STD This is the standard XP-12K device and is not involved in any sp
- PVOL Primary device involved in cloning
- SVOL Secondary device involved in cloning.
- ThP Thin Provisioning volume

Suppressing or including devices for VxVM or DMP control

The `vxdmpadm exclude` command suppresses devices from VxVM based on the criteria that you specify. The devices can be added back into VxVM control by using the `vxdmpadm include` command. The devices can be included or excluded based on VID:PID combination, paths, controllers, or disks. You can use the bang symbol (!) to exclude or include any paths or controllers except the one specified.

```
# vxdmpadm exclude { all | product=<VID:PID> | ctlr=[!]<ctlr> |
path=[!]<pathname> | dmpnodename=<diskname> [ path=!<pathname>] }
```

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

where:

all – all devices

product=<VID:PID> – all devices with the specified VID:PID

ctlr=<ctlr> – all devices through the given controller

path=<pathname> - the specified device

dmpnodename=<diskname> - all paths under the DMP node

dmpnodename=<diskname> path=!<pathname> - all paths under the DMP node except the one specified.

Gathering and displaying I/O statistics

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

To enable the gathering of statistics, enter this command:

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

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

```
# vxdmpadm iostat reset
```

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

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

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

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

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

To disable the gathering of statistics, enter this command:

```
# vxdmpadm iostat stop
```

Examples of using the vxdmpadm iostat command

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

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

```
# vxdmpadm iostat start
```

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

```
# vxdmpadm iostat show all
                    cpu usage = 7952us    per cpu memory = 8192b
              OPERATIONS              KBYTES              AVG TIME(ms)
PATHNAME    READS     WRITES      READS     WRITES     READS   WRITES
c0t0d0       1088          0     557056          0  0.009542 0.000000
c2t118d0       87          0      44544          0  0.001194 0.000000
c3t118d0        0          0          0          0  0.000000 0.000000
c2t122d0       87          0      44544          0  0.007265 0.000000
c3t122d0        0          0          0          0  0.000000 0.000000
c2t115d0       87          0      44544          0  0.001200 0.000000
c3t115d0        0          0          0          0  0.000000 0.000000
c2t103d0       87          0      44544          0  0.007315 0.000000
c3t103d0        0          0          0          0  0.000000 0.000000
c2t102d0       87          0      44544          0  0.001132 0.000000
c3t102d0        0          0          0          0  0.000000 0.000000
c2t121d0       87          0      44544          0  0.000997 0.000000
c3t121d0        0          0          0          0  0.000000 0.000000
c2t112d0       87          0      44544          0  0.001559 0.000000
c3t112d0        0          0          0          0  0.000000 0.000000
c2t96d0        87          0      44544          0  0.007057 0.000000
c3t96d0         0          0          0          0  0.000000 0.000000
c2t106d0       87          0      44544          0  0.007247 0.000000
c3t106d0        0          0          0          0  0.000000 0.000000
c2t113d0       87          0      44544          0  0.007235 0.000000
c3t113d0        0          0          0          0  0.000000 0.000000
c2t119d0       87          0      44544          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             BYTES             AVG TIME(ms)
PATHNAME     READS     WRITES      READS     WRITES     READS   WRITES
c3t115d0s2       0          0          0          0  0.000000 0.000000

# vxdmpadm iostat show dmpnodename=c0t0d0s2
                    cpu usage = 8501us    per cpu memory = 4096b
               OPERATIONS             BYTES             AVG TIME(ms)
PATHNAME     READS     WRITES      READS     WRITES     READS   WRITES
c0t0d0  s2    1088          0     557056          0  0.009542 0.000000

# vxdmpadm iostat show enclosure=Disk
                    cpu usage = 8626us    per cpu memory = 4096b
               OPERATIONS             BYTES             AVG TIME(ms)
PATHNAME     READS     WRITES      READS     WRITES     READS   WRITES
c0t0d0  s2    1088          0     557056          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             BYTES             AVG TIME(ms)
PATHNAME     READS     WRITES      READS     WRITES     READS   WRITES
c3t115d0s2       0          0          0          0  0.000000 0.000000

                    cpu usage = 59us    per cpu memory = 4096b
               OPERATIONS             BYTES             AVG TIME(ms)
               OPERATIONS             BYTES             AVG TIME(ms)
PATHNAME     READS     WRITES      READS     WRITES     READS   WRITES
c3t115d0s2       0          0          0          0  0.000000 0.000000
```

Displaying statistics for queued or erroneous I/Os

Use the `vxdmpadm iostat show` command with the `-q` option to display the I/Os queued in DMP for a specified DMP node, or for a specified path or controller. For

a DMP node, the -q option displays the I/Os on the specified DMP node that were sent to underlying layers. If a path or controller is specified, the -q option displays I/Os that were sent to the given path or controller and not yet returned to DMP.

See the vxdmpadm(1m) manual page for more information about the vxdmpadm iostat command.

To display queued I/O counts on a DMP node:

```
# vxdmpadm -q iostat show clauses
```

For example:

```
# vxdmpadm -q iostat show dmpnodename=c5t2d1s2

QUEUED I/Os             Pending I/Os
DMPNODENAME  READS  WRITES
c5t2d1s2     2     15           30
```

To display the count of I/Os that returned with errors on a DMP node, path or controller:

```
# vxdmpadm -e iostat show clauses
```

For example, to show the I/O counts that returned errors on a path:

```
# vxdmpadm -e iostat show pathname=c5t2d1s2
```

Displaying cumulative I/O statistics

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

To group by DMP node:

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

For example:

```
# vxdmpadm iostat show groupby=dmpnode dmpnodename=c5t0d1s2

OPERATIONS              BLOCKS             AVG TIME(ms)
 DMPNODENAME    READS      WRITES      READS      WRITES      READS    WRITES
 c5t0d1s2         0           0          0           0        0.000     0.000
```

To group by controller:

```
# vxdmpadm iostat show groupby=ctlr [ all | ctlr=<ctlr> ]
```

For example:

```
# vxdmpadm iostat show groupby=ctlr ctlr=c5

OPERATIONS        BLOCKS          AVG TIME(ms)
 CTLRNAME    READS    WRITES  READS  WRITES     READS   WRITES
 c5         224       14     54       7        4.200  11.100
```

To group by arrayport:

```
vxdmpadm iostat show groupby=arrayport [ all | pwwn=<array port wwn>
 | enclosure=<enclr> portid=<array-port-id> ]
```

For example:

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

OPERATIONS        BLOCKS          AVG TIME(ms)
 PORTNAME    READS    WRITES  READS  WRITES     READS   WRITES
 1A         224       14     54       7        4.200  11.100
```

To group by enclosure:

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

For example:

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

OPERATIONS             BLOCKS            AVG TIME(ms)
 ENCLRNAME         READS     WRITES      READS     WRITES     READS   WRITES
 EMC_CLARiiON 0   0   0     0        0.000     0.000
```

You can also filter out entities for which all data entries are zero. This option is especially useful in a cluster environment which contains many failover devices. You can display only the statistics for the active paths.

To filter all zero entries from the output of the `iostat show` command:

```
vxdmpadm -z iostat show <clauses>
```

For example:

```
# vxdmpadm -z iostat show dmpnodename=c2t16d4s2

           OPERATIONS             BLOCKS            AVG TIME(ms)
PATHNAME     READS     WRITES      READS     WRITES       READS   WRITES
```

```
c3t16d4s2    10        110          2         25        12.000   27.960
c2t17d4s2    20        126          4         29         9.500   19.413
```

You can now specify the units in which the statistics data is displayed. By default, the read/write times are displayed in milliseconds up to 2 decimal places. The throughput data is displayed in terms of 'BLOCKS' and the output is scaled, meaning that the small values are displayed in small units and the larger values are displayed in bigger units, keeping significant digits constant.The `-u` option accepts the following options:

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

For example: To display average read/write times in microseconds.

```
# vxdmpadm -u us iostat show pathname=c2t17d4s2

OPERATIONS              BLOCKS             AVG TIME(microsec)
PATHNAME    READS     WRITES      READS     WRITES       READS    WRITES
c2t17d4s2     20        126          4         29     9500.000   19413.793
```

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 following example 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 optionally assigns a priority number to it. If specified, the priority number must be an integer that is greater than or equal to one. Higher priority numbers indicate that a path is able to carry a greater I/O load.

See "Specifying the I/O policy" on page 197.

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 redundancy level of a device or enclosure

Use the `vxdmpadm getdmpnode` command to list the devices with less than the required redundancy level.

To list the devices with fewer than a given number of active paths, use the following command:

```
# vxdmpadm getdmpnode redundancy=value
```

For example, to list the devices with fewer than 3 active paths, use the following command:

```
# vxdmpadm getdmpnode redundancy=3

NAME STATE PATHS  ENBL  DSBL  ENCLR-NAME
================================================================================
c5t0d1 ENABLED   2      2     0     HDS9500-ALUA0
c5t1d0 ENABLED   4      2     2     HDS9500-ALUA1
c6t0d5 ENABLED   1      1     0     EMC_CLARiiON0
```

You can list the devices on a specified enclosure with the specified redundancy level, as in the following example:

```
# vxdmpadm getdmpnode enclosure=EMC_CLARiiON0 redundancy=3

NAME STATE PATHS  ENBL  DSBL
================================================
c6t0d5 ENABLED   1      1
```

Use the keyword `mrl` to list all devices with less than the number specified as the minimum redundancy level for that device, as follows:

```
# vxdmpadm getdmpnode redundancy=mrl
```

To display the minimum redundancy level for a particular device, use the `vxdmpadm getattr` command, as follows:

```
# vxdmpadm getattr enclosure|arrayname|arraytype component-name redundancy
```

For example, to show the minimum redundancy level for the enclosure HDS9500-ALUA0:

```
# vxdmpadm getattr enclosure HDS9500-ALUA0 redundancy

ENCLR_NAME  DEFAULT  CURRENT
============================================
 HDS9500-ALUA0    0      4
```

Specifying the minimum number of active paths

You can set the minimum redundancy level for a device or an enclosure. The minimum redundancy level is the minimum number of paths that should be active for the device or the enclosure. If the number of paths falls below the minimum redundancy level for the enclosure, a message is sent to the system console and also logged to the DMP log file. Also, notification is sent to `vxnotify` clients.

The value set for minimum redundancy level is stored in the `dmppolicy.info` file, and is persistent. If no minimum redundancy level is set, the default value is 0.

You can use the `vxdmpadm setattr` command to set the minimum redundancy level.

To specify the minimum number of active paths

- Use the `vxdmpadm setattr` command with the redundancy attribute as follows:

  ```
  # vxdmpadm setattr enclosure|arrayname|arraytype component-name
   redundancy=value
  ```

  where *value* is the number of active paths.

  For example, to set the minimum redundancy level for the enclosure HDS9500-ALUA0:

  ```
  # vxdmpadm setattr enclosure HDS9500-ALUA0 redundancy=2
  ```

Displaying the I/O policy

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

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

```
# vxdmpadm getattr enclosure Disk iopolicy

ENCLR_NAME        DEFAULT         CURRENT
------------------------------------------
Disk              MinimumQ        Balanced
```

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

```
# vxdmpadm getattr enclosure enc0 partitionsize
ENCLR_NAME      DEFAULT         CURRENT
```

```
----------------------------------------
enc0            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 (such as `A/A` for Active/Active, or `A/P` for Active/Passive).

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

Do not edit this file yourself.

The following policies may be set:

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

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

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

`balanced [partitionsize=size]`

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

You can use the size argument to the partitionsize attribute to specify the partition size. The partition size in blocks is adjustable in powers of 2 from 2 up to 231.

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 the `dmp_pathswitch_blks_shift` tunable parameter.

See "DMP tunable parameters " on page 565.

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

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

`minimumq`

This policy sends I/O on paths that have the minimum number of outstanding I/O requests in the queue for a LUN. No further configuration is possible as DMP automatically determines the path with the shortest queue.

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

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

This is the default I/O policy for all arrays.

`priority` This policy is useful when the paths in a SAN have unequal performance, and you want to enforce load balancing manually. You can assign priorities to each path based on your knowledge of the configuration and performance characteristics of the available paths, and of other aspects of your system.

See "Setting the attributes of the paths to an enclosure" on page 193.

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

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

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

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

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

`singleactive` This policy routes I/O down the single active path. This policy can be configured for A/P arrays with one active path per controller, where the other paths are used in case of failover. If configured for A/A arrays, there is no load balancing across the paths, and the alternate paths are only used to provide high availability (HA). If the 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
```

Scheduling I/O on the paths of an Asymmetric Active/Active array

You can specify the `use_all_paths` attribute in conjunction with the `adaptive`, `balanced`, `minimumq`, `priority` and `round-robin` I/O policies to specify whether I/O requests are to be scheduled on the secondary paths in addition to the primary paths of an Asymmetric Active/Active (A/A-A) array. Depending on the

characteristics of the array, the consequent improved load balancing can increase the total I/O throughput. However, this feature should only be enabled if recommended by the array vendor. It has no effect for array types other than A/A-A.

For example, the following command sets the `balanced` I/O policy with a partition size of 4096 blocks (2MB) on the enclosure `enc0`, and allows scheduling of I/O requests on the secondary paths:

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

The default setting for this attribute is `use_all_paths=no`.

You can display the current setting for `use_all_paths` for an enclosure, arrayname or arraytype. To do this, specify the `use_all_paths` option to the `vxdmpadm gettattr` command.

```
# vxdmpadm getattr enclosure HDS9500-ALUA0 use_all_paths

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

The `use_all_paths` attribute only applies to A/A-A arrays. For other arrays, the above command displays the message:

```
Attribute is not applicable for this array.
```

Example of applying load balancing in a SAN

This example describes how to configure load balancing in a SAN environment where there are multiple primary paths to an Active/Passive device through several SAN switches. As can be seen in this sample output from the `vxdisk list` command, the device `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              Round-Robin     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          Round-Robin         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 I/O for paths, controllers or array ports

Disabling I/O through a path, HBA controller or array port prevents DMP from issuing I/O requests through the specified path, or the paths that are connected

to the specified controller or array port. The command blocks until all pending I/O requests issued through the paths are completed.

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

Before detaching a system board, stop all I/O to the HBA controllers that are located on the board. To do this, execute the `vxdmpadm disable` command, and then run the Dynamic Reconfiguration (DR) facility provided by Sun.

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

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

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

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

To disable I/O for the paths connected to an array port, use one of the following commands:

```
# vxdmpadm [-c|-f] disable enclosure=enclr_name portid=array_port_ID
# vxdmpadm [-c|-f] disable pwwn=array_port_WWN
```

where the array port is specified either by the enclosure name and the array port ID, or by the array port's worldwide name (WWN) identifier.

The following are examples of using the command to disable I/O on an array port:

```
# vxdmpadm disable enclosure=HDS9500V0 portid=1A
# vxdmpadm disable pwwn=20:00:00:E0:8B:06:5F:19
```

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

The `disable` operation fails if it is issued to a controller that is connected to the root disk through a single path, and there are no root disk mirrors configured on alternate paths. If such mirrors exist, the command succeeds.

Enabling I/O for paths, controllers or array ports

Enabling a controller allows a previously disabled path, HBA controller or array port to accept I/O again. This operation succeeds only if the path, controller or array port is accessible to the host, and I/O can be performed on it. When

connecting Active/Passive disk arrays, the `enable` operation results in failback of I/O to the primary path. The `enable` operation can also be used to allow I/O to the controllers on a system board that was previously detached.

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

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

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

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

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

To enable I/O for the paths connected to an array port, use one of the following commands:

```
# vxdmpadm enable enclosure=enclr_name portid=array_port_ID
# vxdmpadm [-f] enable pwwn=array_port_WWN
```

where the array port is specified either by the enclosure name and the array port ID, or by the array port's worldwide name (WWN) identifier.

The following are examples of using the command to enable I/O on an array port:

```
# vxdmpadm enable enclosure=HDS9500V0 portid=1A
# vxdmpadm enable pwwn=20:00:00:E0:8B:06:5F:19
```

Upgrading disk controller firmware

You can upgrade disk controller firmware without performing a system reboot or unloading the VxVM drivers.

First obtain the appropriate firmware upgrades from your disk drive vendor. You can usually download the appropriate files and documentation from the vendor's support website.

To upgrade the disk controller firmware

1. Disable the plex that is associated with the disk device:

   ```
   # /opt/VRTS/bin/vxplex -g diskgroup det plex
   ```

   (The example is a volume mirrored across 2 controllers on one HBA.)

2. Stop I/O to all disks through one controller of the HBA:

   ```
   # /opt/VRTS/bin/vxdmpadm disable ctlr=first_cntlr
   ```

   For the other controller on the HBA, enter:

   ```
   # /opt/VRTS/bin/vxdmpadm -f disable ctlr=second_cntlr
   ```

3. Upgrade the firmware on those disks for which the controllers have been disabled using the procedures that you obtained from the disk drive vendor.

4. After doing the upgrade, re-enable all the controllers:

   ```
   # /opt/VRTS/bin/vxdmpadm enable ctlr=first_cntlr
   # /opt/VRTS/bin/vxdmpadm enable ctlr=second_cntlr
   ```

5. Re-enable the plex associated with the device:

   ```
   # /opt/VRTS/bin/vxplex -g diskgroup att volume plex
   ```

   This command takes some time depending upon the size of the mirror set.

Renaming an enclosure

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

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

This example changes the name of an enclosure from `enc0` to `GRP1`.

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

The following command shows the changed name:

```
# vxdmpadm listenclosure all

ENCLR_NAME    ENCLR_TYPE    ENCLR_SNO                STATUS
```

```
============================================================
other0          OTHER         OTHER_DISKS               CONNECTED
jbod0           X1            X1_DISKS                  CONNECTED
GRP1            ACME          60020f20000001a90000      CONNECTED
```

Configuring the response to I/O failures

You can configure how DMP responds to failed I/O requests on the paths to a specified enclosure, disk array name, or type of array. By default, DMP is configured to retry a failed I/O request up to 5 times for a single path.

To display the current settings for handling I/O request failures that are applied to the paths to an enclosure, array name or array type, use the `vxdmpadm getattr` command.

See "Displaying recovery option values" on page 209.

To set a limit for the number of times that DMP attempts to retry sending an I/O request on a path, use the following command:

```
# vxdmpadm setattr \
  {enclosure enc-name|arrayname name|arraytype type} \
  recoveryoption=fixedretry retrycount=n
```

The value of the argument to `retrycount` specifies the number of retries to be attempted before DMP reschedules the I/O request on another available path, or fails the request altogether.

As an alternative to specifying a fixed number of retries, the following version of the command specifies how long DMP should allow an I/O request to be retried on a path:

```
# vxdmpadm setattr \
  {enclosure enc-name|arrayname name|arraytype type} \
  recoveryoption=timebound iotimeout=seconds
```

The value of the argument to `iotimeout` specifies the time in seconds that DMP waits for an outstanding I/O request to succeed before it reschedules the request on another available path, or fails the I/O request altogether. The effective number of retries is the value of `iotimeout` divided by the sum of the times taken for each retry attempt. DMP abandons retrying to send the I/O request before the specified time limit has expired if it predicts that the next retry will take the total elapsed time over this limit.

The default value of `iotimeout` is 10 seconds. For some applications, such as Oracle, it may be desirable to set `iotimeout` to a larger value, such as 60 seconds.

Note: The `fixedretry` and `timebound` settings are mutually exclusive.

The following example configures time-bound recovery for the enclosure `enc0`, and sets the value of `iotimeout` to 60 seconds:

```
# vxdmpadm setattr enclosure enc0 recoveryoption=timebound \
  iotimeout=60
```

The next example sets a fixed-retry limit of 10 for the paths to all Active/Active arrays:

```
# vxdmpadm setattr arraytype A/A recoveryoption=fixedretry \
  retrycount=10
```

Specifying `recoveryoption=default` resets DMP to the default settings corresponding to `recoveryoption=fixedretry retrycount=5`, for example:

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

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

See "Configuring the I/O throttling mechanism" on page 207.

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

Configuring the I/O throttling mechanism

By default, DMP is configured with I/O throttling turned off for all paths. To display the current settings for I/O throttling that are applied to the paths to an enclosure, array name or array type, use the `vxdmpadm getattr` command.

See "Displaying recovery option values" on page 209.

If enabled, I/O throttling imposes a small overhead on CPU and memory usage because of the activity of the statistics-gathering daemon. If I/O throttling is disabled, the daemon no longer collects statistics, and remains inactive until I/O throttling is re-enabled.

To turn off I/O throttling, use the following form of the `vxdmpadm setattr` command:

```
# vxdmpadm setattr \
  {enclosure enc-name|arrayname name|arraytype type} \
  recoveryoption=nothrottle
```

The following example shows how to disable I/O throttling for the paths to the enclosure `enc0`:

```
# vxdmpadm setattr enclosure enc0 recoveryoption=nothrottle
```

The `vxdmpadm setattr` command can be used to enable I/O throttling on the paths to a specified enclosure, disk array name, or type of array:

```
# vxdmpadm setattr \
  {enclosure enc-name|arrayname name|arraytype type}\
  recoveryoption=throttle {iotimeout=seconds|queuedepth=n}
```

If the `iotimeout` attribute is specified, its argument specifies the time in seconds that DMP waits for an outstanding I/O request to succeed before invoking I/O throttling on the path. The default value of `iotimeout` is 10 seconds. Setting `iotimeout` to a larger value potentially causes more I/O requests to become queued up in the SCSI driver before I/O throttling is invoked.

If the `queuedepth` attribute is specified, its argument specifies the number of I/O requests that can be outstanding on a path before DMP invokes I/O throttling. The default value of `queuedepth` is 20. Setting `queuedepth` to a larger value allows more I/O requests to become queued up in the SCSI driver before I/O throttling is invoked.

Note: The `iotimeout` and `queuedepth` attributes are mutually exclusive.

The following example sets the value of `iotimeout` to 60 seconds for the enclosure `enc0`:

```
# vxdmpadm setattr enclosure enc0 recoveryoption=throttle \
  iotimeout=60
```

The next example sets the value of `queuedepth` to 30 for the paths to all Active/Active arrays:

```
# vxdmpadm setattr arraytype A/A recoveryoption=throttle \
  queuedepth=30
```

Specify `recoveryoption=default` to reset I/O throttling to the default settings, as follows:

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

The above command configures the default behavior, corresponding to `recoveryoption=nothrottle`. The above command also configures the default behavior for the response to I/O failures.

See "Configuring the response to I/O failures" on page 206.

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

Displaying recovery option values

To display the current settings for handling I/O request failures that are applied to the paths to an enclosure, array name or array type, use the following command:

```
# vxdmpadm getattr \
  {enclosure enc-name|arrayname name|arraytype type} \
  recoveryoption
```

The following example shows the `vxdmpadm getattr` command being used to display the `recoveryoption` option values that are set on an enclosure.

```
# vxdmpadm getattr enclosure HDS9500-ALUA0 recoveryoption
ENCLR-NAME      RECOVERY-OPTION  DEFAULT[VAL]    CURRENT[VAL]
==============================================================
HDS9500-ALUA0   Throttle         Nothrottle[0]   Queuedepth[60]
HDS9500-ALUA0   Error-Retry      Fixed-Retry[5]  Timebound[20]
```

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

Table 3-1 summarizes the possible recovery option settings for retrying I/O after an error.

Table 3-1 Recovery options for retrying I/O after an error

Recovery option	Possible settings	Description
recoveryoption=fixedretry	Fixed-Retry (retrycount)	DMP retries a failed I/O request for the specified number of times if I/O fails.
recoveryoption=timebound	Timebound (iotimeout)	DMP retries a failed I/O request for the specified time in seconds if I/O fails.

Table 3-2 summarizes the possible recovery option settings for throttling I/O.

Table 3-2 Recovery options for I/O throttling

Recovery option	Possible settings	Description
recoveryoption=nothrottle	None	I/O throttling is not used.
recoveryoption=throttle	Queuedepth (queuedepth)	DMP throttles the path if the specified number of queued I/O requests is exceeded.
recoveryoption=throttle	Timebound (iotimeout)	DMP throttles the path if an I/O request does not return within the specified time in seconds.

Configuring DMP path restoration policies

DMP maintains a kernel thread that re-examines the condition of paths at a specified interval. The type of analysis that is performed on the paths depends on the checking policy that is configured.

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

Use the `vxdmpadm start restore` command to configure one of the following restore policies:

- `check_all`

  The path restoration thread analyzes all paths in the system and revives the paths that are back online, as well as disabling the paths that are inaccessible. The command to configure this policy is:

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

- `check_alternate`

  The path restoration thread checks that at least one alternate path is healthy. It generates a notification if this condition is not met. This policy avoids inquiry commands on all healthy paths, and is less costly than `check_all` in cases where a large number of paths are available. This policy is the same as `check_all` if there are only two paths per DMP node. The command to configure this policy is:

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

- check_disabled

  This is the default path restoration policy. The path restoration thread checks the condition of paths that were previously disabled due to hardware failures, and revives them if they are back online. The command to configure this policy is:

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

- check_periodic

  The path restoration thread performs check_all once in a given number of cycles, and check_disabled in the remainder of the cycles. This policy may lead to periodic slowing down (due to check_all) if there is a large number of paths available. The command to configure this policy is:

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

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

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

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

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

See "DMP tunable parameters " on page 565.

You can also stop the path restoration thread, and then restart it with new attributes.

See "Stopping the DMP path restoration thread" on page 212.

If the vxdmpadm start restore command is given without specifying a policy or interval, the path restoration thread is started with the persistent policy and interval settings previously set by the administrator with the vxdmpadm settune command. If the administrator has not set a policy or interval, the system defaults are used. The system default restore policy is check_disabled. The system default interval is 300 seconds.

Warning: Decreasing the interval below the system default can adversely affect system performance.

Stopping the DMP path restoration thread

Use the following command to stop the DMP path restoration thread:

```
# vxdmpadm stop restore
```

Warning: Automatic path failback stops if the path restoration thread is stopped.

Displaying the status of the DMP path restoration thread

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

```
# vxdmpadm stat restored
```

This produces output such as the following:

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

Displaying information about the DMP error-handling thread

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

```
# vxdmpadm stat errord
```

One daemon should be shown as running.

Configuring array policy modules

An array policy module (APM) is a dynamically loadable kernel module 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 loaded and in use. To see detailed information for an individual module, specify the module name as the argument to the command:

```
# vxdmpadm listapm module_name
```

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

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

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

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

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

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

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

Chapter 4

Creating and administering disk groups

This chapter includes the following topics:

- About disk groups
- Displaying disk group information
- Creating a disk group
- Adding a disk to a disk group
- Removing a disk from a disk group
- Deporting a disk group
- Importing a disk group
- Handling cloned disks with duplicated identifiers
- Renaming a disk group
- Moving disks between disk groups
- Moving disk groups between systems
- Handling conflicting configuration copies
- Reorganizing the contents of disk groups
- Disabling a disk group
- Destroying a disk group
- Upgrading a disk group

- Managing the configuration daemon in VxVM
- Backing up and restoring disk group configuration data
- Using vxnotify to monitor configuration changes

About disk groups

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

In releases of Veritas Volume Manager (VxVM) prior to 4.0, a system installed with VxVM was configured with a default disk group, `rootdg`, that had to contain at least one disk. By default, operations were directed to the `rootdg` disk group. From release 4.0 onward, VxVM can function without any disk group having been configured. Only when the first disk is placed under VxVM control must a disk group be configured. There is no longer a requirement that you name any disk group `rootdg`, and any disk group that is named `rootdg` has no special properties because of this name.

See "Specification of disk groups to commands" on page 217.

Note: Most VxVM commands require superuser or equivalent privileges.

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

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

See the *Veritas Storage Foundation Cross-Platform Data Sharing Administrator's Guide*.

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 approximately 256 bytes, 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.

See "Displaying disk group information" on page 220.

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

See "Reorganizing the contents of disk groups" on page 250.

See "Backing up and restoring disk group configuration data" on page 269.

Specification of disk groups to commands

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

System-wide reserved disk groups

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

bootdg	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).
defaultdg	Specifies the default disk group. This is an alias for the disk group name that should be assumed if the -g option is not specified to a command, or if the VXVM_DEFAULTDG environment variable is undefined. By default, defaultdg is set to nodg (no disk group).
nodg	Specifies to an operation that no disk group has been defined. For example, if the root disk is not under VxVM control, bootdg is set to nodg.

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

If you have upgraded your system, you may find it convenient to continue to configure a disk group named rootdg as the default disk group (defaultdg). There is no requirement that both defaultdg and bootdg refer to the same disk group, nor that either the default disk group or the boot disk group be named rootdg.

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:

- Use the default disk group name that is specified by the environment variable VXVM_DEFAULTDG. This variable can also be set to one of the reserved system-wide disk group names: bootdg, defaultdg, or nodg. If the variable is undefined, the following rule is applied.
- Use the disk group that has been assigned to the system-wide default disk group alias, defaultdg. If this alias is undefined, the following rule is applied.
  See "Displaying and specifying the system-wide default disk group" on page 219.
- If the operation can be performed without requiring a disk group name (for example, an edit operation on disk access records), do so.

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

Warning: In releases of VxVM prior to 4.0, a subset of commands attempted to deduce the disk group by searching for the object name that was being operated upon by a command. This functionality is no longer supported. Scripts that rely on deducing the disk group from an object name may fail.

Displaying the system-wide boot disk group

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

```
# vxdg bootdg
```

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

Displaying and specifying the system-wide default disk group

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

```
#  vxdg defaultdg
```

If a default disk group has not been defined, `nodg` is displayed. 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.

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

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

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

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: 140
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
```

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:   simple
flags:  online ready private autoconfig autoimport imported
```

```
diskid: 963504891.1070.bass
dgname: newdg
dgid:   963504895.1075.bass
hostid: bass
info:   privoffset=128
```

Displaying free space in a disk group

Before you add volumes and file systems to your system, make sure you have enough free disk space to meet your needs.

To display free space in the system, use the following command:

```
# vxdg free
```

The following is example output:

```
GROUP DISK      DEVICE    TAG       OFFSET  LENGTH   FLAGS
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 select `Add or initialize one or more disks` from the main menu of the `vxdiskadm` command to add disks to VxVM control. The disks to be added to a disk group must not belong to an existing disk group.

You can 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 similar to that described for the `vxdiskadm` command.

See "Adding a disk to VxVM" on page 113.

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, `c1t0d0s2`, 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, select `Add or initialize one or more disks` from the main menu of the `vxdiskadm` command.

You can also use the `vxdiskadd` command to add a disk to a disk group, 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 similar to that described for the `vxdiskadm` command.

See "Adding a disk to VxVM" on page 113.

Removing a disk from a disk group

Before you can remove the last disk from a disk group, you must disable the disk group.

See "Disabling a disk group" on page 263.

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

See "Destroying a disk group" on page 263.

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.

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

Warning: 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 the disk `c1t0d0s2` from VxVM control, use this command:

```
# vxdiskunsetup c1t0d0s2
```

You can remove a disk on which some subdisks of volumes are defined. For example, you can consolidate all the volumes onto one disk. If you use `vxdiskadm` to remove a disk, you can choose to move volumes off that disk. To do this, run `vxdiskadm` and select `Remove a disk` from the main menu.

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

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

home usrvol

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

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

If you choose `y`, 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

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.

   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 `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`):

   ```
   Enter name of disk group [<group>,list,q,?] (default: list)
   newdg
   ```

5. At the following prompt, enter `y` if you intend to remove the disks in this disk group:

   ```
   Disable (offline) the indicated disks? [y,n,q,?]  (default: n) y
   ```

6. At the following prompt, press Return to continue with the operation:

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

   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

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

   ```
   # vxdisk -s list
   ```

2. Select `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`):

   ```
   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
   ```

Handling cloned disks with duplicated identifiers

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

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

From release 5.0, a unique disk identifier (UDID) is added to the disk's private region when the disk is initialized or when the disk is imported into a disk group (if this identifier does not already exist). Whenever a disk is brought online, the current UDID value that is known to the Device Discovery Layer (DDL) is compared with the UDID that is set in the disk's private region. If the UDID values do not match, the `udid_mismatch` flag is set on the disk. This flag can be viewed with the `vxdisk list` command.

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

The following is sample output from the `vxdisk list` command showing that disks `c2t66d0s2`, `c2t67d0s2` and `c2t68d0s2` are marked with the `udid_mismatch` flag:

```
# vxdisk list

DEVICE       TYPE          DISK     GROUP     STATUS
c0t0d0s2     auto:cdsdisk  -        -         online
c0t1d0s2     auto:cdsdisk  -        -         online
   .
   .
   .
c2t64d0s2    auto:cdsdisk  -        -         online
c2t65d0s2    auto:cdsdisk  -        -         online
c2t66d0s2    auto:cdsdisk  -        -         online udid_mismatch
c2t67d0s2    auto:cdsdisk  -        -         online udid_mismatch
c2t68d0s2    auto:cdsdisk  -        -         online udid_mismatch
```

Writing a new UDID to a disk

You can use the following command to update the unique disk identifier (UDID) for one or more disks:

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

This command uses the current value of the UDID that is stored in the Device Discovery Layer (DDL) database to correct the value in the private region. The `-f` option must be specified if VxVM has not raised the `udid_mismatch` flag for a disk.

For example, the following command updates the UDIDs for the disks `c2t66d0s2` and `c2t67d0s2`:

```
# vxdisk updateudid c2t66d0s2 c2t67d0s2
```

Importing a disk group containing cloned disks

By default, disks on which the `udid_mismatch` flag or the `clone_disk` flag has been set are not imported by the `vxdg import` command unless all disks in the disk group have at least one of these flags set, and no two of the disks have the same UDID. You can then import the cloned disks by specifying the `-o useclonedev=on` option to the `vxdg import` command, as shown in this example:

```
# vxdg -o useclonedev=on [-o updateid] import mydg
```

This form of the command allows only cloned disks to be imported. All non-cloned disks remain unimported.

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

The `-o updateid` option can be specified to write new identification attributes to the disks, and to set the `clone_disk` flag on the disks. (The `vxdisk set clone=on` command can also be used to set the flag.) However, the import fails if multiple copies of one or more cloned disks exist. In this case, you can use the following command to tag all the disks in the disk group that are to be imported:

```
# vxdisk [-g diskgroup ] settag tagname disk ...
```

where *tagname* is a string of up to 128 characters, not including spaces or tabs.

For example, the following command sets the tag, `my_tagged_disks`, on several disks that are to be imported together:

```
# vxdisk settag my_tagged_disks c2t66d0s2 c2t67d0s2
```

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

See "Writing a new UDID to a disk" on page 227.

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

```
# vxdisk listtag

DANAME        DMNAME        NAME                VALUE
c0t0d0s2      mydg01        -                   -
c0t1d0s2      mydg02        -                   -
   .
   .
   .
c2t64d0s2     mydg05        my_tagged_disks     -
c2t65d0s2     mydg06        my_tagged_disks     -
c2t66d0s2     mydg07        my_tagged_disks     -
c2t67d0s2     mydg08        my_tagged_disks     -
c2t68d0s2     mydg09        -                   -
```

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

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

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

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

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

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

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

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

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

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

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

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

See "Renaming a disk group" on page 235.

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

```
#  vxdisk rmtag tag=my_tagged_disks c2t67d0s2
```

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

Sample cases of operations on cloned disks

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

See "Enabling configuration database copies on tagged disks" on page 230.

See "Importing cloned disks without tags" on page 231.

See "Importing cloned disks with tags" on page 233.

Enabling configuration database copies on tagged disks

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

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

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

```
# vxdisk listtag

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

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

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

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

```
# vxdisk -o alldgs list

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

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

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

DEVICE              TYPE         DISK    GROUP   STATUS
TagmaStore-USP0_10 auto:cdsdisk -       -       online
TagmaStore-USP0_22 auto:cdsdisk -       -       online
TagmaStore-USP0_23 auto:cdsdisk -       -       online
TagmaStore-USP0_24 auto:cdsdisk mydg02  mydg    online
TagmaStore-USP0_25 auto:cdsdisk mydg03  mydg    online keepmeta
TagmaStore-USP0_28 auto:cdsdisk mydg01  mydg    online
```

Importing cloned disks without tags

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

```
# vxdisk -o alldgs list

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

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

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

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

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

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

```
# vxdisk -o alldgs list

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Importing cloned disks with tags

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

```
# vxdisk -o alldgs list

DEVICE              TYPE          DISK    GROUP  STATUS
EMC0_4              auto:cdsdisk mydg01  mydg   online
EMC0_6              auto:cdsdisk mydg02  mydg   online
EMC0_8              auto:cdsdisk -      (mydg)  online udid_mismatch
EMC0_15             auto:cdsdisk -      (mydg)  online udid_mismatch
EMC0_18             auto:cdsdisk mydg03  mydg   online
EMC0_24             auto:cdsdisk -      (mydg)  online udid_mismatch
```

The disks are tagged as follows:

```
# vxdisk listtag

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

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

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

DEVICE              TYPE         DISK    GROUP  STATUS
EMC0_4              auto:cdsdisk mydg01  mydg   online
EMC0_6              auto:cdsdisk mydg02  mydg   online
EMC0_8              auto:cdsdisk mydg03  newdg  online clone_disk
EMC0_15             auto:cdsdisk -      (mydg)  online udid_mismatch
EMC0_18             auto:cdsdisk mydg03  mydg   online
EMC0_24             auto:cdsdisk mydg01  newdg  online clone_disk
```

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

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

```
# vxdisk set EMC0_8 clone=off
# vxdisk -o alldgs list

DEVICE              TYPE         DISK    GROUP  STATUS
EMC0_4              auto:cdsdisk mydg01  mydg   online
EMC0_6              auto:cdsdisk mydg02  mydg   online
EMC0_8              auto:cdsdisk mydg03  newdg  online
EMC0_15             auto:cdsdisk -      (mydg)  online udid_mismatch
EMC0_18             auto:cdsdisk mydg03  mydg   online
EMC0_24             auto:cdsdisk mydg01  newdg  online clone_disk
```

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

```
# vxdisk -o alldgs list

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

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

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

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

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

Renaming a disk group

Only one disk group of a given name can exist per system. It is not possible to import or deport a disk group when the target system already has a disk group of the same name. To avoid this problem, VxVM allows you to rename a disk group during import or deport.

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

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

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

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

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

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

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

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

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

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

You cannot use this method to rename the boot disk group because it contains volumes that are in use by mounted file systems (such as /). To rename the boot disk group, you must first unmirror and unencapsulate the root disk, and then re-encapsulate and remirror the root disk in a different disk group. This disk group becomes the new boot disk group.

To temporarily move the boot disk group, bootdg, from one host to another (for repair work on the root volume, for example) and then move it back

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

   ```
   # vxdisk -g bootdg -s list

   dgname: rootdg
   dgid: 774226267.1025.tweety
   ```

   In this example, the administrator has chosen to name the boot disk group as rootdg. The ID of this disk group is 774226267.1025.tweety.

   This procedure assumes that all the disks in the boot disk group are accessible by both hosts.

2. Shut down the original host.

3 On the importing host, import and rename the `rootdg` disk group with this command:

```
# vxdg -tC -n newdg import diskgroup
```

The `-t` option indicates a temporary import name, and the `-C` option clears import locks. The `-n` option specifies an alternate name for the `rootdg` being imported so that it does not conflict with the existing `rootdg`. *diskgroup* is the disk group ID of the disk group being imported (for example, `774226267.1025.tweety`).

If a reboot or crash occurs at this point, the temporarily imported disk group becomes unimported and requires a reimport.

4 After the necessary work has been done on the imported disk group, deport it back to its original host with this command:

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

Here *hostname* is the name of the system whose `rootdg` is being returned (the system name can be confirmed with the command `uname -n`).

This command removes the imported disk group from the importing host and returns locks to its original host. The original host can then automatically import its boot disk group at the next reboot.

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
```

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

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

The preferred method of moving disks between disk groups preserves VxVM objects, such as volumes, that are configured on the disks.

See "Moving objects between disk groups" on page 257.

Moving disk groups between systems

An important feature of disk groups is that they can be moved between systems. If all disks in a disk group are moved from one system to another, then the disk group can be used by the second system. You do not have to re-specify the configuration.

To move a disk group between systems

1. 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
```

Warning: 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.

Warning: 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
```

Warning: 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.

A disk group can be imported successfully if all the disks are accessible that were visible when the disk group was last imported successfully. However, sometimes you may need to specify the `-f` option to forcibly import a disk group if some disks are not available. If the `import` operation fails, an error message is displayed.

The following error message indicates a fatal error that requires hardware repair or the creation of a new disk group, and recovery of the disk group configuration and data:

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

The following error message indicates a recoverable error.

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

If some of the disks in the disk group have failed, you can force the disk group to be imported by specifying the `-f` option to the `vxdg import` command:

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

Warning: Be careful when using the `-f` option. It can cause the same disk group to be imported twice from different sets of disks. This can cause the disk group configuration to become inconsistent.

See "Handling conflicting configuration copies" on page 244.

As using the `-f` option to force the import of an incomplete disk group counts as a successful import, an incomplete disk group may be imported subsequently without this option being specified. This may not be what you expect.

These operations can also be performed using the `vxdiskadm` utility. To deport a disk group using `vxdiskadm`, select `Remove access to (deport) a disk group` from the main menu. To import a disk group, select `Enable access to (import) a disk group`. The `vxdiskadm` import operation checks for host import locks and prompts to see if you want to clear any that are found. It also starts volumes in the disk group.

Reserving minor numbers for disk groups

A device minor number uniquely identifies some characteristic of a device to the device driver that controls that device. It is often used to identify some characteristic mode of an individual device, or to identify separate devices that are all under the control of a single controller. VxVM assigns unique device minor numbers to each object (volume, plex, subdisk, disk, or disk group) that it controls.

When you move a disk group between systems, it is possible for the minor numbers that it used on its previous system to coincide with those of objects known to VxVM on the new system. To get around this potential problem, you can allocate separate ranges of minor numbers for each disk group. VxVM uses the specified range of minor numbers when it creates volume objects from the disks in the disk group. This guarantees that each volume has the same minor number across reboots or reconfigurations. Disk groups may then be moved between machines without causing device number collisions.

VxVM chooses minor device numbers for objects created from this disk group starting at the base minor number *base_minor*. Minor numbers can range from this value up to 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 | grep minors
minors: >=45000

# vxprint -g mydg -m | egrep base_minor
base_minor=45000
```

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

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

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

```
# vxdg init newdg minor=30000 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.

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 256 with a base of 0. This has the effect of limiting the number of volumes and disks that can be supported system-wide to a smaller value than is allowed on other operating system platforms. The number of disks that are supported by a pre-2.6 Linux kernel is typically limited to a few hundred. With the extended major numbering scheme that was implemented in VxVM 4.0 on Linux, a maximum of 4079 volumes could be configured, provided that a contiguous block of 15 extended major numbers was available.

VxVM 4.1 and later releases run on a 2.6 version Linux kernel, which increases the number of minor devices that are configurable from 256 to 65,536 per major device number. This allows a large number of volumes and disk devices to be configured on a system. The theoretical limit on the number of DMP and volume devices that can be configured on such a system are 65,536 and 1,048,576 respectively. However, in practice, the number of VxVM devices that can be configured in a single disk group is limited by the size of the private region.

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

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

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

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

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

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

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

Handling conflicting configuration copies

If an incomplete disk group is imported on several different systems, this can create inconsistencies in the disk group configuration copies that you may need to resolve manually. This section and following sections describe how such a condition can occur, and how to correct it. (When the condition occurs in a cluster that has been split, it is usually referred to as a serial split brain condition).

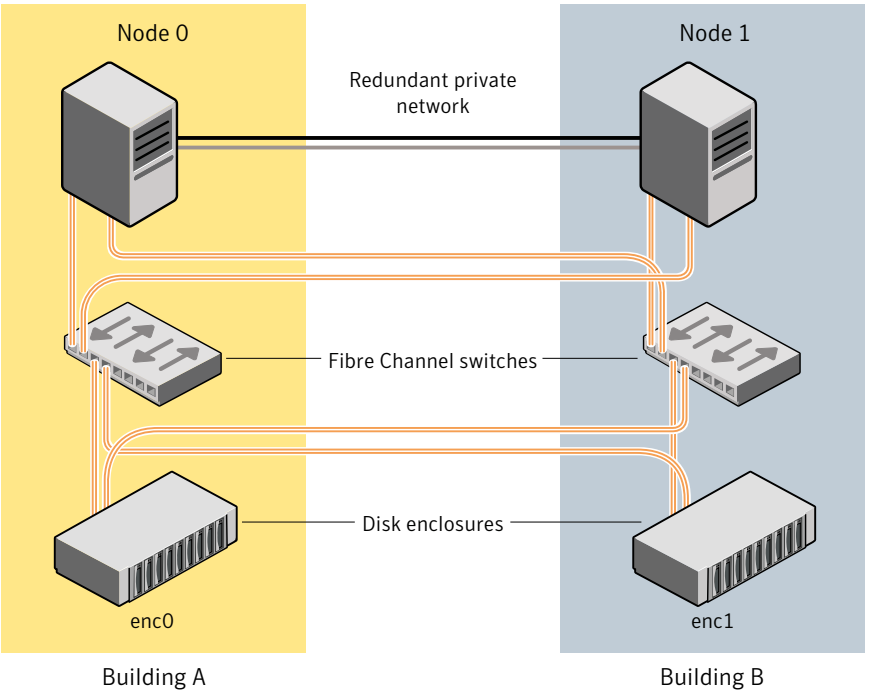
Example of a serial split brain condition in a cluster

This section presents an example of how a serial split brain condition might occur for a shared disk group in a cluster. Conflicts between configuration copies can also occur for private disk groups in clustered and non-clustered configurations where the disk groups have been partially imported on different systems.

A campus cluster (also known as a stretch cluster or remote mirror configuration) typically consists of a 2-node cluster where each component (server, switch and storage) of the cluster exists in a separate building.

Figure 4-1 shows a 2-node cluster with node 0, a fibre channel switch and disk enclosure `enc0` in building A, and node 1, another switch and enclosure `enc1` in building B.

Figure 4-1 Typical arrangement of a 2-node campus cluster



The fibre channel connectivity is multiply redundant to implement redundant-loop access between each node and each enclosure. As usual, the two nodes are also linked by a redundant private network.

A serial split brain condition typically arises in a cluster when a private (non-shared) disk group is imported on Node 0 with Node 1 configured as the failover node.

If the network connections between the nodes are severed, both nodes think that the other node has died. (This is the usual cause of the split brain condition in clusters). If a disk group is spread across both enclosure `enc0` and `enc1`, each portion loses connectivity to the other portion of the disk group. Node 0 continues to update to the disks in the portion of the disk group that it can access. Node 1, operating as the failover node, imports the other portion of the disk group (with the `-f` option set), and starts updating the disks that it can see.

When the network links are restored, attempting to reattach the missing disks to the disk group on Node 0, or to re-import the entire disk group on either node, fails. This serial split brain condition arises because VxVM increments the serial ID in the disk media record of each imported disk in all the disk group configuration databases on those disks, and also in the private region of each imported disk.

The value that is stored in the configuration database represents the serial ID that the disk group expects a disk to have. The serial ID that is stored in a disk's private region is considered to be its actual value.

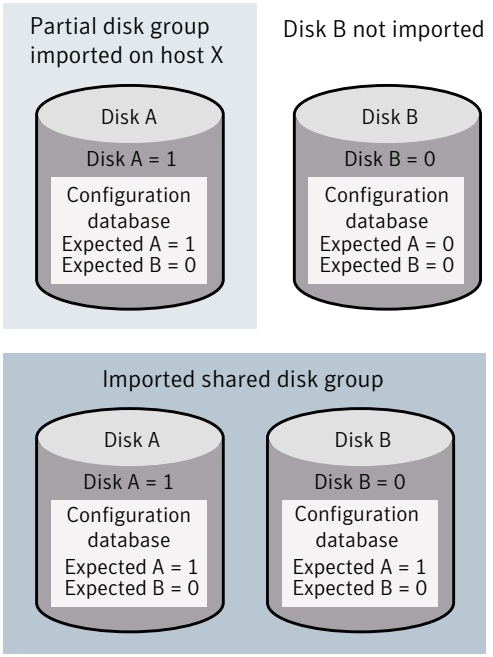
If some disks went missing from the disk group (due to physical disconnection or power failure) and those disks were imported by another host, the serial IDs for the disks in their copies of the configuration database, and also in each disk's private region, are updated separately on that host. When the disks are subsequently re-imported into the original shared disk group, the actual serial IDs on the disks do not agree with the expected values from the configuration copies on other disks in the disk group.

Depending on what happened to the different portions of the split disk group, there are two possibilities for resolving inconsistencies between the configuration databases:

- If the other disks in the disk group were not imported on another host, VxVM resolves the conflicting values of the serial IDs by using the version of the configuration database from the disk with the greatest value for the updated ID (shown as `update_tid` in the output from the `vxdg list` *`diskgroup`* command).

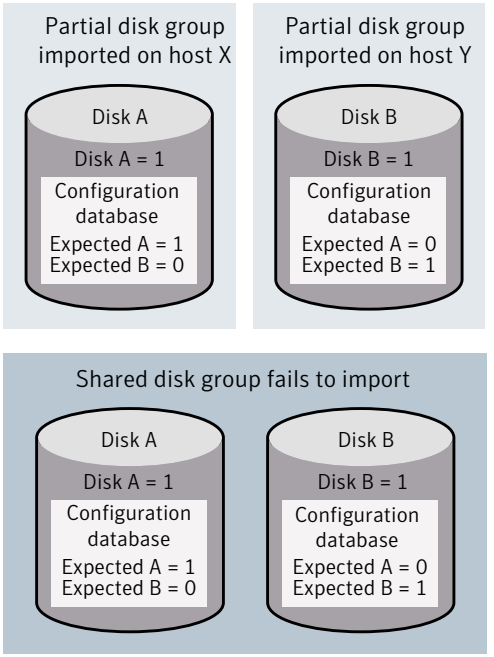
  Figure 4-2 shows an of a serial split brain condition that can be resolved automatically by VxVM.

Figure 4-2 Example of a serial split brain condition that can be resolved automatically



- If the other disks were also imported on another host, no disk can be considered to have a definitive copy of the configuration database.
  Figure 4-3 shows an example of a true serial split brain condition that cannot be resolved automatically by VxVM.

Figure 4-3 Example of a true serial split brain condition that cannot be resolved automatically



In this case, the disk group import fails, and the `vxdg` utility outputs error messages similar to the following before exiting:

```
VxVM vxconfigd NOTICE V-5-0-33 Split Brain. da id is 0.1, while dm id
is 0.0 for DM mydg01
VxVM vxdg ERROR V-5-1-587 Disk group newdg: import failed: Serial Split Brain
detected. Run vxsplitlines
```

The import does not succeed even if you specify the `-f` flag to `vxdg`.

Although it is usually possible to resolve this conflict by choosing the version of the configuration database with the highest valued configuration ID (shown as `config_tid` in the output from the `vxdg list` *`diskgroup`* command), this may not be the correct thing to do in all circumstances.

See "Correcting conflicting configuration information" on page 249.

See "About sites and remote mirrors" on page 511.

Correcting conflicting configuration information

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

Note: The disk group must have a version number of at least 110.

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

```
# vxsplitlines -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
To import the dg with the configuration from this split, run:
    /usr/sbin/vxdg -o selectcp=1045852127.33.olancha import newdg
To get more information about this particular configuration, run:
    /usr/sbin/vxsplitlines -g newdg -c 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.

Reorganizing the contents of disk groups

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

- To group volumes or disks differently as the needs of your organization change. For example, you might want to split disk groups to match the boundaries of separate departments, or to join disk groups when departments are merged.
- To reduce the size of a disk group's configuration database in the event that its private region is nearly full. This is a much simpler solution than the alternative of trying to grow the private region.
- To perform online maintenance and upgrading of fault-tolerant systems that can be split into separate hosts for this purpose, and then rejoined.

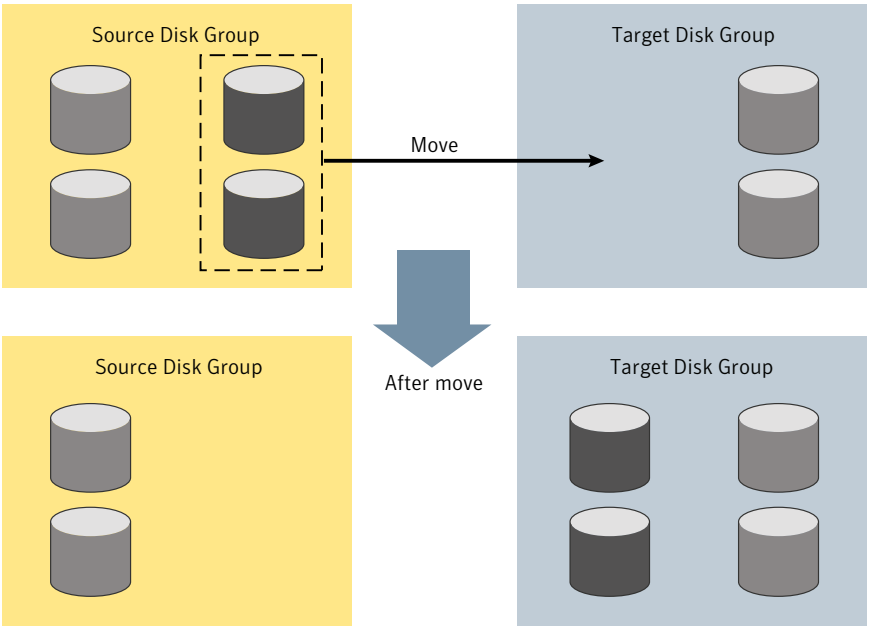
- To isolate volumes or disks from a disk group, and process them independently on the same host or on a different host. This allows you to implement off-host processing solutions for the purposes of backup or decision support.
  See "About off-host processing solutions" on page 443.

You can use either the Veritas Enterprise Administrator (VEA) or the `vxdg` command to reorganize your disk groups.

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

- `move` moves a self-contained set of VxVM objects between imported disk groups. This operation fails if it would remove all the disks from the source disk group. Volume states are preserved across the move.
  Figure 4-4 shows the `move` operation.

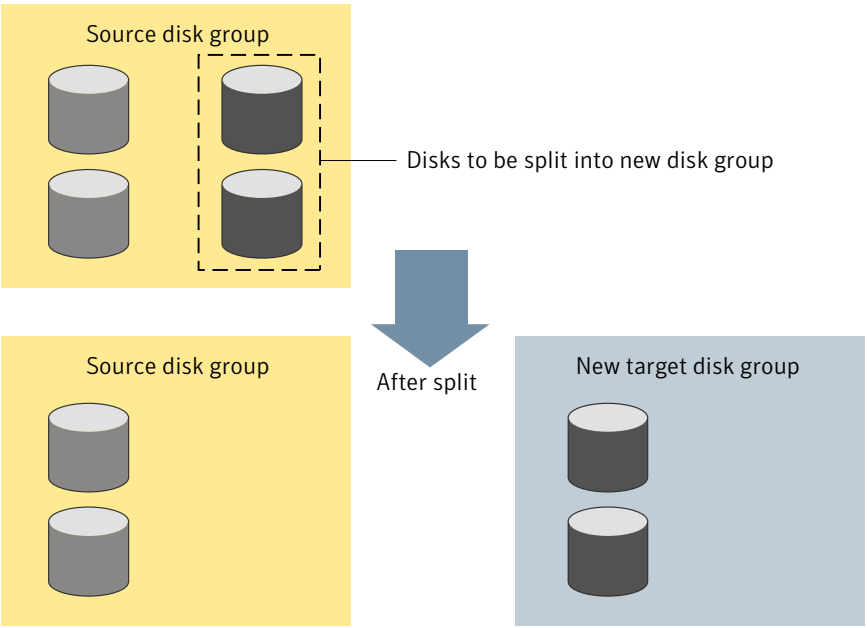
Figure 4-4 Disk group move operation



- `split` removes a self-contained set of VxVM objects from an imported disk group, and moves them to a newly created target disk group. This operation fails if it would remove all the disks from the source disk group, or if an imported disk group exists with the same name as the target disk group. An existing deported disk group is destroyed if it has the same name as the target disk group (as is the case for the `vxdg init` command).

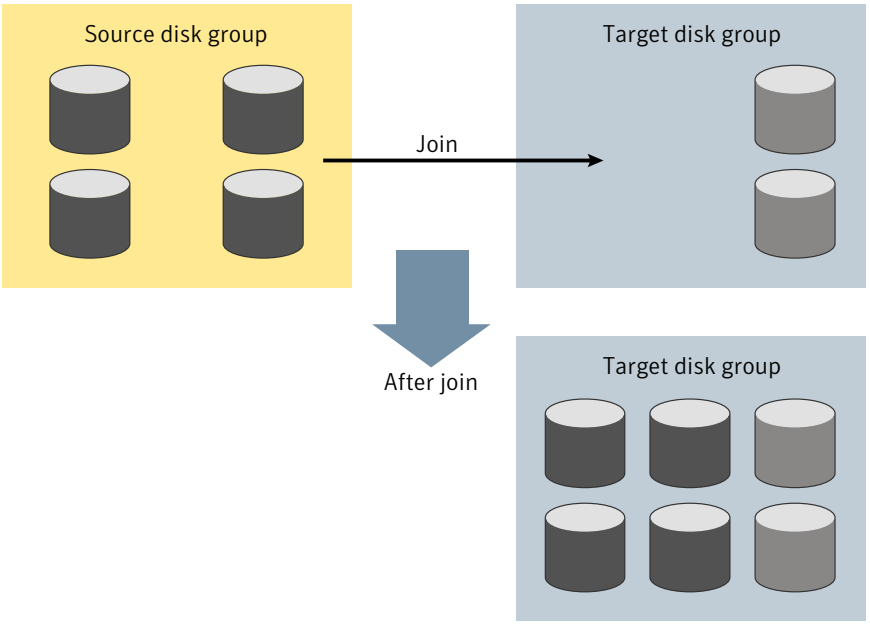
Figure 4-5 shows the `split` operation.

Figure 4-5 Disk group split operation



- `join` removes all VxVM objects from an imported disk group and moves them to an imported target disk group. The source disk group is removed when the join is complete.
  Figure 4-6 shows the `join` operation.

Figure 4-6 Disk group join operation



These operations are performed on VxVM objects such as disks or top-level volumes, and include all component objects such as sub-volumes, plexes and subdisks. The objects to be moved must be self-contained, meaning that the disks that are moved must not contain any other objects that are not intended for the move.

If you specify one or more disks to be moved, all VxVM objects on the disks are moved. You can use the `-o expand` option to ensure that `vxdg` moves all disks on which the specified objects are configured. Take care when doing this as the result may not always be what you expect. You can use the `listmove` operation with `vxdg` to help you establish what is the self-contained set of objects that corresponds to a specified set of objects.

Warning: Before moving volumes between disk groups, stop all applications that are accessing the volumes, and unmount all file systems that are configured on these volumes.

If the system crashes or a hardware subsystem fails, VxVM attempts to complete or reverse an incomplete disk group reconfiguration when the system is restarted or the hardware subsystem is repaired, depending on how far the reconfiguration had progressed. If one of the disk groups is no longer available because it has been

imported by another host or because it no longer exists, you must recover the disk group manually.

See the *Veritas Volume Manager Troubleshooting Guide*.

Limitations of disk group split and join

The disk group split and join feature has the following limitations:

- Disk groups involved in a move, split or join must be version 90 or greater. See "Upgrading a disk group" on page 264.
- The reconfiguration must involve an integral number of physical disks.
- Objects to be moved must not contain open volumes.
- Disks cannot be moved between CDS and non-CDS compatible disk groups.
- Moved volumes are initially disabled following a disk group move, split or join. Use the `vxrecover -m` and `vxvol startall` commands to recover and restart the volumes.
- Data change objects (DCOs) and snap objects that have been dissociated by Persistent FastResync cannot be moved between disk groups.
- Veritas Volume Replicator (VVR) objects cannot be moved between disk groups.
- For a disk group move to succeed, the source disk group must contain at least one disk that can store copies of the configuration database after the move.
- For a disk group split to succeed, both the source and target disk groups must contain at least one disk that can store copies of the configuration database after the split.
- For a disk group move or join to succeed, the configuration database in the target disk group must be able to accommodate information about all the objects in the enlarged disk group.
- Splitting or moving a volume into a different disk group changes the volume's record ID.
- The operation can only be performed on the master node of a cluster if either the source disk group or the target disk group is shared.
- In a cluster environment, disk groups involved in a move or join must both be private or must both be shared.
- When used with objects that have been created using the Veritas Intelligent Storage Provisioning (ISP) feature, only complete storage pools may be split or moved from a disk group. Individual objects such as application volumes within storage pools may not be split or moved.

See the *Veritas Storage Foundation Intelligent Storage Provisioning Administrator's Guide*.

- If a cache object or volume set that is to be split or moved uses ISP volumes, the storage pool that contains these volumes must also be specified.

Listing objects potentially affected by a move

To display the VxVM objects that would be moved for a specified list of objects, use the following command:

```
# vxdg [-o expand] listmove sourcedg targetdg object ...
```

The following example lists the objects that would be affected by moving volume `vol1` from disk group `mydg` to `newdg`:

```
# vxdg listmove mydg newdg vol1
mydg01 c0t1d0s2 mydg05 c1t96d0s2 vol1 vol1-01 vol1-02 mydg01-01
mydg05-01
```

However, the following command produces an error because only a part of the volume `vol1` is configured on the disk `mydg01`:

```
# vxdg listmove mydg newdg mydg01
VxVM vxdg ERROR V-5-2-4597 vxdg listmove mydg newdg failed
VxVM vxdg ERROR V-5-2-3091 mydg05 : Disk not moving, but
subdisks on it are
```

Specifying the `-o expand` option, as shown below, ensures that the list of objects to be moved includes the other disks (in this case, `mydg05`) that are configured in `vol1`:

```
# vxdg -o expand listmove mydg newdg mydg01
mydg01 c0t1d0s2 mydg05 c1t96d0s2 vol1 vol1-01 vol1-02 mydg01-01 mydg05-01
```

Moving DCO volumes between disk groups

When you move the parent volume (such as a snapshot volume) to a different disk group, its DCO volume must accompany it. If you use the `vxassist addlog`, `vxmake` or `vxdco` commands to set up a DCO for a volume, you must ensure that the disks that contain the plexes of the DCO volume accompany their parent volume during the move. You can use the `vxprint` command on a volume to examine the configuration of its associated DCO volume.

If you use the `vxassist` command or the Veritas Enterprise Administrator (VEA) to create both a volume and its DCO, or the `vxsnap prepare` command to add a

DCO to a volume, the DCO plexes are automatically placed on different disks from the data plexes of the parent volume. In previous releases, version 0 DCO plexes were placed on the same disks as the data plexes for convenience when performing disk group split and move operations. As version 20 DCOs support dirty region logging (DRL) in addition to Persistent FastResync, it is preferable for the DCO plexes to be separated from the data plexes. This improves the performance of I/O from/to the volume, and provides resilience for the DRL logs.

Figure 4-7 shows some instances in which it is not be possible to split a disk group because of the location of the DCO plexes on the disks of the disk group.

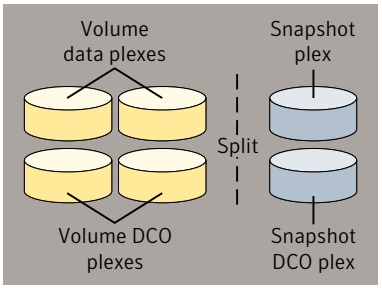
See "Specifying storage for version 0 DCO plexes" on page 431.

See "Specifying storage for version 20 DCO plexes" on page 342.

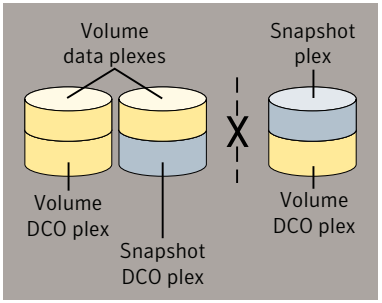
See "FastResync" on page 68.

See "Volume snapshots" on page 65.

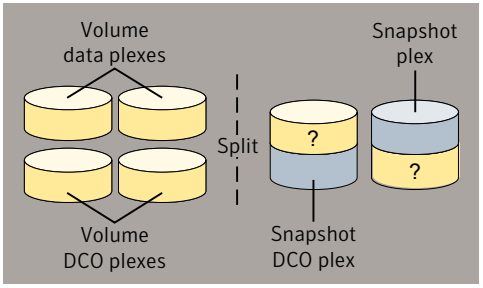
Figure 4-7 Examples of disk groups that can and cannot be split



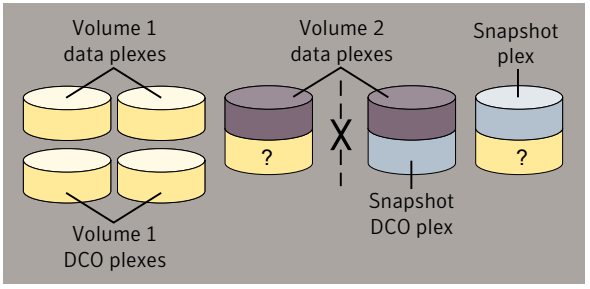
The disk group can be split as the DCO plexes are on dedicated disks, and can therefore accompany the disks that contain the volume data



The disk group cannot be split as the DCO plexes cannot accompany their volumes. One solution is to relocate the DCO plexes. In this example, use an additional disk in the disk group as an intermediary to swap the misplaced DCO plexes. Alternatively, to improve DRL performance and resilience, allocate the DCO plexes to dedicated disks.



The disk group can be split as the DCO plexes can accompany their volumes. However, you may not wish the data in the portions of the disks marked "?" to be moved as well.



The disk group cannot be split as this would separate the disks containing Volume 2's data plexes. Possible solutions are to relocate the snapshot DCO plex to the snapshot plex disk, or to another suitable disk that can be moved.

Moving objects between disk groups

To move a self-contained set of VxVM objects from an imported source disk group to an imported target disk group, use the following command:

```
# vxdg [-o expand] [-o override|verify] move sourcedg targetdg \
  object ...
```

The `-o expand` option ensures that the objects that are actually moved include all other disks containing subdisks that are associated with the specified objects or with objects that they contain.

The default behavior of `vxdg` when moving licensed disks in an EMC array is to perform an EMC disk compatibility check for each disk involved in the move. If the compatibility checks succeed, the move takes place. `vxdg` then checks again to ensure that the configuration has not changed since it performed the compatibility check. If the configuration has changed, `vxdg` attempts to perform the entire move again.

The `-o override` option enables the move to take place without any EMC checking.

The `-o verify` option returns the access names of the disks that would be moved but does not perform the move.

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

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

```
# vxprint
Disk group: rootdg
TY NAME         ASSOC      KSTATE   LENGTH     PLOFFS   STATE   TUTIL0   PUTIL0
dg rootdg       rootdg     -        -          -        -       -        -
dm rootdg02     c1t97d0s2  -        17678493   -        -       -        -
dm rootdg03     c1t112d0s2 -        17678493   -        -       -        -
dm rootdg04     c1t114d0s2 -        17678493   -        -       -        -
dm rootdg06     c1t98d0s2  -        17678493   -        -       -        -

Disk group: mydg
TY NAME         ASSOC      KSTATE   LENGTH     PLOFFS   STATE   TUTIL0   PUTIL0
dg mydg         mydg       -        -          -        -       -        -
dm mydg01       c0t1d0s2   -        17678493   -        -       -        -
dm mydg05       c1t96d0s2  -        17678493   -        -       -        -
dm mydg07       c1t99d0s2  -        17678493   -        -       -        -
dm mydg08       c1t100d0s2 -        17678493   -        -       -        -
v  vol1         fsgen      ENABLED  2048       -        ACTIVE  -        -
pl  vol1-01     vol1       ENABLED  3591       -        ACTIVE  -        -
sd mydg01-01    vol1-01    ENABLED  3591       0        -       -        -
```

```
pl vol1-02     vol1       ENABLED  3591       -        ACTIVE  -        -
sd mydg05-01   vol1-02    ENABLED  3591       0        -       -        -
```

The following command moves the self-contained set of objects implied by specifying disk `mydg01` from disk group `mydg` to `rootdg`:

```
# vxdg -o expand move mydg rootdg mydg01
```

The moved volumes are initially disabled following the move. Use the following commands to recover and restart the volumes in the target disk group:

```
# vxrecover -g targetdg -m [volume ...]
# vxvol -g targetdg startall
```

The output from `vxprint` after the move shows that not only `mydg01` but also volume `vol1` and `mydg05` have moved to `rootdg`, leaving only `mydg07` and `mydg08` in disk group `mydg`:

```
# vxprint
Disk group: rootdg
TY NAME         ASSOC      KSTATE    LENGTH     PLOFFS   STATE   TUTIL0   PUTIL0
dg rootdg       rootdg     -         -          -        -       -        -
dm mydg01       c0t1d0s2   -         17678493   -        -       -        -
dm rootdg02     c1t97d0s2  -         17678493   -        -       -        -
dm rootdg03     c1t112d0s2 -         17678493   -        -       -        -
dm rootdg04     c1t114d0s2 -         17678493   -        -       -        -
dm mydg05       c1t96d0s2  -         17678493   -        -       -        -
dm rootdg06     c1t98d0s2  -         17678493   -        -       -        -
v  vol1         fsgen      ENABLED   2048       -        ACTIVE  -        -
pl  vol1-01     vol1       ENABLED   3591       -        ACTIVE  -        -
sd mydg01-01    vol1-01    ENABLED   3591       0        -       -        -
pl vol1-02      vol1       ENABLED   3591       -        ACTIVE  -        -
sd mydg05-01    vol1-02    ENABLED   3591       0        -       -        -

Disk group: mydg
TY NAME         ASSOC      KSTATE    LENGTH     PLOFFS   STATE   TUTIL0   PUTIL0
dg mydg         mydg       -         -          -        -       -        -
dm mydg07       c1t99d0s2  -         17678493   -        -       -        -
dm mydg08       c1t100d0s2 -         17678493   -        -       -        -
```

The following commands would also achieve the same result:

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

See "Moving objects between shared disk groups" on page 505.

Splitting disk groups

To remove a self-contained set of VxVM objects from an imported source disk group to a new target disk group, use the following command:

```
# vxdg [-o expand] [-o override|verify] split sourcedg targetdg \
  object ...
```

See "Moving objects between disk groups" on page 257.

For example, the following output from `vxprint` shows the contents of disk group `rootdg`:

```
# vxprint
Disk group: rootdg
TY NAME         ASSOC      KSTATE    LENGTH     PLOFFS    STATE    TUTIL0   PUTIL0
dg rootdg       rootdg     -         -          -         -        -        -
dm rootdg01     c0t1d0s2   -         17678493   -         -        -        -
dm rootdg02     c1t97d0s2  -         17678493   -         -        -        -
dm rootdg03     c1t112d0s2 -         17678493   -         -        -        -
dm rootdg04     c1t114d0s2 -         17678493   -         -        -        -
dm rootdg05     c1t96d0s2  -         17678493   -         -        -        -
dm rootdg06     c1t98d0s2  -         17678493   -         -        -        -
dm rootdg07     c1t99d0s2  -         17678493   -         -        -        -
dm rootdg08     c1t100d0s2 -         17678493   -         -        -        -
v  vol1         fsgen      ENABLED   2048       -         ACTIVE   -        -
pl  vol1-01     vol1       ENABLED   3591       -         ACTIVE   -        -
sd rootdg01-01 vol1-01     ENABLED   3591       0         -        -        -
pl vol1-02      vol1       ENABLED   3591       -         ACTIVE   -        -
sd rootdg05-01 vol1-02     ENABLED   3591       0         -        -        -
```

The following command removes disks `rootdg07` and `rootdg08` from `rootdg` to form a new disk group, `mydg`:

```
# vxdg -o expand split rootdg mydg rootdg07 rootdg08
```

The moved volumes are initially disabled following the split. Use the following commands to recover and restart the volumes in the new target disk group:

```
# vxrecover -g targetdg -m [volume ...]
# vxvol -g targetdg startall
```

The output from `vxprint` after the split shows the new disk group, `mydg`:

```
# vxprint
Disk group: rootdg
TY NAME         ASSOC      KSTATE    LENGTH     PLOFFS    STATE    TUTIL0   PUTIL0
```

```
dg rootdg       rootdg     -          -          -         -        -          -
dm rootdg01     c0t1d0s2   -          17678493   -         -        -          -
dm rootdg02     c1t97d0s2  -          17678493   -         -        -          -
dm rootdg03     c1t112d0s2-           17678493   -         -        -          -
dm rootdg04     c1t114d0s2-           17678493   -         -        -          -
dm rootdg05     c1t96d0s2  -          17678493   -         -        -          -
dm rootdg06     c1t98d0s2  -          17678493   -         -        -          -
v  vol1         fsgen      ENABLED    2048       -         ACTIVE   -          -
pl  vol1-01     vol1       ENABLED    3591       -         ACTIVE   -          -
sd rootdg01-01 vol1-01    ENABLED    3591       0         -        -          -
pl vol1-02      vol1       ENABLED    3591       -         ACTIVE   -          -
sd rootdg05-01 vol1-02    ENABLED    3591       0         -        -          -

Disk group: mydg
TY NAME         ASSOC      KSTATE     LENGTH     PLOFFS    STATE    TUTIL0     PUTIL0
dg mydg         mydg       -          -          -         -        -          -
dm rootdg07     c1t99d0s2  -          17678493   -         -        -          -
dm rootdg08     c1t100d0s2-           17678493   -         -        -          -
```

See “Splitting shared disk groups” on page 506.

Joining disk groups

To remove all VxVM objects from an imported source disk group to an imported target disk group, use the following command:

```
# vxdg [-o override|verify] join sourcedg targetdg
```

See “Moving objects between disk groups” on page 257.

Note: You cannot specify `rootdg` as the source disk group for a `join` operation.

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

```
# vxprint
Disk group: rootdg
TY NAME         ASSOC       KSTATE     LENGTH     PLOFFS    STATE    TUTIL0     PUTIL0
dg rootdg       rootdg      -          -          -         -        -          -
dm rootdg01     c0t1d0s2    -          17678493   -         -        -          -
dm rootdg02     c1t97d0s2   -          17678493   -         -        -          -
dm rootdg03     c1t112d0s2  -          17678493   -         -        -          -
dm rootdg04     c1t114d0s2  -          17678493   -         -        -          -
dm rootdg07     c1t99d0s2   -          17678493   -         -        -          -
```

```
dm rootdg08    c1t100d0s2  -          17678493   -         -        -         -

Disk group: mydg
TY NAME         ASSOC       KSTATE     LENGTH     PLOFFS    STATE    TUTIL0    PUTIL0
dg mydg         mydg        -          -          -         -        -         -
dm mydg05       c1t96d0s2   -          17678493   -         -        -         -
dm mydg06       c1t98d0s2   -          17678493   -         -        -         -
v  vol1         fsgen       ENABLED    2048       -         ACTIVE   -         -
pl  vol1-01     vol1        ENABLED    3591       -         ACTIVE   -         -
sd mydg01-01    vol1-01     ENABLED    3591       0         -        -         -
pl vol1-02      vol1        ENABLED    3591       -         ACTIVE   -         -
sd mydg05-01    vol1-02     ENABLED    3591       0         -        -         -
```

The following command joins disk group `mydg` to `rootdg`:

```
# vxdg join mydg rootdg
```

The moved volumes are initially disabled following the join. Use the following commands to recover and restart the volumes in the target disk group:

```
# vxrecover -g targetdg -m [volume ...]
# vxvol -g targetdg startall
```

The output from `vxprint` after the join shows that disk group `mydg` has been removed:

```
# vxprint
Disk group: rootdg
TY NAME         ASSOC       KSTATE     LENGTH     PLOFFS    STATE    TUTIL0    PUTIL0
dg rootdg       rootdg      -          -          -         -        -         -
dm mydg01       c0t1d0s2    -          17678493   -         -        -         -
dm rootdg02     c1t97d0s2   -          17678493   -         -        -         -
dm rootdg03     c1t112d0s2  -          17678493   -         -        -         -
dm rootdg04     c1t114d0s2  -          17678493   -         -        -         -
dm mydg05       c1t96d0s2   -          17678493   -         -        -         -
dm rootdg06     c1t98d0s2   -          17678493   -         -        -         -
dm rootdg07     c1t99d0s2   -          17678493   -         -        -         -
dm rootdg08     c1t100d0s2  -          17678493   -         -        -         -
v  vol1         fsgen       ENABLED    2048       -         ACTIVE   -         -
pl  vol1-01     vol1        ENABLED    3591       -         ACTIVE   -         -
sd mydg01-01    vol1-01     ENABLED    3591       0         -        -         -
pl vol1-02      vol1        ENABLED    3591       -         ACTIVE   -         -
sd mydg05-01    vol1-02     ENABLED    3591       0         -        -         -
```

See "Joining shared disk groups" on page 506.

Disabling a disk group

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

```
# vxdg deport diskgroup
```

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

Destroying a disk group

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

```
# vxdg destroy diskgroup
```

Warning: This command destroys all data on the disks.

When a disk group is destroyed, the disks that are released can be re-used in other disk groups.

Recovering a destroyed disk group

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

To recover a destroyed disk group

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

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

The disk must be specified by its disk access name, such as `c0t12d0s2`. Examine the output from the command for a line similar to the following that specifies the disk group ID.

```
dgid:   963504895.1075.bass
```

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

```
# vxdg import dgid
```

Upgrading a disk group

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.

Note: On some platforms, the first release of Veritas Volume Manager was 3.0 or 3.2.

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.

Table 4-1 summarizes the Veritas Volume Manager releases that introduce and support specific disk group versions.

Table 4-1 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
5.0	140	20-140

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.

Table 4-2 summarizes the features that are supported by disk group versions 20 through 140.

Table 4-2 Features supported by disk group versions

Disk group version	New features supported	Previous version features supported
140	■ Data Migration Features ■ DMP Enhancements ■ EFI Disks ■ Import of Cloned Disks ■ Intelligent Storage Provisioning (ISP) Enhancements ■ Remote Mirror (Campus Cluster)	20, 30, 40, 50, 60, 70, 80, 90, 110, 120, 130
130	■ Veritas Volume Replicator (VVR) Enhancements	20, 30, 40, 50, 60, 70, 80, 90, 110, 120
120	■ Automatic Cluster-wide Failback for A/P arrays ■ 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

Table 4-2 Features supported by disk group versions *(continued)*

Disk group version	New features supported	Previous version features supported
90	■ Cluster Support for Oracle Resilvering ■ Disk Group Move, Split and Join ■ Device Discovery Layer (DDL) 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
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 5.0 creates disk groups with version 140.

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 5.0 is 140. 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 5.0 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.

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

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

See the vxddladm(1M) manual page.

If your system is configured to use Dynamic 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 /dev/vx/dmp and /dev/vx/rdmp directories.
- Update the DMP database with changes in path type for active/passive disk arrays. Use the utilities provided by the disk-array vendor to change the path type between primary and secondary.

See the vxdctl(1M) manual page.

Backing up and restoring disk group configuration data

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

See the *Veritas Volume Manager Troubleshooting Guide*.

See the vxconfigbackup(1M) manual page.

See the vxconfigrestore(1M) manual page.

Using vxnotify to monitor configuration changes

You can use the vxnotify utility to display events relating to disk and configuration changes that are managed by the vxconfigd configuration daemon. If vxnotify is running on a system where the VxVM clustering feature is active, it displays events that are related to changes in the cluster state of the system on which it is running. The vxnotify utility displays the requested event types until you kill it, until it has received a specified number of events, or until a specified period of time has elapsed.

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

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

```
# vxnotify -f
```

The following command provides information about cluster configuration changes, including the import and deport of shared disk groups:

```
# vxnotify -s -i
```

See the vxnotify(1M) manual page.

Chapter 5

Creating and administering subdisks

This chapter includes the following topics:

- About subdisks
- Creating subdisks
- Displaying subdisk information
- Moving subdisks
- Splitting subdisks
- Joining subdisks
- Associating subdisks with plexes
- Associating log subdisks
- Dissociating subdisks from plexes
- Removing subdisks
- Changing subdisk attributes

About subdisks

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

Subdisks are created automatically if you use the `vxassist` command or the Veritas Enterprise Administrator (VEA) to create volumes.

See "Creating a volume" on page 299.

Note: Most VxVM commands require superuser or equivalent privileges.

Creating subdisks

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

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

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

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

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

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

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

See "Associating subdisks with plexes" on page 275.

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

Displaying subdisk information

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

```
# vxprint -st
```

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

The following is example output:

```
SD NAME     PLEX       DISK    DISKOFFS LENGTH [COL/]OFF DEVICE   MODE
SV NAME     PLEX       VOLNAME NVOLLAYR LENGTH [COL/]OFF AM/NM   MODE
```

```
sd mydg01-01 vol1-01 mydg01 0          102400 0      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:   device=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.

Subdisk can also be moved manually after hot-relocation.

See "Moving relocated subdisks" on page 467.

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.

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

See "Dirty region logging" on page 62.

See "Dirty region logging in cluster environments" on page 496.

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

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

See "Preparing a volume for DRL and instant snapshots" on page 341.

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

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

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

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

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

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

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

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

Dissociating subdisks from plexes

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

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

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

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

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

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

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

Removing subdisks

To remove a subdisk, use the following command:

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

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

```
# vxedit -g mydg rm mydg02-01
```

Changing subdisk attributes

Warning: To avoid possible data loss, change subdisk attributes with extreme care.

The `vxedit` command changes attributes of subdisks and other VxVM objects. To change subdisk attributes, use the following command:

```
# vxedit [-g diskgroup] set attribute=value ... subdisk ...
```

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 Symantec 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.

Chapter 6

Creating and administering plexes

This chapter includes the following topics:

- About plexes
- Creating plexes
- Creating a striped plex
- Displaying plex information
- Attaching and associating plexes
- Taking plexes offline
- Detaching plexes
- Reattaching plexes
- Moving plexes
- Copying plexes
- Dissociating and removing plexes
- Changing plex attributes

About plexes

Plexes are logical groupings of subdisks that create an area of disk space independent of physical disk size or other restrictions. Replication (mirroring) of disk data is set up by creating multiple data plexes for a single volume. Each data plex in a mirrored volume contains an identical copy of the volume data. Because

each data plex must reside on different disks from the other plexes, the replication provided by mirroring prevents data loss in the event of a single-point disk-subsystem failure. Multiple data plexes also provide increased data integrity and reliability.

Plexes are created automatically if you use the `vxassist` command or the Veritas Enterprise Administrator (VEA) to create volumes.

See "About subdisks" on page 271.

See "Creating a volume" on page 299.

Note: Most VxVM commands require superuser or equivalent privileges.

Creating plexes

Use the `vxmake` command to create VxVM objects, such as plexes. When creating a plex, identify the subdisks that are to be associated with it:

To create a plex from existing subdisks, use the following command:

```
# vxmake [-g diskgroup] plex plex sd=subdisk1[,subdisk2,...]
```

For example, to create a concatenated plex named `vol01-02` from two existing subdisks named `mydg02-01` and `mydg02-02` in the disk group, `mydg`, use the following command:

```
# vxmake -g mydg plex vol01-02 sd=mydg02-01,mydg02-02
```

Creating a striped plex

To create a striped plex, you must specify additional attributes. For example, to create a striped plex named `pl-01` in the disk group, `mydg`, with a stripe width of 32 sectors and 2 columns, use the following command:

```
# vxmake -g mydg plex pl-01 layout=stripe stwidth=32 ncolumn=2 \
  sd=mydg01-01,mydg02-01
```

To use a plex to build a volume, you must associate the plex with the volume.

See "Attaching and associating plexes" on page 287.

Displaying plex information

Listing plexes helps identify free plexes for building volumes. Use the plex (-p) option to the vxprint command to list information about all plexes.

To display detailed information about all plexes in the system, use the following command:

```
# vxprint -lp
```

To display detailed information about a specific plex, use the following command:

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

The -t option prints a single line of information about the plex. To list free plexes, use the following command:

```
# vxprint -pt
```

The following section describes the meaning of the various plex states that may be displayed in the STATE field of vxprint output.

Plex states

Plex states reflect whether or not plexes are complete and are consistent copies (mirrors) of the volume contents. VxVM utilities automatically maintain the plex state. However, if a volume should not be written to because there are changes to that volume and if a plex is associated with that volume, you can modify the state of the plex. For example, if a disk with a particular plex located on it begins to fail, you can temporarily disable that plex.

A plex does not have to be associated with a volume. A plex can be created with the vxmake plex command and be attached to a volume later.

VxVM utilities use plex states to:

- indicate whether volume contents have been initialized to a known state
- determine if a plex contains a valid copy (mirror) of the volume contents
- track whether a plex was in active use at the time of a system failure
- monitor operations on plexes

This section explains the individual plex states in detail.

See the *Veritas Volume Manager Troubleshooting Guide*.

Table 6-1shows the states that may be associated with a plex.

Table 6-1 Plex states

State	Description
ACTIVE	A plex can be in the ACTIVE state in the following ways: ■ when the volume is started and the plex fully participates in normal volume I/O (the plex contents change as the contents of the volume change) ■ when the volume is stopped as a result of a system crash and the plex is ACTIVE at the moment of the crash In the latter case, a system failure can leave plex contents in an inconsistent state. When a volume is started, VxVM does the recovery action to guarantee that the contents of the plexes marked as ACTIVE are made identical. On a system that is running well, ACTIVE should be the most common state you see for any volume plexes.
CLEAN	A plex is in a CLEAN state when it is known to contain a consistent copy (mirror) of the volume contents and an operation has disabled the volume. As a result, when all plexes of a volume are clean, no action is required to guarantee that the plexes are identical when that volume is started.
DCOSNP	This state indicates that a data change object (DCO) plex attached to a volume can be used by a snapshot plex to create a DCO volume during a snapshot operation.
EMPTY	Volume creation sets all plexes associated with the volume to the EMPTY state to indicate that the plex is not yet initialized.
IOFAIL	The IOFAIL plex state is associated with persistent state logging. When the `vxconfigd` daemon detects an uncorrectable I/O failure on an ACTIVE plex, it places the plex in the IOFAIL state to exclude it from the recovery selection process at volume start time. This state indicates that the plex is out-of-date with respect to the volume, and that it requires complete recovery. It is likely that one or more of the disks associated with the plex should be replaced.
LOG	The state of a dirty region logging (DRL) or RAID-5 log plex is always set to LOG.

Table 6-1 Plex states *(continued)*

State	Description
OFFLINE	The `vxmend off` task indefinitely detaches a plex from a volume by setting the plex state to OFFLINE. Although the detached plex maintains its association with the volume, changes to the volume do not update the OFFLINE plex. The plex is not updated until the plex is put online and reattached with the `vxplex att` task. When this occurs, the plex is placed in the STALE state, which causes its contents to be recovered at the next `vxvol start` operation.
SNAPATT	This state indicates a snapshot plex that is being attached by the snapstart operation. When the attach is complete, the state for the plex is changed to SNAPDONE. If the system fails before the attach completes, the plex and all of its subdisks are removed.
SNAPDIS	This state indicates a snapshot plex that is fully attached. A plex in this state can be turned into a snapshot volume with the `vxplex snapshot` command. If the system fails before the attach completes, the plex is dissociated from the volume. See the `vxplex`(1M) manual page.
SNAPDONE	The SNAPDONE plex state indicates that a snapshot plex is ready for a snapshot to be taken using `vxassist snapshot`.
SNAPTMP	The SNAPTMP plex state is used during a `vxassist snapstart` operation when a snapshot is being prepared on a volume.
STALE	If there is a possibility that a plex does not have the complete and current volume contents, that plex is placed in the STALE state. Also, if an I/O error occurs on a plex, the kernel stops using and updating the contents of that plex, and the plex state is set to STALE. A `vxplex att` operation recovers the contents of a STALE plex from an ACTIVE plex. Atomic copy operations copy the contents of the volume to the STALE plexes. The system administrator can force a plex to the STALE state with a `vxplex det` operation.
TEMP	Setting a plex to the TEMP state eases some plex operations that cannot occur in a truly atomic fashion. For example, attaching a plex to an enabled volume requires copying volume contents to the plex before it can be considered fully attached. A utility sets the plex state to TEMP at the start of such an operation and to an appropriate state at the end of the operation. If the system fails for any reason, a TEMP plex state indicates that the operation is incomplete. A later `vxvol start` dissociates plexes in the TEMP state.

Table 6-1 Plex states *(continued)*

State	Description
TEMPRM	A TEMPRM plex state is similar to a TEMP state except that at the completion of the operation, the TEMPRM plex is removed. Some subdisk operations require a temporary plex. Associating a subdisk with a plex, for example, requires updating the subdisk with the volume contents before actually associating the subdisk. This update requires associating the subdisk with a temporary plex, marked TEMPRM, until the operation completes and removes the TEMPRM plex. If the system fails for any reason, the TEMPRM state indicates that the operation did not complete successfully. A later operation dissociates and removes TEMPRM plexes.
TEMPRMSD	The TEMPRMSD plex state is used by `vxassist` when attaching new data plexes to a volume. If the synchronization operation does not complete, the plex and its subdisks are removed.

Plex condition flags

Table 6-2 shows the plex condition flags that `vxprint` may display in the STATE field.

Table 6-2 Plex condition flags

Condition flag	Description
IOFAIL	The plex was detached as a result of an I/O failure detected during normal volume I/O. The plex is out-of-date with respect to the volume, and in need of complete recovery. However, this condition also indicates a likelihood that one of the disks in the system should be replaced.
NODAREC	No physical disk was found for one of the subdisks in the plex. This implies either that the physical disk failed, making it unrecognizable, or that the physical disk is no longer attached through a known access path. The plex cannot be used until this condition is fixed, or the affected subdisk is dissociated.
NODEVICE	A physical device could not be found corresponding to the disk ID in the disk media record for one of the subdisks associated with the plex. The plex cannot be used until this condition is fixed, or the affected subdisk is dissociated.

Table 6-2 Plex condition flags *(continued)*

Condition flag	Description
RECOVER	A disk corresponding to one of the disk media records was replaced, or was reattached too late to prevent the plex from becoming out-of-date with respect to the volume. The plex required complete recovery from another plex in the volume to synchronize its contents.
REMOVED	Set in the disk media record when one of the subdisks associated with the plex is removed. The plex cannot be used until this condition is fixed, or the affected subdisk is dissociated.

Plex kernel states

The plex kernel state indicates the accessibility of the plex to the volume driver which monitors it.

No user intervention is required to set these states; they are maintained internally. On a system that is operating properly, all plexes are enabled.

Table 6-3 shows the possible plex kernel states.

Table 6-3 Plex kernel states

Kernel state	Description
DETACHED	Maintenance is being performed on the plex. Any write request to the volume is not reflected in the plex. A read request from the volume is not satisfied from the plex. Plex operations and `ioctl` function calls are accepted.
DISABLED	The plex is offline and cannot be accessed.
ENABLED	The plex is online. A write request to the volume is reflected in the plex. A read request from the volume is satisfied from the plex. If a plex is sparse, this is indicated by the `SPARSE` modifier being displayed in the output from the `vxprint -t` command.

Attaching and associating plexes

A plex becomes a participating plex for a volume by attaching it to a volume. (Attaching a plex associates it with the volume and enables the plex for use.) To attach a plex to an existing volume, use the following command:

```
# vxplex [-g diskgroup] att volume plex
```

For example, to attach a plex named vol01-02 to a volume named vol01 in the disk group, mydg, use the following command:

```
# vxplex -g mydg att vol01 vol01-02
```

If the volume does not already exist, 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
```

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, it can be reattached to the volume.

See "Reattaching plexes" on page 289.

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, set one of the plexes to `CLEAN` using the following command:

```
# vxmend [-g diskgroup] fix clean plex
```

Start the volume using the following command:

```
# vxvol [-g diskgroup] start volume
```

See the *Veritas Volume Manager Troubleshooting Guide*.

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

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 283."

To dissociate a plex from the associated volume and remove it as an object from VxVM, use the following command:

```
# vxplex [-g diskgroup] -o rm dis plex
```

For example, to dissociate and remove a plex named `vol01-02` in the disk group, `mydg`, use the following command:

```
# vxplex -g mydg -o rm dis vol01-02
```

This command removes the plex vol01-02 and all associated subdisks.

Alternatively, you can first dissociate the plex and subdisks, and then remove them with the following commands:

```
# vxplex [-g diskgroup] dis plex
# vxedit [-g diskgroup] -r rm plex
```

When used together, these commands produce the same result as the vxplex -o rm dis command. The -r option to vxedit rm recursively removes all objects from the specified object downward. In this way, a plex and its associated subdisks can be removed by a single vxedit command.

Changing plex attributes

Warning: To avoid possible data loss, change plex attributes with extreme care.

The vxedit command changes the attributes of plexes and other volume Manager objects. To change plex attributes, use the following command:

```
# vxedit [-g diskgroup] set attribute=value ... plex
```

Plex fields that can be changed using the vxedit command include:

- name
- *putiln*
- *tutiln*
- comment

The *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 Symantec 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.

Chapter 7

Creating volumes

This chapter includes the following topics:

- About volume creation
- Types of volume layouts
- Creating a volume
- Using vxassist
- Discovering the maximum size of a volume
- Disk group alignment constraints on volumes
- Creating a volume on any disk
- Creating a volume on specific disks
- Creating a mirrored volume
- Creating a volume with a version 0 DCO volume
- Creating a volume with a version 20 DCO volume
- Creating a volume with dirty region logging enabled
- Creating a striped volume
- Mirroring across targets, controllers or enclosures
- Creating a RAID-5 volume
- Creating tagged volumes
- Creating a volume using vxmake
- Initializing and starting a volume

■ Accessing a volume

About volume creation

Volumes are logical devices that appear as physical disk partition devices to data management systems. Volumes enhance recovery from hardware failure, data availability, performance, and storage configuration.

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.

See the *Veritas Storage Foundation Intelligent Storage Provisioning Administrator's Guide*.

Volumes are created to take advantage of the VxVM concept of virtual disks. A file system can be placed on the volume to organize the disk space with files and directories. In addition, you can configure applications such as databases to organize data on volumes.

Disks and disk groups must be initialized and defined to VxVM before volumes can be created from them.

See "About disk management" on page 82.

See "About disk groups" on page 216.

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. See "Concatenation and spanning" on page 40.

Striped	A volume with data spread evenly across multiple disks. Stripes are equal-sized fragments that are allocated alternately and evenly to the subdisks of a single plex. There must be at least two subdisks in a striped plex, each of which must exist on a different disk. Throughput increases with the number of disks across which a plex is striped. Striping helps to balance I/O load in cases where high traffic areas exist on certain subdisks. See "Striping (RAID-0)" on page 42.
Mirrored	A volume with multiple data plexes that duplicate the information contained in a volume. Although a volume can have a single data plex, at least two are required for true mirroring to provide redundancy of data. For the redundancy to be useful, each of these data plexes should contain disk space from different disks. See "Mirroring (RAID-1)" on page 46.
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. See "RAID-5 (striping with parity)" on page 49.
Mirrored-stripe	A volume that is configured as a striped plex and another plex that mirrors the striped one. This requires at least two disks for striping and one or more other disks for mirroring (depending on whether the plex is simple or striped). The advantages of this layout are increased performance by spreading data across multiple disks and redundancy of data. See "Striping plus mirroring (mirrored-stripe or RAID-0+1)" on page 47.

Layered Volume	A volume constructed from other volumes. Non-layered volumes are constructed by mapping their subdisks to VM disks. Layered volumes are constructed by mapping their subdisks to underlying volumes (known as storage volumes), and allow the creation of more complex forms of logical layout. Examples of layered volumes are striped-mirror and concatenated-mirror volumes. See Layered volumes. A striped-mirror volume is created by configuring several mirrored volumes as the columns of a striped volume. This layout offers the same benefits as a non-layered mirrored-stripe volume. In addition it provides faster recovery as the failure of single disk does not force an entire striped plex offline. See Mirroring plus striping (striped-mirror, RAID-1+0 or RAID-10). A concatenated-mirror volume is created by concatenating several mirrored volumes. This provides faster recovery as the failure of a single disk does not force the entire mirror offline.

Supported volume logs and maps

Veritas Volume Manager supports the use of the following types of logs and maps with volumes:

- FastResync Maps are used to perform quick and efficient resynchronization of mirrors.
  See "FastResync" on page 68.

  These maps are supported either in memory (Non-Persistent FastResync), or on disk as part of a DCO volume (Persistent FastResync). Two types of DCO volume are supported:

  - Version 0 DCO volumes only support Persistent FastResync for the traditional third-mirror break-off type of volume snapshot.
    See "Version 0 DCO volume layout" on page 70.
    See "Creating a volume with a version 0 DCO volume" on page 311.
  - Version 20 DCO volumes, introduced in VxVM 4.0, support DRL logging (see below) and Persistent FastResync for full-sized and space-optimized instant volume snapshots.
    See "Version 20 DCO volume layout" on page 71.
    See "Creating a volume with a version 20 DCO volume" on page 315.
    See "Enabling FastResync on a volume" on page 361.
- Dirty region logs allow the fast recovery of mirrored volumes after a system crash.

See “Dirty region logging” on page 62.
These logs are supported either as DRL log plexes, or as part of a version 20 DCO volume. Refer to the following sections for information on creating a volume on which DRL is enabled:
See “Creating a volume with dirty region logging enabled” on page 315.
See “Creating a volume with a version 20 DCO volume” on page 315.

- RAID-5 logs are used to prevent corruption of data during recovery of RAID-5 volumes.
  See “RAID-5 logging” on page 54.
  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 319.

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.

Note: Most VxVM commands require superuser or equivalent privileges.

Advanced approach

The advanced approach consists of a number of commands that typically require you to specify detailed input. These commands use a “building block” approach that requires you to have a detailed knowledge of the underlying structure and components to manually perform the commands necessary to accomplish a certain task. Advanced operations are performed using several different VxVM commands.

To create a volume using the advanced approach, perform the following steps in the order specified:

- Create subdisks using `vxmake sd`.
  See “Creating subdisks” on page 272.
- Create plexes using `vxmake plex`, and associate subdisks with them.
  See “Creating plexes” on page 282.
  See “Associating subdisks with plexes” on page 275.
  See “Creating a volume using vxmake” on page 322.
- Associate plexes with the volume using `vxmake vol`.

See "Creating a volume using vxmake" on page 322.

- Initialize the volume using `vxvol start` or `vxvol init zero`.
  See "Initializing and starting a volume created using vxmake" on page 325.

The steps to create the subdisks and plexes, and to associate the plexes with the volumes can be combined by using a volume description file with the `vxmake` command.

See "Creating a volume using a vxmake description file" on page 323.

See "Creating a volume using vxmake" on page 322.

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.

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, gigabytes or terabytes by using a suffix character of s, k, m, g, or t. If no suffix is specified, the size is assumed to be in sectors.

See the vxintro(1M) manual page.

Additional attributes can be specified as appropriate, depending on the characteristics that you wish the volume to have. Examples are stripe unit width, number of columns in a RAID-5 or stripe volume, number of mirrors, number of logs, and log type.

By default, the vxassist command creates volumes in a default disk group according to a set of rules.

See "Rules for determining the default disk group" on page 218.

To use a different disk group, specify the *-g diskgroup* option to vxassist.

A large number of vxassist keywords and attributes are available for use.

See the vxassist(1M) manual page.

The simplest way to create a volume is to use default attributes.

Creating a volume on any disk

More complex volumes can be created with specific attributes by controlling how vxassist uses the available storage space.

See "Creating a volume on specific disks" on page 305.

Setting default values for vxassist

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

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

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

See the vxassist(1M) manual page.

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

```
# vxassist help showattrs
```

The following is a sample vxassist defaults file:

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

#  use the fsgen usage type, except when creating RAID-5 volumes
    usetype=fsgen
```

```
#  allow only root access to a volume
     mode=u=rw,g=,o=
     user=root
     group=root

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

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

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

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

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

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

Using the SmartMove™ feature while attaching a plex

The SmartMove™ feature reduces the time and I/O required to attach or reattach a plex to an existing VxVM volume, in the specific case where a VxVM volume has a VxFS file system mounted on it. The SmartMove feature uses the VxFS information to detect free extents and avoid copying them.

To turn on the SmartMove feature, specify `usefssmartmove=all` in the file `/etc/default/vxsf`. This tunable is system-wide and persistent, so it only needs to be set once per server. The SmartMove feature is turned off by default.

The SmartMove feature takes effect when a plex is attached or reattached using the `vxplex`, `vxsd`, or `vxassist` commands.

Note: The file system must be mounted to get the benefits of the SmartMove™ feature.

When the SmartMove feature is on, less I/O is sent through the host, through the storage network and to the disks or LUNs. The SmartMove feature can be used for faster plex creation and faster array migrations.

The SmartMove feature enables migration from a traditional LUN to a thinly provisioned LUN, removing unused space in the process.

See "Migrating to thin provisioning" on page 609.

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 305.

The maximum size of a VxVM volume that you can create is 256TB.

Disk group alignment constraints on volumes

Certain constraints apply to the length of volumes and to the numeric values of size attributes that apply to volumes. If a volume is created in a disk group that is compatible with the Cross-platform Data Sharing (CDS) feature, the volume's length and the values of volume attributes that define the sizes of objects such as logs or stripe units, must be an integer multiple of the alignment value of 16 blocks (8 kilobytes). If the disk group is not compatible with the CDS feature, the volume's length and attribute size values must be multiples of 1 block (512 bytes).

To discover the value in blocks of the alignment that is set on a disk group, use this command:

```
# vxprint -g diskgroup -G -F %align
```

By default, `vxassist` automatically rounds up the volume size and attribute size values to a multiple of the alignment value. (This is equivalent to specifying the attribute `dgalign_checking=round` as an additional argument to the `vxassist` command.)

If you specify the attribute `dgalign_checking=strict` to `vxassist`, the command fails with an error if you specify a volume length or attribute size value that is not a multiple of the alignment value for the disk group.

Creating a volume on any disk

By default, the `vxassist make` command creates a concatenated volume that uses one or more sections of disk space. On a fragmented disk, this allows you to put together a volume larger than any individual section of free disk space available.

To change the default layout, edit the definition of the `layout` attribute defined in the `/etc/default/vxassist` file.

If there is not enough space on a single disk, `vxassist` creates a spanned volume. A spanned volume is a concatenated volume with sections of disk space spread across more than one disk. A spanned volume can be larger than any disk on a system, since it takes space from more than one disk.

To create a concatenated, default volume, use the following form of the `vxassist` command:

```
# vxassist [-b] [-g diskgroup] make volume length
```

Specify the `-b` option if you want to make the volume immediately available for use.

See "Initializing and starting a volume" on page 324.

For example, to create the concatenated volume `voldefault` with a length of 10 gigabytes in the default disk group:

```
# vxassist -b make voldefault 10g
```

Creating a volume on specific disks

VxVM automatically selects the disks on which each volume resides, unless you specify otherwise. If you want a volume to be created on specific disks, you must designate those disks to VxVM. More than one disk can be specified.

To create a volume on a specific disk or disks, use the following command:

```
# vxassist [-b] [-g diskgroup] make volume length \
  [layout=layout] diskname ...
```

Specify the `-b` option if you want to make the volume immediately available for use.

See "Initializing and starting a volume" on page 324.

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
```

Any storage attributes that you specify for use must belong to the disk group. Otherwise, `vxassist` will not use them to create a volume.

You can also use storage attributes to control how `vxassist` uses available storage, for example, when calculating the maximum size of a volume, when growing a volume or when removing mirrors or logs from a volume. The following example excludes disks `dgrp07` and `dgrp08` when calculating the maximum size of RAID-5 volume that `vxassist` can create using the disks in the disk group `dg`:

```
# vxassist -b -g dgrp maxsize layout=raid5 nlog=2 !dgrp07 !dgrp08
```

It is also possible to control how volumes are laid out on the specified storage.

See "Specifying ordered allocation of storage to volumes" on page 307.

If you are using VxVM in conjunction with Veritas SANPoint Control 2.0, you can specify how `vxassist` should use the available storage groups when creating volumes.

See "Configuration of volumes on SAN storage" on page 77.

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

Specifying ordered allocation of storage to volumes

Ordered allocation gives you complete control of space allocation. It requires that the number of disks that you specify to the `vxassist` command must match the number of disks that are required to create a volume. The order in which you specify the disks to `vxassist` is also significant.

If you specify the `-o ordered` option to `vxassist` when creating a volume, any storage that you also specify is allocated in the following order:

- Concatenate disks.
- Form columns.
- Form mirrors.

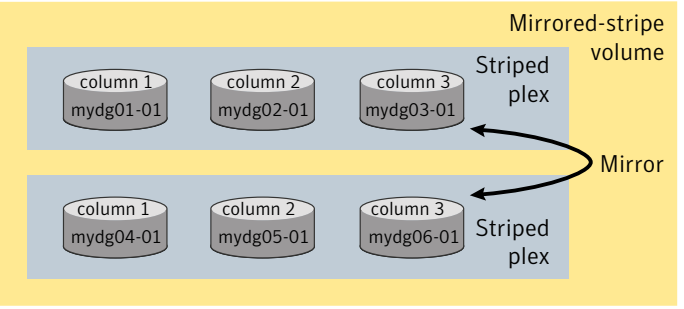
For example, the following command creates a mirrored-stripe volume with 3 columns and 2 mirrors on 6 disks in the disk group, `mydg`:

```
# vxassist -b -g mydg -o ordered make mirstrvol 10g \
  layout=mirror-stripe ncol=3 mydg01 mydg02 mydg03 mydg04 mydg05 mydg06
```

This command places columns 1, 2 and 3 of the first mirror on disks `mydg01`, `mydg02` and `mydg03` respectively, and columns 1, 2 and 3 of the second mirror on disks `mydg04`, `mydg05` and `mydg06` respectively.

Figure 7-1 shows an example of using ordered allocation to create a mirrored-stripe volume.

Figure 7-1 Example of using ordered allocation to create a mirrored-stripe volume



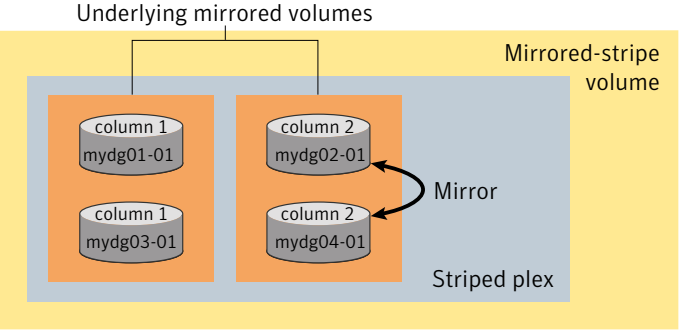
For layered volumes, `vxassist` applies the same rules to allocate storage as for non-layered volumes. For example, the following command creates a striped-mirror volume with 2 columns:

```
# vxassist -b -g mydg -o ordered make strmirvol 10g \
  layout=stripe-mirror ncol=2 mydg01 mydg02 mydg03 mydg04
```

This command mirrors column 1 across disks `mydg01` and `mydg03`, and column 2 across disks `mydg02` and `mydg04`.

Figure 7-2 shows an example of using ordered allocation to create a striped-mirror volume.

Figure 7-2 Example of using ordered allocation to create a striped-mirror volume



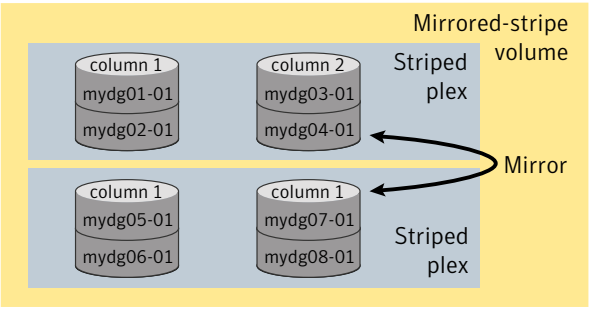
Additionally, you can use the `col_switch` attribute to specify how to concatenate space on the disks into columns. For example, the following command creates a mirrored-stripe volume with 2 columns:

```
# vxassist -b -g mydg -o ordered make strmir2vol 10g \
  layout=mirror-stripe ncol=2 col_switch=3g,2g \
  mydg01 mydg02 mydg03 mydg04 mydg05 mydg06 mydg07 mydg08
```

This command allocates 3 gigabytes from `mydg01` and 2 gigabytes from `mydg02` to column 1, and 3 gigabytes from `mydg03` and 2 gigabytes from `mydg04` to column 2. The mirrors of these columns are then similarly formed from disks `mydg05` through `mydg08`.

Figure 7-3 shows an example of using concatenated disk space to create a mirrored-stripe volume.

Figure 7-3 Example of using concatenated disk space to create a mirrored-stripe volume



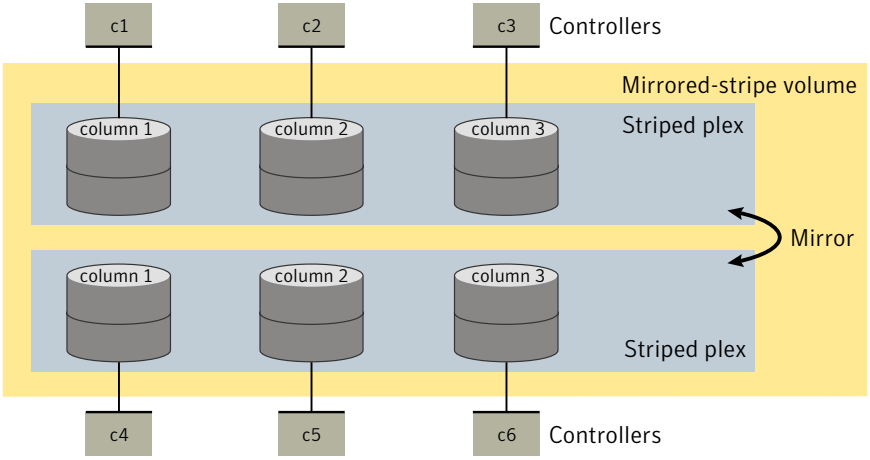
Other storage specification classes for controllers, enclosures, targets and trays can be used with ordered allocation. For example, the following command creates a 3-column mirrored-stripe volume between specified controllers:

```
# vxassist -b -g mydg -o ordered make mirstr2vol 80g \
  layout=mirror-stripe ncol=3 \
  ctlr:c1 ctlr:c2 ctlr:c3 ctlr:c4 ctlr:c5 ctlr:c6
```

This command allocates space for column 1 from disks on controllers `c1`, for column 2 from disks on controller `c2`, and so on.

Figure 7-4 shows an example of using storage allocation to create a mirrored-stripe volume across controllers.

Figure 7-4 Example of storage allocation used to create a mirrored-stripe volume across controllers



There are other ways in which you can control how `vxassist` lays out mirrored volumes across controllers.

See "Mirroring across targets, controllers or enclosures" on page 318.

Creating a mirrored volume

A mirrored volume provides data redundancy by containing more than one copy of its data. Each copy (or mirror) is stored on different disks from the original copy of the volume and from other mirrors. Mirroring a volume ensures that its data is not lost if a disk in one of its component mirrors fails.

A mirrored volume requires space to be available on at least as many disks in the disk group as the number of mirrors in the volume.

To create a new mirrored volume, use the following command:

```
# vxassist [-b] [-g diskgroup] make volume length \
  layout=mirror [nmirror=number] [init=active]
```

Specify the `-b` option if you want to make the volume immediately available for use.

See "Initializing and starting a volume" on page 324.

For example, to create the mirrored volume, `volmir`, in the disk group, `mydg`, use the following command:

```
# vxassist -b -g mydg make volmir 5g layout=mirror
```

To create a volume with 3 instead of the default of 2 mirrors, modify the command to read:

```
# vxassist -b -g mydg make volmir 5g layout=mirror nmirror=3
```

Creating a mirrored-concatenated volume

A mirrored-concatenated volume mirrors several concatenated plexes. To create a concatenated-mirror volume, use the following command:

```
# vxassist [-b] [-g diskgroup] make volume length \
  layout=mirror-concat [nmirror=number]
```

Specify the `-b` option if you want to make the volume immediately available for use.

See "Initializing and starting a volume" on page 324.

Alternatively, first create a concatenated volume, and then mirror it.

See "Adding a mirror to a volume " on page 336.

Creating a concatenated-mirror volume

A concatenated-mirror volume is an example of a layered volume which concatenates several underlying mirror volumes. To create a concatenated-mirror volume, use the following command:

```
# vxassist [-b] [-g diskgroup] make volume length \
  layout=concat-mirror [nmirror=number]
```

Specify the `-b` option if you want to make the volume immediately available for use.

See "Initializing and starting a volume" on page 324.

Creating a volume with a version 0 DCO volume

If a data change object (DCO) and DCO volume are associated with a volume, this allows Persistent FastResync to be used with the volume.

See "How persistent FastResync works with snapshots" on page 72.

The version 0 data change object (DCO) and DCO volume layout was introduced in VxVM 3.2. The version 0 layout supports traditional (third-mirror) snapshots,

but not full-sized instant snapshots, space-optimized instant snapshots nor DRL configured within the DCO volume.

See "Determining the DCO version number" on page 344.

See "Version 0 DCO volume layout" on page 70.

See "Version 20 DCO volume layout" on page 71.

See "Creating a volume with a version 20 DCO volume" on page 315.

To perform fast resynchronization of mirrors after a system crash or reboot, you must also enable dirty region logging (DRL) on a mirrored volume.

See "Adding a version 0 DCO and DCO volume" on page 429.

Note: You need a license to use the Persistent FastResync feature. If you do not have a license, you can configure a DCO object and DCO volume so that snap objects are associated with the original and snapshot volumes. However, without a license, only full resynchronization can be performed.

See "How persistent FastResync works with snapshots" on page 72.

To create a volume with an attached version 0 DCO object and volume

1 Ensure that the disk group has been upgraded to version 90. Use the following command to check the version of a disk group:

```
# vxdg list diskgroup
```

To upgrade a disk group to version 90, use the following command:

```
# vxdg -T 90 upgrade diskgroup
```

See "Upgrading a disk group" on page 264.

2 Use the following command to create the volume (you may need to specify additional attributes to create a volume with the desired characteristics):

```
# vxassist [-g diskgroup] make volume length layout=layout \
  logtype=dco [ndcomirror=number] [dcolen=size] \
  [fastresync=on] [other attributes]
```

For non-layered volumes, the default number of plexes in the mirrored DCO volume is equal to the lesser of the number of plexes in the data volume or 2. For layered volumes, the default number of DCO plexes is always 2. If required, use the `ndcomirror` attribute to specify a different number. It is recommended that you configure as many DCO plexes as there are data plexes in the volume. For example, specify `ndcomirror=3` when creating a 3-way mirrored volume.

The default size of each plex is 132 blocks unless you use the `dcolen` attribute to specify a different size. If specified, the size of the plex must be a multiple of 33 blocks from 33 up to a maximum of 2112 blocks.

By default, FastResync is not enabled on newly created volumes. Specify the `fastresync=on` attribute if you want to enable FastResync on the volume. If a DCO object and DCO volume are associated with the volume, Persistent FastResync is enabled; otherwise, Non-Persistent FastResync is enabled.

3 To enable DRL or sequential DRL logging on the newly created volume, use the following command:

```
# vxvol [-g diskgroup] set logtype=drl|drlseq volume
```

If you use ordered allocation when creating a mirrored volume on specified storage, you can use the optional `logdisk` attribute to specify on which disks dedicated log plexes should be created. Use the following form of the `vxassist` command to specify the disks from which space for the logs is to be allocated:

```
# vxassist [-g diskgroup] -o ordered make volume length \
  layout=mirror logtype=log_type logdisk=disk[,disk,...] \
  storage_attributes
```

If you do not specify the `logdisk` attribute, `vxassist` locates the logs in the data plexes of the volume.

See Specifying ordered allocation of storage to volumes.

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

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

Creating a volume with a version 20 DCO volume

To create a volume with an attached version 20 DCO object and volume

1. Ensure that the disk group has been upgraded to the latest version. Use the following command to check the version of a disk group:

   ```
   # vxdg list diskgroup
   ```

   To upgrade a disk group to the most recent version, use the following command:

   ```
   # vxdg upgrade diskgroup
   ```

   See "Upgrading a disk group" on page 264.

2. Use the following command to create the volume (you may need to specify additional attributes to create a volume with the desired characteristics):

   ```
   # vxassist [-g diskgroup] make volume length layout=layout \
     logtype=dco dcoversion=20 [drl=on|sequential|off] \
     [ndcomirror=number] [fastresync=on] [other attributes]
   ```

   Set the value of the `drl` attribute to `on` if dirty region logging (DRL) is to be used with the volume (this is the default setting). For a volume that will be written to sequentially, such as a database log volume, set the value to `sequential` to enable sequential DRL. The DRL logs are created in the DCO volume. The redundancy of the logs is determined by the number of mirrors that you specify using the `ndcomirror` attribute.

   By default, Persistent FastResync is not enabled on newly created volumes. Specify the `fastresync=on` attribute if you want to enable Persistent FastResync on the volume.

   See "Determining the DCO version number" on page 344.

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

Creating a volume with dirty region logging enabled

Dirty region logging (DRL), if enabled, speeds recovery of mirrored volumes after a system crash. To enable DRL on a volume that is created within a disk group with a version number between 20 and 100, specify the `logtype=drl` attribute to the `vxassist make` command as shown in this example usage:

```
# vxassist [-g diskgroup] make volume length layout=layout \
  logtype=drl [nlog=n] [loglen=size] [other attributes]
```

The `nlog` attribute can be used to specify the number of log plexes to add. By default, one log plex is added. The `loglen` attribute specifies the size of the log, where each bit represents one region in the volume. For example, the size of the log would need to be 20K for a 10GB volume with a region size of 64 kilobytes.

For example, to create a mirrored 10GB volume, `vol02`, with two log plexes in the disk group, `mydg`, use the following command:

```
# vxassist -g mydg make vol02 10g layout=mirror logtype=drl \
  nlog=2 nmirror=2
```

Sequential DRL limits the number of dirty regions for volumes that are written to sequentially, such as database replay logs. To enable sequential DRL on a volume that is created within a disk group with a version number between 70 and 100, specify the `logtype=drlseq` attribute to the `vxassist make` command.

```
# vxassist [-g diskgroup] make volume length layout=layout \
  logtype=drlseq [nlog=n] [other attributes]
```

It is also possible to enable the use of Persistent FastResync with this volume.

See "Creating a volume with a version 0 DCO volume" on page 311.

Note: Operations on traditional DRL log plexes are usually applicable to volumes that are created in disk groups with a version number of less than 110. If you enable DRL or sequential DRL on a volume that is created within a disk group with a version number of 110 or greater, the DRL logs are usually created within the plexes of a version 20 DCO volume.

See "Creating a volume with a version 20 DCO volume" on page 315.

Creating a striped volume

A striped volume contains at least one plex that consists of two or more subdisks located on two or more physical disks. A striped volume requires space to be available on at least as many disks in the disk group as the number of columns in the volume.

See "Striping (RAID-0)" on page 42.

To create a striped volume, use the following command:

```
# vxassist [-b] [-g diskgroup] make volume length layout=stripe
```

Specify the `-b` option if you want to make the volume immediately available for use.

See "Initializing and starting a volume" on page 324.

For example, to create the 10-gigabyte striped volume `volzebra`, in the disk group, `mydg`, use the following command:

```
# vxassist -b -g mydg make volzebra 10g layout=stripe
```

This creates a striped volume with the default stripe unit size (64 kilobytes) and the default number of stripes (2).

You can specify the disks on which the volumes are to be created by including the disk names on the command line. For example, to create a 30-gigabyte striped volume on three specific disks, `mydg03`, `mydg04`, and `mydg05`, use the following command:

```
# vxassist -b -g mydg make stripevol 30g layout=stripe \
  mydg03 mydg04 mydg05
```

To change the number of columns or the stripe width, use the `ncolumn` and `stripeunit` modifiers with `vxassist`. For example, the following command creates a striped volume with 5 columns and a 32-kilobyte stripe size:

```
# vxassist -b -g mydg make stripevol 30g layout=stripe \
  stripeunit=32k ncol=5
```

Creating a mirrored-stripe volume

A mirrored-stripe volume mirrors several striped data plexes. A mirrored-stripe volume requires space to be available on at least as many disks in the disk group as the number of mirrors multiplied by the number of columns in the volume.

To create a striped-mirror volume, use the following command:

```
# vxassist [-b] [-g diskgroup] make volume length \
  layout=mirror-stripe [nmirror=number_of_mirrors] \
  [ncol=number_of_columns] [stripewidth=size]
```

Specify the `-b` option if you want to make the volume immediately available for use.

See "Initializing and starting a volume" on page 324.

Alternatively, first create a striped volume, and then mirror it. In this case, the additional data plexes may be either striped or concatenated.

See "Adding a mirror to a volume " on page 336.

Creating a striped-mirror volume

A striped-mirror volume is an example of a layered volume which stripes several underlying mirror volumes. A striped-mirror volume requires space to be available on at least as many disks in the disk group as the number of columns multiplied by the number of stripes in the volume.

To create a striped-mirror volume, use the following command:

```
# vxassist [-b] [-g diskgroup] make volume length \
  layout=stripe-mirror [nmirror=number_of_mirrors] \
  [ncol=number_of_columns] [stripewidth=size]
```

Specify the `-b` option if you want to make the volume immediately available for use.

See "Initializing and starting a volume" on page 324.

By default, VxVM attempts to create the underlying volumes by mirroring subdisks rather than columns if the size of each column is greater than the value for the attribute `stripe-mirror-col-split-trigger-pt` that is defined in the `vxassist` defaults file.

If there are multiple subdisks per column, you can choose to mirror each subdisk individually instead of each column. To mirror at the subdisk level, specify the layout as `stripe-mirror-sd` rather than `stripe-mirror`. To mirror at the column level, specify the layout as `stripe-mirror-col` rather than `stripe-mirror`.

Mirroring across targets, controllers or enclosures

To create a volume whose mirrored data plexes lie on different controllers (also known as disk duplexing) or in different enclosures, use the `vxassist` command as described in this section.

In the following command, the attribute `mirror=target` specifies that volumes should be mirrored between targets on different controllers.

```
# vxassist [-b] [-g diskgroup] make volume length \
  layout=layout mirror=target [attributes]
```

Specify the `-b` option if you want to make the volume immediately available for use.

See "Initializing and starting a volume" on page 324.

The attribute `mirror=ctlr` specifies that disks in one mirror should not be on the same controller as disks in other mirrors within the same volume:

```
# vxassist [-b] [-g diskgroup] make volume length \
  layout=layout mirror=ctlr [attributes]
```

Note: Both paths of an active/passive array are not considered to be on different controllers when mirroring across controllers.

The following command creates a mirrored volume with two data plexes in the disk group, `mydg`:

```
# vxassist -b -g mydg make volspec 10g layout=mirror nmirror=2 \
  mirror=ctlr ctlr:c2 ctlr:c3
```

The disks in one data plex are all attached to controller `c2`, and the disks in the other data plex are all attached to controller `c3`. This arrangement ensures continued availability of the volume should either controller fail.

The attribute `mirror=enclr` specifies that disks in one mirror should not be in the same enclosure as disks in other mirrors within the same volume.

The following command creates a mirrored volume with two data plexes:

```
# vxassist -b make -g mydg volspec 10g layout=mirror nmirror=2 \
  mirror=enclr enclr:enc1 enclr:enc2
```

The disks in one data plex are all taken from enclosure `enc1`, and the disks in the other data plex are all taken from enclosure `enc2`. This arrangement ensures continued availability of the volume should either enclosure become unavailable.

There are other ways in which you can control how volumes are laid out on the specified storage.

See "Specifying ordered allocation of storage to volumes" on page 307.

Creating a RAID-5 volume

A RAID-5 volume requires space to be available on at least as many disks in the disk group as the number of columns in the volume. Additional disks may be required for any RAID-5 logs that are created.

Note: VxVM supports the creation of RAID-5 volumes in private disk groups, but not in shareable disk groups in a cluster environment.

You can create RAID-5 volumes by using either the `vxassist` command (recommended) or the `vxmake` command. Both approaches are described below.

A RAID-5 volume contains a RAID-5 data plex that consists of three or more subdisks located on three or more physical disks. Only one RAID-5 data plex can exist per volume. A RAID-5 volume can also contain one or more RAID-5 log plexes, which are used to log information about data and parity being written to the volume.

See "RAID-5 (striping with parity)" on page 49.

Warning: Do not create a RAID-5 volume with more than 8 columns because the volume will be unrecoverable in the event of the failure of more than one disk.

To create a RAID-5 volume, use the following command:

```
# vxassist [-b] [-g diskgroup] make volume length layout=raid5 \
  [ncol=number_of_columns] [stripewidth=size] [nlog=number] \
  [loglen=log_length]
```

Specify the `-b` option if you want to make the volume immediately available for use.

See "Initializing and starting a volume" on page 324.

For example, to create the RAID-5 volume `volraid` together with 2 RAID-5 logs in the disk group, `mydg`, use the following command:

```
# vxassist -b -g mydg make volraid 10g layout=raid5 nlog=2
```

This creates a RAID-5 volume with the default stripe unit size on the default number of disks. It also creates two RAID-5 logs rather than the default of one log.

If you require RAID-5 logs, you must use the `logdisk` attribute to specify the disks to be used for the log plexes.

RAID-5 logs can be concatenated or striped plexes, and each RAID-5 log associated with a RAID-5 volume has a complete copy of the logging information for the volume. To support concurrent access to the RAID-5 array, the log should be several times the stripe size of the RAID-5 plex.

It is suggested that you configure a minimum of two RAID-5 log plexes for each RAID-5 volume. These log plexes should be located on different disks. Having two RAID-5 log plexes for each RAID-5 volume protects against the loss of logging information due to the failure of a single disk.

If you use ordered allocation when creating a RAID-5 volume on specified storage, you must use the `logdisk` attribute to specify on which disks the RAID-5 log plexes

should be created. Use the following form of the `vxassist` command to specify the disks from which space for the logs is to be allocated:

```
# vxassist [-b] [-g diskgroup] -o ordered make volume length \
  layout=raid5 [ncol=number_columns] [nlog=number] \
  [loglen=log_length] logdisk=disk[,disk,...] \
  storage_attributes
```

For example, the following command creates a 3-column RAID-5 volume with the default stripe unit size on disks `mydg04`, `mydg05` and `mydg06`. It also creates two RAID-5 logs on disks `mydg07` and `mydg08`.

```
# vxassist -b -g mydg -o ordered make volraid 10g layout=raid5 \
  ncol=3 nlog=2 logdisk=mydg07,mydg08 mydg04 mydg05 mydg06
```

The number of logs must equal the number of disks that is specified to `logdisk`.

See "Specifying ordered allocation of storage to volumes" on page 307.

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

It is possible to add more logs to a RAID-5 volume at a later time.

See "Adding a RAID-5 log" on page 351.

Creating tagged volumes

Volume tags are used to implement the Dynamic Storage Tiering (DST) feature of the Storage Foundation software.

See the *Veritas File System Administrator's Guide*.

You can use the `tag` attribute with the `vxassist make` command to set a named tag and optional tag value on a volume, for example:

```
# vxassist -b -g mydg make volmir 5g layout=mirror tag=mirvol=5g
```

To list the tags that are associated with a volume, use this command:

```
# vxassist [-g diskgroup] listtag volume
```

To list the volumes that have a specified tag name, use this command:

```
# vxassist [-g diskgroup] list tag=tagname volume
```

Tag names and tag values are case-sensitive character strings of up to 256 characters. Tag names can consist of letters (A through Z and a through z), numbers (0 through 9), dashes (-), underscores (_) or periods (.) from the ASCII character set. A tag name must start with either a letter or an underscore. Tag

values can consist of any character from the ASCII character set with a decimal value from 32 through 127. If a tag value includes any spaces, use the `vxassist settag` command to set the tag on the newly created volume.

Dotted tag hierarchies are understood by the `list` operation. For example, the listing for `tag=a.b` includes all volumes that have tag names that start with `a.b`.

The tag names `site`, `udid` and `vdid` are reserved and should not be used. To avoid possible clashes with future product features, it is recommended that tag names do not start with any of the following strings: `asl`, `be`, `isp`, `nbu`, `sf`, `symc` or `vx`.

See Setting tags on volumes.

Creating a volume using vxmake

As an alternative to using `vxassist`, you can create a volume using the `vxmake` command to arrange existing subdisks into plexes, and then to form these plexes into a volume.

See “Creating subdisks” on page 272.

The example given in this section is to create a RAID-5 volume using `vxmake`.

Creating a RAID-5 plex for a RAID-5 volume is similar to creating striped plexes, except that the `layout` attribute is set to `raid5`. Subdisks can be implicitly associated in the same way as with striped plexes. For example, to create a four-column RAID-5 plex with a stripe unit size of 32 sectors, use the following command:

```
# vxmake -g mydg plex raidplex layout=raid5 stwidth=32 \
  sd=mydg00-01,mydg01-00,mydg02-00,mydg03-00
```

Note that because four subdisks are specified, but the number of columns is not specified, the `vxmake` command assumes a four-column RAID-5 plex and places one subdisk in each column. Striped plexes are created using the same method except that the layout is specified as `stripe`. If the subdisks are to be created and added later, use the following command to create the plex:

```
# vxmake -g mydg plex raidplex layout=raid5 ncolumn=4 stwidth=32
```

If no subdisks are specified, the `ncolumn` attribute must be specified. Subdisks can be added to the plex later using the `vxsd assoc` command.

See “Associating subdisks with plexes” on page 275.

If each column in a RAID-5 plex is to be created from multiple subdisks which may span several physical disks, you can specify to which column each subdisk

should be added. For example, to create a three-column RAID-5 plex using six subdisks, use the following form of the vxmake command:

```
# vxmake -g mydg plex raidplex layout=raid5 stwidth=32 \
  sd=mydg00-00:0,mydg01-00:1,mydg02-00:2,mydg03-00:0, \
  mydg04-00:1,mydg05-00:2
```

This command stacks subdisks mydg00-00 and mydg03-00 consecutively in column 0, subdisks mydg01-00 and mydg04-00 consecutively in column 1, and subdisks mydg02-00 and mydg05-00 in column 2. Offsets can also be specified to create sparse RAID-5 plexes, as for striped plexes.

Log plexes may be created as default concatenated plexes by not specifying a layout, for example:

```
# vxmake -g mydg plex raidlog1 sd=mydg06-00
  # vxmake -g mydg plex raidlog2 sd=mydg07-00
```

The following command creates a RAID-5 volume, and associates the prepared RAID-5 plex and RAID-5 log plexes with it:

```
# vxmake -g mydg -Uraid5 vol raidvol \
  plex=raidplex,raidlog1,raidlog2
```

Each RAID-5 volume has one RAID-5 plex where the data and parity are stored. Any other plexes associated with the volume are used as RAID-5 log plexes to log information about data and parity being written to the volume.

After creating a volume using vxmake, you must initialize it before it can be used.

See "Initializing and starting a volume" on page 324.

Creating a volume using a vxmake description file

You can use the vxmake command to add a new volume, plex or subdisk to the set of objects managed by VxVM. vxmake adds a record for each new object to the VxVM configuration database. You can create records either by specifying parameters to vxmake on the command line, or by using a file which contains plain-text descriptions of the objects. The file can also contain commands for performing a list of tasks. Use the following form of the command to have vxmake read the file from the standard input:

```
# vxmake [-g diskgroup] < description_file
```

Alternatively, you can specify the file to vxmake using the -d option:

```
# vxmake [-g diskgroup] -d description_file
```

The following sample description file defines a volume, db, with two plexes, db-01 and db-02:

```
#rty #name       #options
sd   mydg03-01   disk=mydg03 offset=0 len=10000
sd   mydg03-02   disk=mydg03 offset=25000 len=10480
sd   mydg04-01   disk=mydg04 offset=0 len=8000
sd   mydg04-02   disk=mydg04 offset=15000 len=8000
sd   mydg04-03   disk=mydg04 offset=30000 len=4480
plex db-01       layout=STRIPE ncolumn=2 stwidth=16k
                 sd=mydg03-01:0/0,mydg03-02:0/10000,mydg04-01:1/0,
mydg04-02:1/8000,mydg04-03:1/16000
sd   ramd1-01    disk=ramd1 len=640
                 comment="Hot spot for dbvol"
plex db-02       sd=ramd1-01:40320
vol  db          usetype=gen plex=db-01,db-02
                 readpol=prefer prefname=db-02
                 comment="Uses mem1 for hot spot in last 5m"
```

The subdisk definition for plex, db-01, must be specified on a single line. It is shown here split across two lines because of space constraints.

The first plex, db-01, is striped and has five subdisks on two physical disks, mydg03 and mydg04. The second plex, db-02, is the preferred plex in the mirror, and has one subdisk, ramd1-01, on a volatile memory disk.

For detailed information about how to use vxmake, refer to the vxmake(1M) manual page.

After creating a volume using vxmake, you must initialize it before it can be used.

See "Initializing and starting a volume created using vxmake" on page 325.

Initializing and starting a volume

If you create a volume using the vxassist command, vxassist initializes and starts the volume automatically unless you specify the attribute init=none.

When creating a volume, you can make it immediately available for use by specifying the -b option to the vxassist command, as shown here:

```
# vxassist -b [-g diskgroup] make volume length layout=mirror
```

The -b option makes VxVM carry out any required initialization as a background task. It also greatly speeds up the creation of striped volumes by initializing the columns in parallel.

As an alternative to the -b option, you can specify the init=active attribute to make a new volume immediately available for use. In this example, init=active is specified to prevent VxVM from synchronizing the empty data plexes of a new mirrored volume:

```
# vxassist [-g diskgroup] make volume length layout=mirror \
  init=active
```

Warning: There is a very small risk of errors occurring when the init=active attribute is used. Although written blocks are guaranteed to be consistent, read errors can arise in the unlikely event that fsck attempts to verify uninitialized space in the file system, or if a file remains uninitialized following a system crash. If in doubt, use the -b option to vxassist instead.

This command writes zeroes to the entire length of the volume and to any log plexes. It then makes the volume active. You can also zero out a volume by specifying the attribute init=zero to vxassist, as shown in this example:

```
# vxassist [-g diskgroup] make volume length layout=raid5 \
  init=zero
```

You cannot use the -b option to make this operation a background task.

Initializing and starting a volume created using vxmake

A volume may be initialized by running the vxvol command if the volume was created by the vxmake command and has not yet been initialized, or if the volume has been set to an uninitialized state.

To initialize and start a volume, use the following command:

```
# vxvol [-g diskgroup] start volume
```

The following command can be used to enable a volume without initializing it:

```
# vxvol [-g diskgroup] init enable volume
```

This allows you to restore data on the volume from a backup before using the following command to make the volume fully active:

```
# vxvol [-g diskgroup] init active volume
```

If you want to zero out the contents of an entire volume, use this command to initialize it:

```
# vxvol [-g diskgroup] init zero volume
```

Accessing a volume

As soon as a volume has been created and initialized, it is available for use as a virtual disk partition by the operating system for the creation of a file system, or by application programs such as relational databases and other data management software.

Creating a volume in a disk group sets up block and character (raw) device files that can be used to access the volume:

`/dev/vx/dsk/dg/vol`	block device file for volume *vol* in disk group *dg*
`/dev/vx/rdsk/dg/vol`	character device file for volume *vol* in disk group *dg*

The pathnames include a directory named for the disk group. Use the appropriate device node to create, mount and repair file systems, and to lay out databases that require raw partitions.

As the `rootdg` disk group no longer has special significance, VxVM only creates volume device nodes for this disk group in the `/dev/vx/dsk/rootdg` and `/dev/vx/rdsk/rootdg` directories. VxVM does not create device nodes in the `/dev/vx/dsk` or `/dev/vx/rdsk` directories for the `rootdg` disk group.

Chapter 8

Administering volumes

This chapter includes the following topics:

- About volume administration
- Displaying volume information
- Monitoring and controlling tasks
- Stopping a volume
- Starting a volume
- Adding a mirror to a volume
- Removing a mirror
- Adding logs and maps to volumes
- Preparing a volume for DRL and instant snapshots
- Upgrading existing volumes to use version 20 DCOs
- Adding traditional DRL logging to a mirrored volume
- Adding a RAID-5 log
- Resizing a volume
- Setting tags on volumes
- Changing the read policy for mirrored volumes
- Removing a volume
- Moving volumes from a VM disk
- Enabling FastResync on a volume

- Performing online relayout
- Converting between layered and non-layered volumes
- Using Thin Provisioning

About volume administration

Veritas Volume Manager (VxVM) allows you to perform common maintenance tasks on volumes. 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.

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.

See the *Veritas Storage Foundation Intelligent Storage Provisioning Administrator's Guide*.

Note: Most VxVM commands require superuser or equivalent privileges.

Displaying volume information

You can use the `vxprint` command to display information about how a volume is configured.

To display the volume, plex, and subdisk record information for all volumes in the system, use the following command:

```
# vxprint -hvt
```

The `vxprint` command can also be applied to a single disk group:

```
# vxprint -g mydg -hvt
```

This is example output from this command:

```
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.

The headings for sub-volumes (`SV`), storage caches (`SC`), data change objects (`DCO`) and snappoints (`SP`) can be ignored in the sample output. No such objects are associated with the volumes that are shown.

To display volume-related information for a specific volume, use the following command:

```
# vxprint [-g diskgroup] -t volume
```

For example, to display information about the volume, `voldef`, in the disk group, `mydg`, use the following command:

```
# vxprint -g mydg -t voldef
```

This is example output from this command:

```
V   NAME       RVG/VSET/CO KSTATE    STATE     LENGTH     READPOL      PREFPLEX    UTYPE

v   voldef     -           ENABLED   ACTIVE    20480      SELECT       -           fsgen
```

If you enable enclosure-based naming, and use the `vxprint` command to display the structure of a volume, it shows enclosure-based names for the disk devices rather than OS-based names.

The output from the `vxprint` command includes information about the volume state.

See "Volume states" on page 330.

Volume states

Table 8-1 shows the volume states that may be displayed by VxVM commands such as `vxprint`.

Table 8-1 Volume states

Volume state	Description
ACTIVE	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	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	The volume contents are not initialized. The kernel state is always DISABLED when the volume is EMPTY.
INVALID	The contents of an instant snapshot volume no longer represent a true point-in-time image of the original volume.
NEEDSYNC	The volume requires a resynchronization operation the next time it is started. For a RAID-5 volume, a parity resynchronization operation is required.
REPLAY	The volume is in a transient state as part of a log replay. A log replay occurs when it becomes necessary to use logged parity and data. This state is only applied to RAID-5 volumes.

Table 8-1 Volume states *(continued)*

Volume state	Description
SYNC	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).

The interpretation of these states during volume startup is modified by the persistent state log for the volume (for example, the DIRTY/CLEAN flag). If the clean flag is set, an ACTIVE volume was not written to by any processes or was not even open at the time of the reboot; therefore, it can be considered CLEAN. 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.

No user intervention is required to set these states; they are maintained internally. On a system that is operating properly, all volumes are ENABLED.

Table 8-2 shows the volume kernel states that can be defined.

Table 8-2 Volume kernel state

Volume kernel state	Description
DETACHED	Maintenance is being performed on the volume. The volume cannot be read from or written to, but certain plex operations and `ioctl` function calls are accepted.
DISABLED	The volume is offline and cannot be accessed.
ENABLED	The volume is online and can be read from or written to.

Monitoring and controlling tasks

The VxVM task monitor tracks the progress of system recovery by monitoring task creation, maintenance, and completion. The task monitor 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).

Note: VxVM supports this feature for private disk groups, but not for shareable disk groups in a cluster environment.

Specifying task tags

Every task is given a unique task identifier. This is a numeric identifier for the task that can be specified to the `vxtask` utility to specifically identify a single task. Several VxVM utilities also provide a `-t` option to specify an alphanumeric tag of up to 16 characters in length. This allows you to group several tasks by associating them with the same tag.

The `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

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.

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.

New tasks take time to be set up, and so may not be immediately available for use after a command is invoked. Any script that operates on tasks may need to poll for the existence of a new task.

See the vxtask(1M) manual page.

vxtask operations

The vxtask command supports the following operations:

abort	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.)

Using the vxtask command

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, `mydg`, that are currently paused, as well as any tasks with the tag `sysstart`, use the following command:

```
# vxtask -g mydg -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 example, the `vxtask` command can be used to monitor and modify the progress of the Online Relayout feature.

See "Controlling the progress of a relayout" on page 369.

Stopping a volume

Stopping a volume renders it unavailable to the user, and changes the volume kernel state from ENABLED or DETACHED to DISABLED. If the volume cannot be disabled, it remains in its current state. To stop a volume, use the following command:

```
# vxvol [-g diskgroup] [-f] stop volume ...
```

To stop all volumes in a specified disk group, use the following command:

```
# vxvol [-g diskgroup] [-f] stopall
```

Warning: If you use the `-f` option to forcibly disable a volume that is currently open to an application, the volume remains open, but its contents are inaccessible. I/O operations on the volume fail, and this may cause data loss. 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.

See "Starting a volume" on page 335.

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
```

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.

You can use this task on your root disk to make a second copy of the boot information available on an alternate disk. This allows you to boot your system even if your root disk fails.

Note: This task only mirrors concatenated volumes. Volumes are ignored if they are already mirrored or if they contain subdisks that reside on multiple disks.

To mirror volumes on a disk

1. Make sure that the target disk has an equal or greater amount of space as the originating disk.
2. Select `Mirror volumes on a disk` from the `vxdiskadm` main menu.
3. At the following prompt, enter the disk name of the disk that you wish to mirror:

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

4. At the following prompt, enter the target disk name (this disk must be the same size or larger than the originating disk):

   ```
   Enter destination disk [<disk>,list,q,?] (default: any) mydg01
   ```

5 At the following prompt, press Return to make the mirror:

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

The `vxdiskadm` program displays the status of the mirroring operation, as follows:

```
VxVM vxmirror INFO V-5-2-22 Mirror volume voltest-bk00
   .
   .
   .
VxVM INFO V-5-2-674 Mirroring of disk mydg01 is complete.
```

6 At the 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)
```

Additional mirroring considerations

The larger private region size that was introduced in VxVM 3.2 (1MB) and VxVM 5.0 (32MB) may create one of the following mirroring scenarios under which `vxdiskadm` fails:

- Mirroring a full root disk to a target disk which is exactly the same size as the source disk. A full disk means a disk that has no free cylinders.
- Mirroring a disk created using an earlier version of Veritas Volume Manager to a target disk which is exactly the same size as the source disk. You only need to use this step if mirroring using `vxdiskadm` fails.
- Mirroring a full Veritas Volume Manager disk (not a root disk) that was encapsulated in VxVM 3.5 to a target disk that is the same size as the source disk. You only need to use this step if mirroring using `vxdiskadm` fails.

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

To create a mirror under any of these scenarios

1 Determine the size of the source disk's private region, using either of the following methods:

- If the source disk is a root disk, obtain its private region length by running this command:

  ```
  # vxprint -l rootdisk
  ```

  The disk media name of the root disk is typically `rootdisk`.

In the output, find the `privlen` value. In this example, the value is 3071:

```
devinfo: publen=39846240 privlen=3071
```

- If the source disk is not a root disk, obtain its private region length by running this command:

  ```
  # vxdisk list diskname
  ```

  where *diskname* is the disk media name of the source disk.
  In the displayed output, note the `len` value for the `private` field. In this example, the value of this field is 3071:

  ```
  private: slice=4 offset=1 len=3071
  ```

2 Use the `vxdisksetup` program to initialize the target disk:

```
# /usr/lib/vxvm/bin/vxdisksetup -i c#t#d# privoffset=0 \
  privlen=XXXX publen=YYYY
```

where *XXXX* is the size of the source disk's private region, and *YYYY* is the size of its public region.

If your system is configured to use enclosure-based naming instead of OS-based naming, replace the `c#t#d#` name with the enclosure-based name for the disk.

3 Add the newly initialized target disk to the source disk group:

```
# vxdg -g diskgroup adddisk medianame=c#t#d#
```

4 Use the `vxdiskadm` command and select `Mirror volumes on a disk` to create the mirror. Specify the disk media names of the source disk (*rootdisk*) and the target disk (*medianame*).

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
```

See "Creating a volume on specific disks" on page 305.

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 and maps to volumes

Veritas Volume Manager supports the following types of volume logs and maps:

- FastResync Maps are used to perform quick and efficient resynchronization of mirrors. These maps are supported either in memory (Non-Persistent FastResync), or on disk as part of a DCO volume (Persistent FastResync).
  See "FastResync" on page 68.
  See "Enabling FastResync on a volume" on page 361.

  Two types of DCO volumes are supported:

  - Version 0 DCO volumes only support Persistent FastResync for the traditional third-mirror break-off type of volume snapshot.
    See "Version 0 DCO volume layout" on page 70.
    See "Adding a version 0 DCO and DCO volume" on page 429.
  - Version 20 DCO volumes, introduced in VxVM 4.0, support DRL logging (see below) and Persistent FastResync for full-sized and space-optimized instant volume snapshots.
    See "Version 20 DCO volume layout" on page 71.
    See "Preparing a volume for DRL and instant snapshots" on page 341.

- Dirty Region Logs allow the fast recovery of mirrored volumes after a system crash. These logs are supported either as DRL log plexes, or as part of a version 20 DCO volume.
  See "Dirty region logging" on page 62.
  See "Adding traditional DRL logging to a mirrored volume" on page 349.
  See "Preparing a volume for DRL and instant snapshots" on page 341.
- RAID-5 logs are used to prevent corruption of data during recovery of RAID-5 volumes. These logs are configured as plexes on disks other than those that are used for the columns of the RAID-5 volume.
  See "RAID-5 logging" on page 54.
  See "Adding a RAID-5 log" on page 351.

Preparing a volume for DRL and instant snapshots

You can add a version 20 data change object (DCO) and DCO volume to a volume that you previously created in a disk group with a version number of 110 or greater. If you are creating a new volume in a disk group with a version number of 110 or greater, you can create the DCO and DCO volume and enable DRL at the same time as you create the volume.

See "Determining the DCO version number" on page 344.

See "Creating a volume with a version 20 DCO volume" on page 315.

See "Upgrading existing volumes to use version 20 DCOs" on page 347.

Note: You need a license to use the DRL and FastResync feature. If you do not have a license, you can configure a DCO object and DCO volume so that snap objects are associated with the original and snapshot volumes. However, without a license, only full resynchronization can be performed.

See "How persistent FastResync works with snapshots" on page 72.

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 and snapshot plexes in the volume. The DCO plexes are used to set up a DCO volume for any snapshot volume that you subsequently create from the

snapshot plexes. For example, specify `ndcomirs=5` for a volume with 3 data plexes and 2 snapshot plexes.

The value of the `regionsize` attribute specifies the size of the tracked regions in the volume. A write to a region is tracked by setting a bit in the change map. The default value is `64k` (64KB). A smaller value requires more disk space for the change maps, but the finer granularity provides faster resynchronization.

To enable DRL logging on the volume, specify `drl=on` (this is the default 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 or cannot be used for the plexes of the DCO volume.

See "Specifying storage for version 20 DCO plexes" on page 342.

The `vxsnap prepare` command automatically enables Persistent FastResync on the volume. Persistent FastResync is also set automatically on any snapshots that are generated from a volume on which this feature is enabled.

If the volume is a RAID-5 volume, it is converted to a layered volume that can be used with instant snapshots and Persistent FastResync.

See "Using a DCO and DCO volume with a RAID-5 volume" on page 343.

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.

Specifying storage for version 20 DCO plexes

If the disks that contain volumes and their snapshots are to be moved into different disk groups, you must ensure that the disks that contain their DCO plexes can accompany them. You can use storage attributes to specify which disks to use for the DCO plexes. (If you do not want to use dirty region logging (DRL) with a volume, you can specify the same disks as those on which the volume is configured, assuming that space is available on the disks). For example, to add a DCO object and mirrored DCO volume with plexes on `disk05` and `disk06` to the volume, `myvol`, use the following command:

```
# vxsnap -g mydg prepare myvol ndcomirs=2 alloc=disk05,disk06
```

To view the details of the DCO object and DCO volume that are associated with a volume, use the `vxprint` command. The following is example `vxprint -vh` output for the volume named `vol1` (the TUTIL0 and PUTIL0 columns are omitted for clarity):

```
TY    NAME         ASSOC       KSTATE   LENGTH   PLOFFS  STATE       ...
v     vol1         fsgen       ENABLED  1024     -       ACTIVE
pl    vol1-01      vol1        ENABLED  1024     -       ACTIVE
sd    disk01-01    vol1-01     ENABLED  1024     0       -
pl    foo-02       vol1        ENABLED  1024     -       ACTIVE
sd    disk02-01    vol1-02     ENABLED  1024     0       -
dc    vol1_dco     vol1        -        -        -       -
v     vol1_dcl     gen         ENABLED  132      -       ACTIVE
pl    vol1_dcl-01  vol1_dcl    ENABLED  132      -       ACTIVE
sd    disk03-01    vol1_dcl-01 ENABLED  132      0       -
pl    vol1_dcl-02  vol1_dcl    ENABLED  132      -       ACTIVE
sd    disk04-01    vol1_dcl-02 ENABLED  132      0       -
```

In this output, the DCO object is shown as `vol1_dco`, and the DCO volume as `vol1_dcl` with 2 plexes, `vol1_dcl-01` and `vol1_dcl-02`.

If required, you can use the `vxassist move` command to relocate DCO plexes to different disks. For example, the following command moves the plexes of the DCO volume, `vol1_dcl`, for volume `vol1` from `disk03` and `disk04` to `disk07` and `disk08`:

```
# vxassist -g mydg move vol1_dcl !disk03 !disk04 disk07 disk08
```

See "Moving DCO volumes between disk groups" on page 255.

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

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

Using a DCO and DCO volume with a RAID-5 volume

The procedure in the previous section can be used to add a DCO and DCO volume to a RAID-5 volume. This allows you to use Persistent FastResync on the volume for fast resynchronization of snapshots on returning them to their original volume. However, the procedure has the side effect of converting the RAID-5 volume into a special type of layered volume. You can create space-optimized instant snapshots of such a volume, and you can add mirrors that may be broken off as full-sized instant snapshots. You cannot relayout or resize such a volume unless you convert it back to a pure RAID-5 volume.

To convert a volume back to a RAID-5 volume, remove any snapshot plexes from the volume, and dissociate the DCO and DCO volume from the layered volume. You can then perform relayout and resize operations on the resulting non-layered RAID-5 volume.

See "Removing support for DRL and instant snapshots from a volume" on page 346.

To allow Persistent FastResync to be used with the RAID-5 volume again, re-associate the DCO and DCO volume.

See "Preparing a volume for DRL and instant snapshots" on page 341.

Warning: Dissociating a DCO and DCO volume disables FastResync on the volume. A full resynchronization of any remaining snapshots is required when they are snapped back.

Determining the DCO version number

The instant snapshot and DRL-enabled DCO features require that a version 20 DCO be associated with a volume, rather than an earlier version 0 DCO.

To find out the version number of a DCO that is associated with a volume

1. Use the `vxprint` command on the volume to discover the name of its DCO:

```
# DCONAME=`vxprint [-g diskgroup] -F%dco_name volume`
```

2. Use the `vxprint` command on the DCO to determine its version number:

```
# vxprint [-g diskgroup] -F%version $DCONAME
```

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.

   If the number of active mirrors in the volume is less than 2, DRL logging is not performed even if DRL is enabled on the volume.

   See "Determining if DRL logging is active on a volume" on page 346.

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

```
# 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 and instant snapshots from a volume

To remove support for DRL and instant snapshot operation from a volume, use the following command to remove the DCO and DCO volume that are associated with the volume:

```
#  vxsnap [-g diskgroup] unprepare volume
```

This command also has the effect of disabling FastResync tracking on the volume.

Note: This command fails if the volume is part of a snapshot hierarchy.

Upgrading existing volumes to use version 20 DCOs

The procedure described in this section describes how to upgrade a volume created before VxVM 4.0 so that it can take advantage of new features such as instant snapshots, and DRL logs that are configured within the DCO volume. This requires upgrading the version of the disk groups, removing any existing snapshots and version 0 DCOs that are associated with volumes in the disk groups, and finally configuring the volumes with version 20 DCOs.

Note: The plexes of the DCO volume require persistent storage space on disk to be available. To make room for the DCO plexes, you may need to add extra disks to the disk group, or reconfigure existing volumes to free up space in the disk group. Another way to add disk space is to use the disk group move feature to bring in spare disks from a different disk group.

See "Reorganizing the contents of disk groups" on page 250.

The `vxsnap prepare` command automatically enables FastResync on the volume and on any snapshots that are generated from it.

If the volume is a RAID-5 volume, it is converted to a layered volume that can be used with snapshots and FastResync.

To upgrade an existing disk group and the volumes that it contains

1. Upgrade the disk group that contains the volume to the latest version before performing the remainder of the procedure described in this section. Use the following command to check the version of a disk group:

   ```
   # vxdg list diskgroup
   ```

   To upgrade a disk group to the latest version, use the following command:

   ```
   # vxdg upgrade diskgroup
   ```

   See "Upgrading a disk group" on page 264.

2. Use the following command to discover which volumes in the disk group have version 0 DCOs associated with them:

   ```
   # vxprint [-g diskgroup] -F "%name" -e "v_hasdcolog"
   ```

   This command assumes that the volumes can only have version 0 DCOs as the disk group has just been upgraded.

   See "Determining the DCO version number" on page 344.

   Repeat the following steps to upgrade each volume within the disk group as required.

3. If the volume to be upgraded has a traditional DRL plex or subdisk (that is, the DRL logs are not held in a version 20 DCO volume), use the following command to remove this:

   ```
   # vxassist [-g diskgroup] remove log volume [nlog=n]
   ```

   Use the optional attribute `nlog=n` to specify the number, `n`, of logs to be removed. By default, the `vxassist` command removes one log.

4. For a volume that has one or more associated snapshot volumes, use the following command to reattach and resynchronize each snapshot:

   ```
   # vxassist [-g diskgroup] snapback snapvol
   ```

   If FastResync was enabled on the volume before the snapshot was taken, the data in the snapshot plexes is quickly resynchronized from the original volume. If FastResync was not enabled, a full resynchronization is performed.

5. Use the following command to turn off FastResync for the volume:

   ```
   # vxvol [-g diskgroup] set fastresync=off volume
   ```

6 Use the following command to dissociate a version 0 DCO object, DCO volume and snap objects from the volume:

```
# vxassist [-g diskgroup] remove log volume logtype=dco
```

7 Use the following command on the volume to upgrade it:

```
# 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 and snapshot plexes in the volume. The DCO plexes are used to set up a DCO volume for any snapshot volume that you subsequently create from the snapshot plexes. For example, specify `ndcomirs=5` for a volume with 3 data plexes and 2 snapshot plexes.

The value of the `regionsize` attribute specifies the size of the tracked regions in the volume. A write to a region is tracked by setting a bit in the change map. The default value is `64k` (64KB). A smaller value requires more disk space for the change maps, but the finer granularity provides faster resynchronization.

To enable DRL logging on the volume, specify `drl=on` (this is the default 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 or cannot be used for the plexes of the DCO volume.

Adding traditional DRL logging to a mirrored volume

A traditional DRL log is configured within a DRL plex. However, a version 20 DCO volume cannot be used in conjunction with a DRL plex. The version 20 DCO volume layout includes space for a DRL log. The new DCO volume layout also supports traditional (third-mirror), instant (copy-on-write), and instant space-optimized snapshots.

See "Preparing a volume for DRL and instant snapshots" on page 341.

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] [loglen=size]
```

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. The `loglen` attribute specifies the size of the log, where each bit represents one region in the volume. For example, the size of the log would need to be 20K for a 10GB volume with a region size of 64 kilobytes.

For example, to 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 if this is required.

See "Removing a traditional DRL log" on page 350.

Removing a traditional DRL log

The `vxassist remove log` command can be used to remove a traditional DRL log that is configured within a DRL plex, but not to remove a DRL log that is configured within a version 20 DCO.

To remove a traditional DRL log

- Type the following command:

  ```
  # vxassist [-g diskgroup] remove log volume logtype=drl [nlog=n]
  ```

  By default, the `vxassist` command removes one log. Use the optional attribute `nlog=`*n* to specify the number of logs that are to remain after the operation completes.

  You can use storage attributes to specify the storage from which a log is to be removed. For example, to remove a log on disk `mydg10` from volume vol01, enter:

  ```
  # vxassist -g mydg remove log vol01 !mydg10 logtype=drl
  ```

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]
```

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.

Note: You cannot VxVM commands or the VEA to resize a volume or any underlying file system on an encapsulated `root` disk. This is because the underlying disk partitions also need to be reconfigured. IIf you really need to resize the volumes on the root disk, you must first unencasulate the root disk.

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.

Warning: 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.

Table 8-3 shows what operations are permitted, and whether you must first unmount the file system to resize the file system.

Table 8-3 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`	Increases volume size to a specified length.
`growby`	Increases volume size by a specified amount.
`shrinkto`	Reduces volumesize to a specified length.
`shrinkby`	Reduces volume size by a specified amount.

Warning: You cannot grow or shrink any volume associated with an encapsulated root disk (`rootvol`, `usr`, `var`, `opt`, `swapvol`, and so on) because these map to a physical underlying partition on the disk and must be contiguous. If you attempt to grow `rootvol`, `usrvol`, `varvol`, or `swapvol`, the system could become unbootable if you need to revert back to booting from slices. It can also prevent a successful Solaris upgrade and you might have to do a fresh install. Additionally, the `upgrade_start` script might fail.

Extending to a given length

To extend a volume to a specific length, use the following command:

```
# vxassist [-b] [-g diskgroup] growto volume length
```

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
```

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
```

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
```

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
```

Warning: Do not shrink the volume below the current size of the file system or database using the volume. The `vxassist shrinkto` command can be used safely on empty volumes.

Shrinking by a given length

To shrink a volume by a specific length, use the following command:

```
# vxassist [-g diskgroup] shrinkby volume length
```

For example, to shrink `volcat` by 300 sectors, use the following command:

```
# vxassist -g mydg shrinkby volcat 300
```

Warning: Do not shrink the volume below the current size of the file system or database using the volume. 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 be used to 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
```

Warning: 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).

Setting tags on volumes

Volume tags are used to implement the Dynamic Storage Tiering feature of the Storage Foundation software.

See the *Veritas File System Administrator's Guide*.

You can use the following forms of the `vxassist` command to set a named tag and optional tag value on a volume, to replace a tag, and to remove a tag from a volume:

```
# vxassist [-g diskgroup] settag volume tagname[=tagvalue]
# vxassist [-g diskgroup] replacetag volume oldtag newtag
# vxassist [-g diskgroup] removetag volume tagname
```

To list the tags that are associated with a volume, use this command:

```
# vxassist [-g diskgroup] listtag volume
```

To list the volumes that have a specified tag name, use this command:

```
# vxassist [-g diskgroup] list tag=tagname volume
```

Tag names and tag values are case-sensitive character strings of up to 256 characters. Tag names can consist of letters (A through Z and a through z),

numbers (0 through 9), dashes (-), underscores (_) or periods (.) from the ASCII character set. A tag name must start with either a letter or an underscore. Tag values can consist of any character from the ASCII character set with a decimal value from 32 through 127. If a tag value includes any spaces, quote the specification to protect it from the shell, as shown here:

```
# vxassist -g mydg settag myvol "dbvol=table space 1"
```

Dotted tag hierarchies are understood by the `list` operation. For example, the listing for `tag=a.b` includes all volumes that have tag names that start with `a.b`.

The tag names `site`, `udid` and `vdid` are reserved and should not be used. To avoid possible clashes with future product features, it is recommended that tag names do not start with any of the following strings: `asl`, `be`, `isp`, `nbu`, `sf`, `symc` or `vx`.

Changing the read policy for mirrored volumes

VxVM offers the choice of the following read policies on the data plexes in a mirrored volume:

`round`	Reads each plex in turn in "round-robin" fashion for each nonsequential I/O detected. Sequential access causes only one plex to be accessed. This 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.
`siteread`	Reads preferentially from plexes at the locally defined site. This is the default policy for volumes in disk groups where site consistency has been enabled. See "Configuring site consistency on a disk group" on page 518.

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
```

See "Volume read policies" on page 547.

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.

To remove a volume

1. Remove all references to the volume by application programs, including shells, that are running on the system.
2. If the volume is mounted as a file system, unmount it with 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
```

5 Remove the volume using the `vxassist` command:

```
# vxassist [-g diskgroup] remove volume volume
```

Alternatively, you can use the `vxedit` command to remove the 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 that have sufficient space.

To move volumes from a disk

1 Select `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:

```
Enter disk name [<disk>,list,q,?] mydg01
```

You can now optionally specify a list of disks to which the volume(s) should be moved. At the prompt, press **Enter** 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)
```

Enabling FastResync on a volume

The recommended method for enabling FastResync on a volume with a version 20 DCO is to use the `vxsnap prepare` command.

See "Preparing a volume for DRL and instant snapshots" on page 341.

Note: You need a FastResync license to use this feature.

FastResync performs quick and efficient resynchronization of stale mirrors. It also increases the efficiency of the VxVM snapshot mechanism when used with operations such as backup and decision support.

See "About volume snapshots" on page 373.

See "FastResync" on page 68.

There are two possible versions of FastResync that can be enabled on a volume:

- Persistent FastResync holds copies of the FastResync maps on disk. These can be used for the speedy recovery of mirrored volumes if a system is rebooted. This form of FastResync requires that either a version 0 or a version 20 data change object (DCO) and DCO volume first be associated with the volume.
  See "Adding a version 0 DCO and DCO volume" on page 429.
  See "Upgrading existing volumes to use version 20 DCOs" on page 347.
  See "Preparing a volume for DRL and instant snapshots" on page 341.
- Non-Persistent FastResync holds the FastResync maps in memory. These do not survive on a system that is rebooted.

By default, FastResync is not enabled on newly created volumes. Specify the `fastresync=on` attribute to the `vxassist make` command if you want to enable FastResync on a volume that you are creating.

Note: It is not possible to configure both Persistent and Non-Persistent FastResync on a volume. Persistent FastResync is used if a DCO is associated with the volume. Otherwise, Non-Persistent FastResync is used.

To turn FastResync on for an existing volume, specify `fastresync=on` to the `vxvol` command as shown here:

```
# vxvol [-g diskgroup] set fastresync=on volume
```

To use FastResync with a snapshot, FastResync must be enabled before the snapshot is taken, and must remain enabled until after the snapback is completed.

Checking whether FastResync is enabled on a volume

To check whether FastResync is enabled on a volume, use the following command:

```
# vxprint [-g diskgroup] -F%fastresync volume
```

This command returns `on` if FastResync is enabled; otherwise, it returns `off`.

If FastResync is enabled, to check whether it is Non-Persistent or Persistent FastResync, use the following command:

```
# vxprint [-g diskgroup] -F%hasdcolog volume
```

This command returns `on` if Persistent FastResync is enabled; otherwise, it returns `off`.

To list all volumes on which Non-Persistent FastResync is enabled, use the following command:

```
# vxprint [-g diskgroup] -F "%name" \
  -e "v_fastresync=on && !v_hasdcolog"
```

To list all volumes on which Persistent FastResync is enabled, use the following command:

```
# vxprint [-g diskgroup] -F "%name" -e "v_fastresync=on \
  && v_hasdcolog"
```

Disabling FastResync

Use the `vxvol` command to turn off Persistent or Non-Persistent FastResync for an existing volume, as shown here:

```
# vxvol [-g diskgroup] set fastresync=off volume
```

Turning FastResync off releases all tracking maps for the specified volume. All subsequent reattaches will not use the FastResync facility, but perform a full resynchronization of the volume. This occurs even if FastResync is later turned on.

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]
```

If specified, the `-b` option makes relayout of the volume a background task.

The following destination layout configurations are supported.

`concat-mirror`	concatenated-mirror
`concat`	concatenated
`nomirror`	concatenated

`nostripe`	concatenated
`raid5`	RAID-5 (not supported for shared disk groups)
`span`	concatenated
`stripe`	striped

See "Permitted relayout transformations" on page 364.

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 368.

Permitted relayout transformations

Table 8-4 shows the supported relayout transformations for concatenated volumes.

Table 8-4 Supported relayout transformations for concatenated volumes

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

Table 8-5 shows the supported relayout transformations for concatenated-mirror volumes.

Table 8-5 Supported relayout transformations for 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.

Table 8-6 shows the supported relayout transformations for RAID-5 volumes.

Table 8-6 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`.

Table 8-7 shows the supported relayout transformations for mirror-concatenated volumes.

Table 8-7 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.

Table 8-8 shows the supported relayout transformations for mirrored-stripe volumes.

Table 8-8 Supported relayout transformations for mirrored-stripe volumes

Relayout to	From mirror-stripe
concat	Yes.
concat-mirror	Yes.
mirror-concat	No. Use `vxassist convert` after relayout to 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`.

Table 8-9 shows the supported relayout transformations for unmirrored stripe and layered striped-mirror volumes.

Table 8-9 Supported relayout transformations for unmirrored stripe and layered striped-mirror volumes

Relayout to	From stripe or stripe-mirror
concat	Yes.
concat-mirror	Yes.
mirror-concat	No. Use `vxassist convert` after relayout to 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. The following are examples of using these options:

`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.

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 "Viewing the status of a relayout" on page 368.

See "Controlling the progress of a relayout" on page 369.

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.

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
```

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 "Managing tasks with vxtask" on page 332.

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

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

Converting between layered and non-layered volumes

The `vxassist convert` command transforms volume layouts between layered and non-layered forms:

```
# vxassist [-b] [-g diskgroup] convert volume [layout=layout] \
  [convert_options]
```

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.

Using Thin Provisioning

This section describes how to use Veritas Volume Manager (VxVM) volumes with Thin Storage LUNs.

About Thin Provisioning

Thin Provisioning is a storage array feature that enables increased storage utilization by automating the task of storage provisioning. Instead of administrators having to estimate how much storage an application requires, Thin Provisioning allows administrators to provision a large Thin LUN to a host, and leave application I/O writes to drive actual physical capacity allocation on the Thin LUN.

About Thin Reclamation

When data is deleted on a host, nothing happens on a Thin Storage LUN. As a result, a host writing 200 GB of data on a file system that is supported by a Thin LUN triggers allocation of at least 200 GB of physical disk space behind that Thin LUN. When that host subsequently deletes 200 GB of data from that same file system, the amount of physical storage behind the Thin LUN does not change. In such a case, there are still at least 200 GB of physical disk space behind the Thin LUN, now lying unused.

The Storage Foundation Thin Reclamation feature leverages Veritas File System (VxFS) knowledge of used and unused block ranges at the file system level, and allows an administrator to trigger online reclamation of unused Thin Storage in storage arrays that support Thin Reclamation. Thin Reclamation can be triggered on a file system, a set of disks, a disk group, or an enclosure, and releases unused Thin Storage to the free storage pool in the array. The Storage Foundation Hardware Compatibility Lists contain Thin Reclamation support statements for the storage arrays that support this feature.

Thin Reclamation of a disk, a disk group, or an enclosure

The `vxdisk reclaim` command can be used to trigger online Thin Reclamation on one or more disks, disk groups, or enclosures. The following example triggers reclamation on disks `disk1` and `disk2`:

```
# vxdisk reclaim disk1 disk2
```

Thin Reclamation can only be performed on `thin_rclm` disks. Device discovery automatically discovers disks that support Thin Reclamation from capable storage arrays. Use `vxdisk -o thin list` command to list devices that are known to be thin or `thin_rclm` on a host.

Thin Reclamation can only be performed if the disks support a VxFS file system that is online.

See the *Veritas File System Administrator's Guide* for more information on how to trigger Thin Reclamation on a VxFS file system.

Chapter 9

Administering volume snapshots

This chapter includes the following topics:

- About volume snapshots
- Traditional third-mirror break-off snapshots
- Full-sized instant snapshots
- Space-optimized instant snapshots
- Emulation of third-mirror break-off snapshots
- Linked break-off snapshot volumes
- Cascaded snapshots
- Creating multiple snapshots
- Restoring the original volume from a snapshot
- Creating instant snapshots
- Creating traditional third-mirror break-off snapshots
- Adding a version 0 DCO and DCO volume

About volume snapshots

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.

See "Volume snapshots" on page 65.

You can also take a snapshot of a volume set.

See "Creating instant snapshots of volume sets" on page 405.

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.

See "Traditional third-mirror break-off snapshots" on page 375.

See "Full-sized instant snapshots" on page 376.

See "Space-optimized instant snapshots" on page 378.

See "Emulation of third-mirror break-off snapshots" on page 379.

See "Linked break-off snapshot volumes" on page 380.

See "Cascaded snapshots" on page 381.

See "Creating multiple snapshots" on page 385.

See "Restoring the original volume from a snapshot" on page 386.

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.

There are two alternative methods of creating volume snapshots.

See "Creating instant snapshots" on page 387.

See "Creating traditional third-mirror break-off snapshots" on page 420.

Volume snapshots can be used to implement off-host online backup.

See "About off-host processing solutions" on page 443.

Snapshot creation using the `vxsnap` command is the preferred mechanism for implementing online and off-host point-in-time copy solutions in VxVM. Support for traditional third-mirror snapshots that are created using the `vxassist` command may be removed in a future release.

Note: Most VxVM commands require superuser or equivalent privileges.

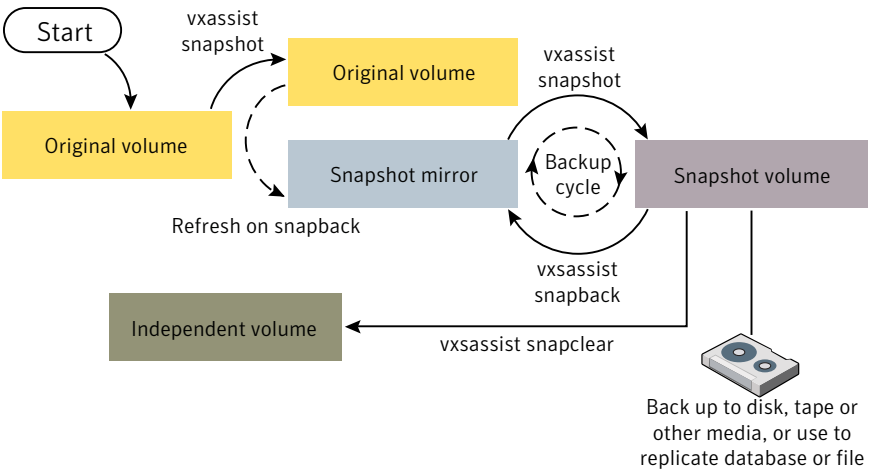
There are several ways of recovering from the failure of instant snapshot commands.

See the *Veritas Volume Manager Troubleshooting Guide*.

Traditional third-mirror break-off snapshots

Figure 9-1 shows the traditional third-mirror break-off volume snapshot model that is supported by the `vxassist` command.

Figure 9-1 Third-mirror snapshot creation and usage



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

As is usual when creating a mirror, the process of copying the volume's contents to the new snapshot plexes can take some time to complete. Full-sized and space-optimized instant snapshots can be used to make snapshots immediately available.

See "Full-sized instant snapshots" on page 376.

See "Space-optimized instant snapshots" on page 378.

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 386.

You can use the FastResync feature of VxVM to minimize the time needed to resynchronize the data in the snapshot mirror. If FastResync is not enabled, a full resynchronization of the data is required.

See "FastResync" on page 68.

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.

The use of the `vxassist` command to administer traditional (third-mirror break-off) snapshots is not supported for volumes that are prepared for instant snapshot creation. The `vxsnap` command may be used instead.

See "Full-sized instant snapshots" on page 376.

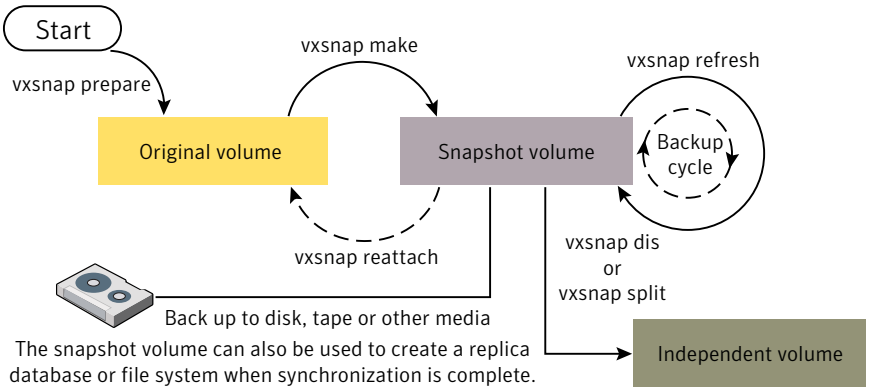
See "Creating traditional third-mirror break-off snapshots" on page 420.

Full-sized instant snapshots

Full-sized instant snapshots are a variation on the third-mirror volume snapshot model that make a snapshot volume available for access as soon as the snapshot plexes have been created.

Figure 9-2 shows the full-sized instant volume snapshot model.

Figure 9-2 Full-sized instant snapshot creation and usage in a backup cycle



To create an instant snapshot, you use the `vxsnap make` command. This command can either be applied to a suitably prepared empty volume that is to be used as the snapshot volume, or it can be used to break off one or more synchronized plexes from the original volume (which is similar to the way that the `vxassist` command creates its snapshots).

Unlike a third-mirror break-off snapshot created using the `vxassist` command, you can make a backup of a full-sized instant snapshot, instantly refresh its contents from the original volume, or attach its plexes to the original volume, without needing to completely synchronize the snapshot plexes from the original volume.

VxVM uses a copy-on-write mechanism to ensure that the snapshot volume preserves the contents of the original volume at the time that the snapshot is taken. Any time that the original contents of the volume are about to be overwritten, the original data in the volume is preserved on the snapshot volume before the write proceeds. As time goes by, and the contents of the volume are updated, its original contents are gradually relocated to the snapshot volume.

If desired, you can additionally select to perform either a background (non-blocking) or foreground (blocking) synchronization of the snapshot volume. This is useful if you intend to move the snapshot volume into a separate disk group for off-host processing, or you want to use the `vxsnap dis` or `vxsnap split` commands to turn the snapshot volume into an independent volume.

The `vxsnap refresh` command allows you to update the data in a snapshot each time that you make a backup.

The command, `vxsnap reattach`, can be used to attach snapshot plexes to the original volume, and to resynchronize the data in these plexes from the original volume. Alternatively, you can use the `vxsnap restore` command to restore the

contents of the original volume from a snapshot that you took at an earlier point in time. You can also choose whether or not to keep the snapshot volume after restoration of the original volume is complete.

See "Restoring the original volume from a snapshot" on page 386.

By default, the FastResync feature of VxVM is used to minimize the time needed to resynchronize the data in the snapshot mirror. If FastResync is not enabled, a full resynchronization of the data is required.

See "FastResync" on page 68.

See "Creating and managing full-sized instant snapshots" on page 397.

An empty volume must be prepared for use by full-sized instant snapshots and linked break-off snapshots.

See "Creating a volume for use as a full-sized instant or linked break-off snapshot" on page 393.

Space-optimized instant snapshots

Volume snapshots require the creation of a complete copy of the original volume, and use as much storage space as the original volume.

Instead of requiring a complete copy of the original volume's storage space, space-optimized instant snapshots use a storage cache. The size of this cache may be configured when the snapshot is created.

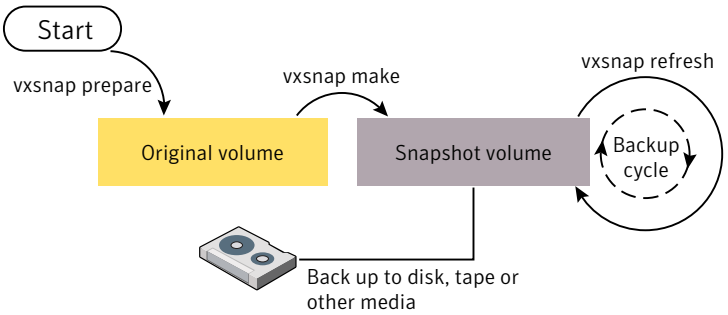
A storage cache may be named and shared among several volumes in the same disk group. If so, the size of the cache that is declared must be the same for each volume's space-optimized snapshot. You may find it convenient to configure a single storage cache in a disk group that can be shared by all the volumes in that disk group.

See "Creating a shared cache object" on page 391.

When the original volume is written to, VxVM preserves the original data contents in the cache before the write is committed. As the storage cache can be configured to require much less storage than the original volume, it is referred to as being space-optimized. If the cache becomes too full, you can configure VxVM to grow the size of the cache automatically using any available free space in the disk group.

Figure 9-3 shows the instant space-optimized snapshot model.

Figure 9-3 Space-optimized instant snapshot creation and usage in a backup cycle



As for instant snapshots, space-optimized snapshots use a copy-on-write mechanism to make them immediately available for use when they are first created, or when their data is refreshed. Unlike instant snapshots, however, you cannot enable synchronization on space-optimized snapshots, reattach them to their original volume, or turn them into independent volumes.

See "Creating and managing space-optimized instant snapshots" on page 394.

A cache object and cache volume must be set up for use by space-optimized instant snapshots.

See "Creating a shared cache object" on page 391.

Emulation of third-mirror break-off snapshots

Third-mirror break-off snapshots are suitable for write-intensive volumes (such as for database redo logs) where the copy-on-write mechanism of space-optimized or full-sized instant snapshots might degrade the performance of the volume.

If you use the `vxsnap prepare` command to enable a volume for use with instant and space-optimized snapshots, you cannot use the `vxassist` snapshot commands to administer snapshots that you create for the volume. If you require snapshots that behave as third-mirror break-off snapshots (that is, they must be fully synchronized before they can be used), there are three ways to achieve this:

- Use the `vxsnap addmir` command to create and attach one or more snapshot mirrors to the volume. When the plexes have been synchronized and are in the `SNAPDONE` state, the `vxsnap make` command can then be used with the `nmirror` attribute to create the snapshot volume. This technique is similar to using the `vxassist snapstart` and `vxassist snapshot` commands.
  See "Traditional third-mirror break-off snapshots" on page 375.

- Use the `vxsnap make` command with the `plex` attribute to use one or more existing plexes of a volume as snapshot plexes. The volume must have a sufficient number of available plexes that are in the `ACTIVE` state.
  The volume must be a non-layered volume with a `mirror` or `mirror-stripe` layout, or a RAID-5 volume that you have converted to a special layered volume and then mirrored.
  See "Using a DCO and DCO volume with a RAID-5 volume" on page 343.
  The plexes in a volume with a `stripe-mirror` layout are mirrored at the sub-volume level, and cannot be broken off.
- Use the `vxsnap make` command with the `sync=yes` and `type=full` attributes specified to create the snapshot volume, and then use the `vxsnap syncwait` command to wait for synchronization of the snapshot volume to complete.

See "Adding snapshot mirrors to a volume" on page 407.

See "Creating and managing third-mirror break-off snapshots" on page 399.

Linked break-off snapshot volumes

A variant of the third-mirror break-off snapshot type are linked break-off snapshot volumes, which use the `vxsnap addmir` command to link a specially prepared volume with the data volume. The volume that is used for the snapshot is prepared in the same way as for full-sized instant snapshots. However, unlike full-sized instant snapshots, this volume can be set up in a different disk group from the data volume. This makes linked break-off snapshots especially suitable for off-host processing applications where you may want to create the snapshot on storage with different characteristics from that used for the data volumes. As for third-mirror break-off snapshots, you must wait for the contents of the snapshot volume to be synchronized with the data volume before you can use the `vxsnap make` command to take the snapshot.

When a link is created between a volume and the mirror that will become the snapshot, separate link objects (similar to snap objects) are associated with the volume and with its mirror. The link object for the original volume points to the mirror volume, and the link object for the mirror volume points to the original volume. All I/O is directed to both the original volume and its mirror, and a synchronization of the mirror from the data in the original volume is started.

You can use the `vxprint` command to display the state of link objects, which appear as type `ln`. Link objects can have the following states:

ACTIVE	The mirror volume has been fully synchronized from the original volume. The `vxsnap make` command can be run to create a snapshot volume.
ATTACHING	Synchronization of the mirror volume is in progress. The `vxsnap make` command cannot be used to create a snapshot volume until the state changes to ACTIVE. The `vxsnap snapwait` command can be used to wait for the synchronization to complete.
BROKEN	The mirror volume has been detached from the original volume because of an I/O error or an unsuccessful attempt to grow the mirror volume. The vxrecover command can be used to recover the mirror volume in the same way as for a DISABLED volume. See "Starting a volume" on page 335.

If you resize (that is, grow or shrink) a volume, all its `ACTIVE` linked mirror volumes are also resized at the same time. The volume and its mirrors can be in the same disk group or in different disk groups. If the operation is successful, the volume and its mirrors will have the same size.

If a volume has been grown, a resynchronization of the grown regions in its linked mirror volumes is started, and the links remain in the `ATTACHING` state until resynchronization is complete. The `vxsnap snapwait` command can be used to wait for the state to become `ACTIVE`.

.When you use the `vxsnap make` command to create the snapshot volume, this removes the link, and establishes a snapshot relationship between the snapshot volume and the original volume.

The `vxsnap reattach` operation re-establishes the link relationship between the two volumes, and starts a resynchronization of the mirror volume.

See "Creating and managing linked break-off snapshot volumes" on page 402.

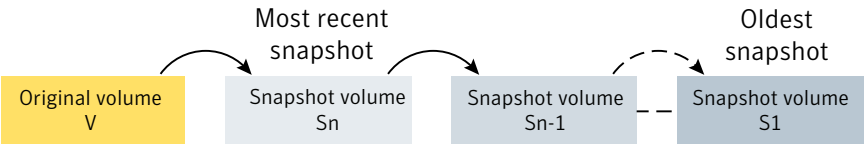
An empty volume must be prepared for use by full-sized instant and linked break-off snapshots.

See "Creating a volume for use as a full-sized instant or linked break-off snapshot" on page 393.

Cascaded snapshots

Figure 9-4 shows a snapshot hierarchy, known as a snapshot cascade, that can improve write performance for some applications.

Figure 9-4 Snapshot cascade



Instead of having several independent snapshots of the volume, it is more efficient to make the older snapshots into children of the latest snapshot.

A snapshot may be added to a cascade by specifying the `infrontof` attribute to the `vxsnap make` command when the second and subsequent snapshots in the cascade are created. Changes to blocks in the original volume are only written to the most recently created snapshot volume in the cascade. If an attempt is made to read data from an older snapshot that does not exist in that snapshot, it is obtained by searching recursively up the hierarchy of more recent snapshots.

A snapshot cascade is most likely to be used for regular online backup of a volume where space-optimized snapshots are written to disk but not to tape.

A snapshot cascade improves write performance over the alternative of several independent snapshots, and also requires less disk space if the snapshots are space-optimized. Only the latest snapshot needs to be updated when the original volume is updated. If and when required, the older snapshots can obtain the changed data from the most recent snapshot.

The following points determine whether it is appropriate for an application to use a snapshot cascade:

- Deletion of a snapshot in the cascade takes time to copy the snapshot's data to the next snapshot in the cascade.
- The reliability of a snapshot in the cascade depends on all the newer snapshots in the chain. Thus the oldest snapshot in the cascade is the most vulnerable.
- Reading from a snapshot in the cascade may require data to be fetched from one or more other snapshots in the cascade.

For these reasons, it is recommended that you do not attempt to use a snapshot cascade with applications that need to remove or split snapshots from the cascade. In such cases, it may be more appropriate to create a snapshot of a snapshot as described in the following section.

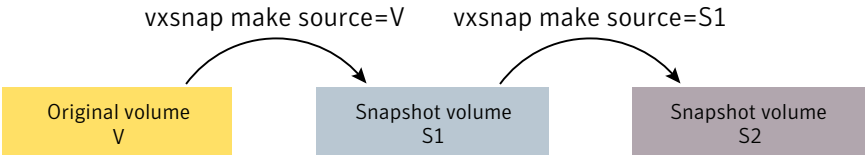
See "Adding a snapshot to a cascaded snapshot hierarchy" on page 408.

Note: Only unsynchronized full-sized or space-optimized instant snapshots are usually cascaded. It is of little utility to create cascaded snapshots if the `infrontof` snapshot volume is fully synchronized (as, for example, with break-off type snapshots).

Creating a snapshot of a snapshot

Figure 9-5 shows how it may be desirable to create a snapshot of an existing snapshot for some applications.

Figure 9-5 Creating a snapshot of a snapshot



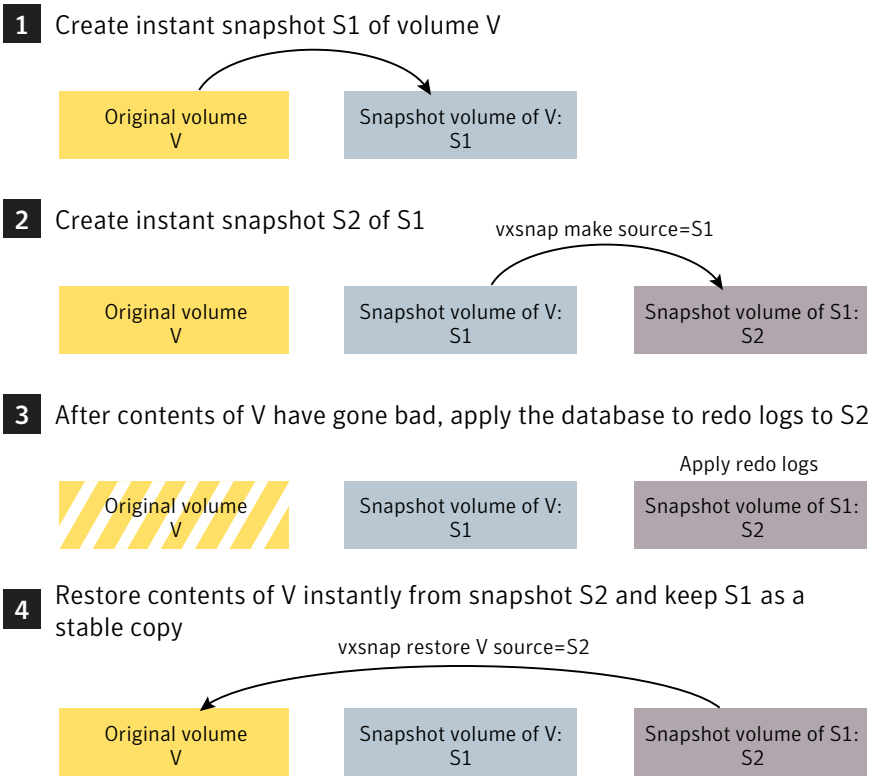
Even though the arrangement of the snapshots in this figure appears similar to a snapshot cascade, the relationship between the snapshots is not recursive. When reading from the snapshot `S2`, data is obtained directly from the original volume, `V`, if it does not exist in `S2` itself.

See Figure 9-4 on page 382.

Such an arrangement may be useful if the snapshot volume, `S1`, is critical to the operation. For example, `S1` could be used as a stable copy of the original volume, `V`. The additional snapshot volume, `S2`, can be used to restore the original volume if that volume becomes corrupted. For a database, you might need to replay a redo log on `S2` before you could use it to restore `V`.

Figure 9-6 shows the sequence of steps that would be required to restore a database.

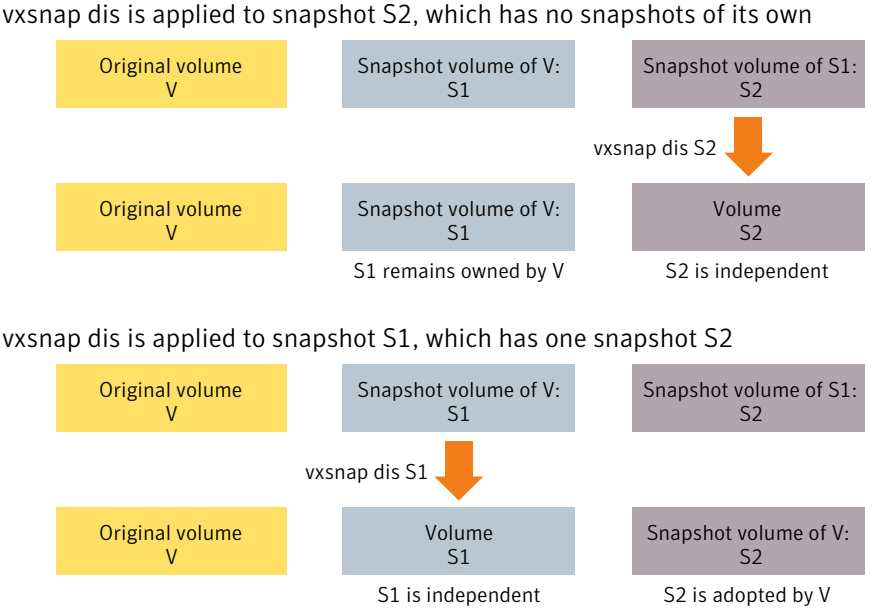
Figure 9-6 Using a snapshot of a snapshot to restore a database



If you have configured snapshots in this way, you may wish to make one or more of the snapshots into independent volumes. There are two `vxsnap` commands that you can use to do this:

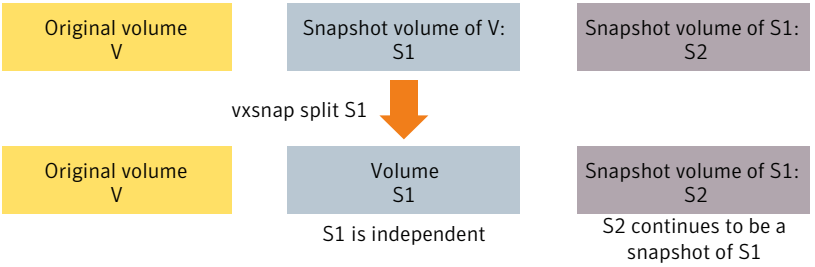
- `vxsnap dis` dissociates a snapshot volume and turns it into an independent volume. The volume to be dissociated must have been fully synchronized from its parent. If a snapshot volume has a child snapshot volume, the child must also have been fully synchronized. If the command succeeds, the child snapshot becomes a snapshot of the original volume.
  Figure 9-7 shows the effect of applying the `vxsnap dis` command to snapshots with and without dependent snapshots.

Figure 9-7 Dissociating a snapshot volume



- `vxsnap split` dissociates a snapshot and its dependent snapshots from its parent volume. The snapshot volume that is to be split must have been fully synchronized from its parent volume.
  Figure 9-8 shows the operation of the `vxsnap split` command.

Figure 9-8 Splitting snapshots



Creating multiple snapshots

To make it easier to create snapshots of several volumes at the same time, both the `vxsnap make` and `vxassist snapshot` commands accept more than one volume name as their argument.

For traditional snapshots, you can create snapshots of all the volumes in a single disk group by specifying the option `-o allvols` to the `vxassist snapshot` command.

By default, each replica volume is named *`SNAPnumber-volume`*, where number is a unique serial number, and *volume* is the name of the volume for which a snapshot is being taken. This default can be overridden by using the option *`-o name=pattern`*.

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

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

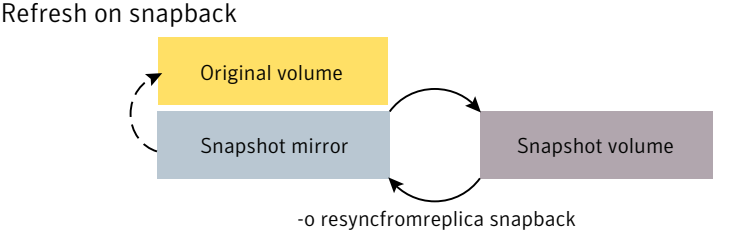
It is also possible to take several snapshots of the same volume. A new FastResync change map is produced for each snapshot taken to minimize the resynchronization time for each snapshot.

Restoring the original volume from a snapshot

For traditional snapshots, the snapshot plex is resynchronized from the data in the original volume during a `vxassist snapback` operation.

Figure 9-9 shows an alternative where you can choose the snapshot plex as the preferred copy of the data when performing a snapback.

Figure 9-9 Resynchronizing an original volume from a snapshot



Specifying the option `-o resyncfromreplica` to `vxassist` resynchronizes the original volume from the data in the snapshot.

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

For instant snapshots, the `vxsnap restore` command may be used to restore the contents of the original volume from an instant snapshot or from a volume derived from an instant snapshot. The volume that is used to restore the original volume can either be a true backup of the contents of the original volume at some point in time, or it may have been modified in some way (for example, by applying a database log replay or by running a file system checking utility such as `fsck`) to create a synthetic replica. All synchronization of the contents of this backup or synthetic replica volume must have been completed before the original volume can be restored from it. The original volume is immediately available for use while its contents are being restored.

You can perform either a destructive or non-destructive restoration of an original volume from an instant snapshot. Only non-destructive restoration is possible from a space-optimized snapshot. In this case, the snapshot remains in existence after the restoration is complete.

Creating instant snapshots

VxVM allows you to make instant snapshots of volumes by using the `vxsnap` command.

You can also take instant snapshots of RAID-5 volumes that have been converted to a special layered volume layout by the addition of a DCO and DCO volume.

See "Using a DCO and DCO volume with a RAID-5 volume" on page 343.

A plex in a full-sized instant snapshot requires as much space as the original volume. If you instead make a space-optimized instant snapshot of a volume, this only requires enough storage to record the original contents of the parent volume as they are changed during the life of the snapshot.

The recommended approach to performing volume backup from the command line, or from a script, is to use the `vxsnap` command. The `vxsnap prepare` and `make` tasks allow you to back up volumes online with minimal disruption to users.

The `vxsnap prepare` step creates a DCO and DCO volume and associates this with the volume. It also enables Persistent FastResync on the volume.

The `vxsnap make` step creates an instant snapshot that is immediately available for making a backup. After the snapshot has been taken, read requests for data in the original volume are satisfied by reading either from a non-updated region of the original volume, or from the copy of the original contents of an updated region that have been recorded by the snapshot.

Note: Synchronization of a full-sized instant snapshot from the original volume is enabled by default. If you specify the `syncing=no` attribute to `vxsnap make`, this disables synchronization, and the contents of the instant snapshot are unlikely ever to become fully synchronized with the contents of the original volume at the point in time that the snapshot was taken. If you wish to move an instant snapshot volume to another disk group for export to another machine for off-host processing, or to turn it into an independent volume, you must ensure that the snapshot volume has been completely synchronized.

You can immediately retake a full-sized or space-optimized instant snapshot at any time by using the `vxsnap refresh` command. If a fully synchronized instant snapshot is required, you must wait for the new resynchronization to complete.

You can create instant snapshots of volume sets by replacing volume names with volume set names in the `vxsnap` command.

See "Creating instant snapshots of volume sets" on page 405.

When using the `vxsnap prepare` or `vxassist make` commands to make a volume ready for instant snapshot operations, if the specified region size exceeds half the value of the tunable `voliomem_maxpool_sz` , the operation succeeds but gives a warning such as the following (for a system where `voliomem_maxpool_sz` is set to 12MB):

```
VxVM vxassist WARNING V-5-1-0 Specified regionsize is
larger than the limit on the system
(voliomem_maxpool_sz/2=6144k).
```

See "DMP tunable parameters " on page 565.

If this message is displayed, `vxsnap make`, `refresh` and `restore` operations on such volumes fail as they might potentially hang the system. Such volumes can be used only for break-off snapshot operations using the `reattach` and `make` operations.

To make the volumes usable for instant snapshot operations, use `vxsnap unprepare` on the volume, and then use `vxsnap prepare` to re-prepare the volume with a region size that is less than half the size of `voliomem_maxpool_sz` (in this example, 1MB):

```
# vxsnap -g mydg -f unprepare vol1
# vxsnap -g mydg prepare vol1 regionsize=1M
```

See "Preparing to create instant and break-off snapshots" on page 390.

See "Creating and managing space-optimized instant snapshots" on page 394.

See “Creating and managing full-sized instant snapshots” on page 397.

See “Creating and managing third-mirror break-off snapshots” on page 399.

See “Creating and managing linked break-off snapshot volumes” on page 402.

Preparing to create instant and break-off snapshots

To prepare a volume for the creation of instant and break-off snapshots

1 Use the following commands to see if the volume is associated with a version 20 data change object (DCO) and DCO volume that allow instant snapshots and Persistent FastResync to be used with the volume, and to check that FastResync is enabled on the volume:

```
# vxprint -g volumedg -F%instant volume
# vxprint -g volumedg -F%fastresync volume
```

If both commands return a value of `on`, the volume can be used for instant snapshot operations, and you should skip to step 3. Otherwise continue with step 2.

2 To prepare a volume for instant snapshots, use the following command:

```
# vxsnap [-g diskgroup] prepare volume [regionsize=size] \
  [ndcomirs=number] [alloc=storage_attributes]
```

It is only necessary to run the `vxsnap prepare` command on a volume if it does not already have a version 20 DCO volume (for example, if you have run the `vxsnap unprepare` command on the volume).

See "Creating a volume with a version 20 DCO volume" on page 315.

See "Preparing a volume for DRL and instant snapshots" on page 341.

See "Removing support for DRL and instant snapshots from a volume" on page 346.

For example, to prepare the volume, my`vol`, in the disk group, `mydg`, use the following command:

```
# vxsnap -g mydg prepare myvol regionsize=128k ndcomirs=2 \
  alloc=mydg10,mydg11
```

This example creates a DCO object and redundant DCO volume with two plexes located on disks `mydg10` and `mydg11`, and associates them with `myvol`. The region size is also increased to 128KB from the default size of 64KB. The region size must be a power of 2, and be greater than or equal to 16KB. A smaller value requires more disk space for the change maps, but the finer granularity provides faster resynchronization.

3 If you need to create several space-optimized instant snapshots for the volumes in a disk group, you may find it more convenient to create a single shared cache object in the disk group rather than a separate cache object for each snapshot.

See "Creating a shared cache object" on page 391.

For full-sized instant snapshots and linked break-off snapshots, you must prepare a volume that is to be used as the snapshot volume. This volume must be the same size as the data volume for which the snapshot is being created, and it must also have the same region size.

See "Creating a volume for use as a full-sized instant or linked break-off snapshot" on page 393.

Creating a shared cache object

To create a shared cache object

1 Decide on the following characteristics that you want to allocate to the cache volume that underlies the cache object:

- The size of the cache volume should be sufficient to record changes to the parent volumes during the interval between snapshot refreshes. A suggested value is 10% of the total size of the parent volumes for a refresh interval of 24 hours.
- If redundancy is a desired characteristic of the cache volume, it should be mirrored. This increases the space that is required for the cache volume in proportion to the number of mirrors that it has.
- If the cache volume is mirrored, space is required on at least as many disks as it has mirrors. These disks should not be shared with the disks used for the parent volumes. The disks should also be chosen to avoid impacting I/O performance for critical volumes, or hindering disk group split and join operations.

2 Having decided on its characteristics, use the `vxassist` command to create the volume that is to be used for the cache volume. The following example creates a mirrored cache volume, `cachevol`, with size 1GB in the disk group, `mydg`, on the disks `mydg16` and `mydg17`:

```
# vxassist -g mydg make cachevol 1g layout=mirror \
  init=active mydg16 mydg17
```

The attribute `init=active` is specified to make the cache volume immediately available for use.

3 Use the `vxmake cache` command to create a cache object on top of the cache volume that you created in the previous step:

```
# vxmake [-g diskgroup] cache cache_object \
  cachevolname=volume [regionsize=size] [autogrow=on] \
  [highwatermark=hwmk] [autogrowby=agbvalue] \
  [maxautogrow=maxagbvalue]]
```

If the region size, `regionsize`, is specified, it must be a power of 2, and be greater than or equal to 16KB (`16k`). If not specified, the region size of the cache is set to 64KB.

All space-optimized snapshots that share the cache must have a region size that is equal to or an integer multiple of the region size set on the cache. Snapshot creation also fails if the original volume's region size is smaller than the cache's region size.

If the region size of a space-optimized snapshot differs from the region size of the cache, this can degrade the system's performance compared to the case where the region sizes are the same.

If the cache is to be allowed to grow in size as required, specify `autogrow=on`. By default, the ability to automatically grow the cache is turned off.

In the following example, the cache object, `cobjmydg`, is created over the cache volume, `cachevol`, the region size of the cache is set to 32KB, and the `autogrow` feature is enabled:

```
# vxmake -g mydg cache cobjmydg cachevolname=cachevol \
  regionsize=32k autogrow=on
```

4 Having created the cache object, use the following command to enable it:

```
# vxcache [-g diskgroup] start cache_object
```

For example to start the cache object, `cobjmydg`:

```
# vxcache -g mydg start cobjmydg
```

See "Removing a cache" on page 420.

Creating a volume for use as a full-sized instant or linked break-off snapshot

To create an empty volume for use by a full-sized instant snapshot or a linked break-off snapshot

1 Use the `vxprint` command on the original volume to find the required size for the snapshot volume.

```
# LEN=`xprint [-g diskgroup] -F%len volume`
```

The command as shown assumes that you are using a Bourne-type shell such as `sh`, `ksh` or `bash`. You may need to modify the command for other shells such as `csh` or `tcsh`.

2 Use the `vxprint` command on the original volume to discover the name of its DCO:

```
# DCONAME=`vxprint [-g diskgroup] -F%dco_name volume`
```

3 Use the `vxprint` command on the DCO to discover its region size (in blocks):

```
# RSZ=`vxprint [-g diskgroup] -F%regionsz $DCONAME`
```

4 Use the `vxassist` command to create a volume, *snapvol*, of the required size and redundancy, together with a version 20 DCO volume with the correct region size:

```
# vxassist [-g diskgroup] make snapvol $LEN \
  [layout=mirror nmirror=number] logtype=dco drl=off \
  dcoversion=20 [ndcomirror=number] regionsz=$RSZ \
  init=active [storage_attributes]
```

Specify the same number of DCO mirrors (`ndcomirror`) as the number of mirrors in the volume (`nmirror`). The `init=active` attribute is used to make the volume available immediately. You can use storage attributes to specify which disks should be used for the volume.

As an alternative to creating the snapshot volume and its DCO volume in a single step, you can first create the volume, and then prepare it for instant snapshot operations as shown here:

```
# vxassist [-g diskgroup] make snapvol $LEN \
  [layout=mirror nmirror=number] init=active \
  [storage_attributes]
# vxsnap [-g diskgroup] prepare snapvol [ndcomirs=number] \
  regionsize=$RSZ [storage_attributes]
```

Creating and managing space-optimized instant snapshots

Space-optimized instant snapshots are not suitable for write-intensive volumes (such as for database redo logs) because the copy-on-write mechanism may degrade the performance of the volume.

If you intend to split the volume and snapshot into separate disk groups (for example, to perform off-host processing), you must use a fully synchronized full-sized instant, third-mirror break-off or linked break-off snapshot (which do not require a cache object). You cannot use a space-optimized instant snapshot for this purpose.

Creation of space-optimized snapshots that use a shared cache fails if the region size specified for the volume is smaller than the region size set on the cache.

If the region size of a space-optimized snapshot differs from the region size of the cache, this can degrade the system's performance compared to the case where the region sizes are the same.

For space-optimized instant snapshots that share a cache object, the specified region size must be greater than or equal to the region size specified for the cache object.

See "Creating a shared cache object" on page 391.

The attributes for a snapshot are specified as a tuple to the `vxsnap make` command. This command accepts multiple tuples. One tuple is required for each snapshot that is being created. Each element of a tuple is separated from the next by a slash character (`/`). Tuples are separated by white space.

To create and manage a space-optimized instant snapshot

1 Use the `vxsnap make` command to create a space-optimized instant snapshot. This snapshot can be created by using an existing cache object in the disk group, or a new cache object can be created for its use.

- To create a space-optimized instant snapshot, *`snapvol`*, that uses a named shared cache object:

```
# vxsnap [-g diskgroup] make source=vol/newvol=snapvol\
/[cache=cacheobject] [alloc=storage_attributes]
```

For example, to create the space-optimized instant snapshot, `snap3myvol`, of the volume, `myvol`, in the disk group, `mydg`, on the disk `mydg14`, and which uses the shared cache object, `cobjmydg`, use the following command:

```
# vxsnap -g mydg make source=myvol/newvol=snap3myvol\
/cache=cobjmydg alloc=mydg14
```

See "Creating a shared cache object" on page 391.

- To create a space-optimized instant snapshot, *`snapvol`*, and also create a cache object for it to use:

```
# vxsnap [-g diskgroup] make source=vol/newvol=snapvol\
[/cachesize=size][/autogrow=yes][/ncachemirror=number]\
  [alloc=storage_attributes]
```

The `cachesize` attribute determines the size of the cache relative to the size of the volume. The `autogrow` attribute determines whether VxVM will automatically enlarge the cache if it is in danger of overflowing. By default, the cache is not grown.
If `autogrow` is enabled, but the cache cannot be grown, VxVM disables the oldest and largest snapshot that is using the same cache, and releases its cache space for use.

The `ncachemirror` attribute specifies the number of mirrors to create in the cache volume. For backup purposes, the default value of 1 should be sufficient.
For example, to create the space-optimized instant snapshot, `snap4myvol`, of the volume, `myvol`, in the disk group, `mydg`, on the disk `mydg15`, and which uses a newly allocated cache object that is 1GB in size, but which can automatically grow in size, use the following command:

```
# vxsnap -g mydg make source=myvol/new=snap4myvol\
/cachesize=1g/autogrow=yes alloc=mydg15
```

If a cache is created implicitly by specifying `cachesize`, and `ncachemirror` is specified to be greater than 1, a DCO is attached to the cache volume to enable dirty region logging (DRL). DRL allows fast recovery of the cache backing store after a system crash. The DCO is allocated on the same disks as those that are occupied by the DCO of the source volume. This is done to allow the cache and the source volume to remain in the same disk group for disk group move, split and join operations.

2 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
```

The specified device must have a valid entry in the `/etc/vfstab` file.

3 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.

4 You now have the following choices of what to do with a space-optimized instant snapshot:

- Refresh the contents of the snapshot. This creates a new point-in-time image of the original volume ready for another backup. If synchronization was already in progress on the snapshot, this operation may result in large portions of the snapshot having to be resynchronized.
  See "Refreshing an instant snapshot" on page 409.
- Restore the contents of the original volume from the snapshot volume. The space-optimized instant snapshot remains intact at the end of the operation.
  See "Restoring a volume from an instant snapshot" on page 411.

Creating and managing full-sized instant snapshots

Full-sized instant snapshots are not suitable for write-intensive volumes (such as for database redo logs) because the copy-on-write mechanism may degrade the performance of the volume.

For full-sized instant snapshots, you must prepare a volume that is to be used as the snapshot volume. This must be the same size as the volume for which the snapshot is being created, and it must also have the same region size.

See "Creating a volume for use as a full-sized instant or linked break-off snapshot" on page 393.

The attributes for a snapshot are specified as a tuple to the `vxsnap make` command. This command accepts multiple tuples. One tuple is required for each snapshot that is being created. Each element of a tuple is separated from the next by a slash character (`/`). Tuples are separated by white space.

To create and manage a full-sized instant snapshot

1 To create a full-sized instant snapshot, use the following form of the `vxsnap make` command:

```
# vxsnap [-g diskgroup] make source=volume/snapvol=snapvol\
[/snapdg=snapdiskgroup] [/syncing=off]
```

The command specifies the volume, *snapvol*, that you prepared earlier.

For example, to use the prepared volume, `snap1myvol`, as the snapshot for the volume, `myvol`, in the disk group, `mydg`, use the following command:

```
# vxsnap -g mydg make source=myvol/snapvol=snap1myvol
```

For full-sized instant snapshots that are created from an empty volume, background synchronization is enabled by default (equivalent to specifying the `syncing=on` attribute). If you want to move a snapshot into a separate disk group, or to turn it into an independent volume, you must wait for its contents to be synchronized with those of its parent volume.

You can use the `vxsnap syncwait` command to wait for the synchronization of the snapshot volume to be completed, as shown here:

```
# vxsnap [-g diskgroup] syncwait snapvol
```

For example, you would use the following command to wait for synchronization to finish on the snapshot volume, snap2myvol:

```
# vxsnap -g mydg syncwait snap2myvol
```

This command exits (with a return code of zero) when synchronization of the snapshot volume is complete. The snapshot volume may then be moved to another disk group or turned into an independent volume.

See "Controlling instant snapshot synchronization" on page 416.

If required, you can use the following command to test if the synchronization of a volume is complete:

```
# vxprint [-g diskgroup] -F%incomplete snapvol
```

This command returns the value `off` if synchronization of the volume, *snapvol*, is complete; otherwise, it returns the value `on`.

You can also use the `vxsnap print` command to check on the progress of synchronization.

See "Displaying instant snapshot information" on page 414.

If you do not want to move the snapshot into a separate disk group, or to turn it into an independent volume, specify the `syncing=off` attribute. This avoids creating unnecessary system overhead. For example, to turn off synchronization when creating the snapshot of the volume, `myvol`, you would use the following form of the `vxsnap make` command:

```
# vxsnap -g mydg make source=myvol/snapvol=snap1myvol\
/syncing=off
```

2 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
```

The specified device must have a valid entry in the `/etc/vfstab` file.

3 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.

4 You now have the following choices of what to do with a full-sized instant snapshot:

- Refresh the contents of the snapshot. This creates a new point-in-time image of the original volume ready for another backup. If synchronization was already in progress on the snapshot, this operation may result in large portions of the snapshot having to be resynchronized.
  See "Refreshing an instant snapshot" on page 409.

- Reattach some or all of the plexes of the snapshot volume with the original volume.
  See "Reattaching an instant snapshot" on page 409.
- Restore the contents of the original volume from the snapshot volume. You can choose whether none, a subset, or all of the plexes of the snapshot volume are returned to the original volume as a result of the operation.
  See "Restoring a volume from an instant snapshot" on page 411.
- Dissociate the snapshot volume entirely from the original volume. This may be useful if you want to use the copy for other purposes such as testing or report generation. If desired, you can delete the dissociated volume.
  See "Dissociating an instant snapshot" on page 412.
- If the snapshot is part of a snapshot hierarchy, you can also choose to split this hierarchy from its parent volumes.
  See "Splitting an instant snapshot hierarchy" on page 413.

Creating and managing third-mirror break-off snapshots

Break-off snapshots are suitable for write-intensive volumes, such as database redo logs.

To turn one or more existing plexes in a volume into a break-off instant snapshot volume, the volume must be a non-layered volume with a `mirror` or `mirror-stripe` layout, or a RAID-5 volume that you have converted to a special layered volume and then mirrored. The plexes in a volume with a `stripe-mirror` layout are mirrored at the subvolume level, and cannot be broken off.

See "Using a DCO and DCO volume with a RAID-5 volume" on page 343.

The attributes for a snapshot are specified as a tuple to the `vxsnap make` command. This command accepts multiple tuples. One tuple is required for each snapshot that is being created. Each element of a tuple is separated from the next by a slash character (/). Tuples are separated by white space.

To create and manage a third-mirror break-off snapshot

1 To create the snapshot, you can either take some of the existing ACTIVE plexes in the volume, or you can use the following command to add new snapshot mirrors to the volume:

```
# vxsnap [-b] [-g diskgroup] addmir volume [nmirror=N] \
  [alloc=storage_attributes]
```

By default, the vxsnap addmir command adds one snapshot mirror to a volume unless you use the nmirror attribute to specify a different number of mirrors. The mirrors remain in the SNAPATT state until they are fully synchronized. The -b option can be used to perform the synchronization in the background. Once synchronized, the mirrors are placed in the SNAPDONE state.

For example, the following command adds 2 mirrors to the volume, vol1, on disks mydg10 and mydg11:

```
# vxsnap -g mydg addmir vol1 nmirror=2 alloc=mydg10,mydg11
```

If you specify the -b option to the vxsnap addmir command, you can use the vxsnap snapwait command to wait for synchronization of the snapshot plexes to complete, as shown in this example:

```
# vxsnap -g mydg snapwait vol1 nmirror=2
```

2 To create a third-mirror break-off snapshot, use the following form of the `vxsnap make` command.

```
# vxsnap [-g diskgroup] make source=volume[/newvol=snapvol]\
{/plex=plex1[,plex2,...]|/nmirror=number]}
```

Either of the following attributes may be specified to create the new snapshot volume, *snapvol*, by breaking off one or more existing plexes in the original volume:

`plex`	Specifies the plexes in the existing volume that are to be broken off. This attribute can only be used with plexes that are in the `ACTIVE` state.
`nmirror`	Specifies how many plexes are to be broken off. This attribute can only be used with plexes that are in the `SNAPDONE` state. (Such plexes could have been added to the volume by using the `vxsnap addmir` command.)

Snapshots that are created from one or more `ACTIVE` or `SNAPDONE` plexes in the volume are already synchronized by definition.

For backup purposes, a snapshot volume with one plex should be sufficient.

For example, to create the instant snapshot volume, `snap2myvol`, of the volume, `myvol`, in the disk group, `mydg`, from a single existing plex in the volume, use the following command:

```
# vxsnap -g mydg make source=myvol/newvol=snap2myvol/nmirror=1
```

The next example shows how to create a mirrored snapshot from two existing plexes in the volume:

```
# vxsnap -g mydg make source=myvol/newvol=snap2myvol/plex=myvol-03,myvo
```

3 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
```

The specified device must have a valid entry in the `/etc/vfstab` file.

4 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.

5 You now have the following choices of what to do with a third-mirror break-off snapshot:

- Refresh the contents of the snapshot. This creates a new point-in-time image of the original volume ready for another backup. If synchronization was already in progress on the snapshot, this operation may result in large portions of the snapshot having to be resynchronized.
  See "Refreshing an instant snapshot" on page 409.
- Reattach some or all of the plexes of the snapshot volume with the original volume.
  See "Reattaching an instant snapshot" on page 409.
- Restore the contents of the original volume from the snapshot volume. You can choose whether none, a subset, or all of the plexes of the snapshot volume are returned to the original volume as a result of the operation.
  See "Restoring a volume from an instant snapshot" on page 411.
- Dissociate the snapshot volume entirely from the original volume. This may be useful if you want to use the copy for other purposes such as testing or report generation. If desired, you can delete the dissociated volume.
  See "Dissociating an instant snapshot" on page 412.
- If the snapshot is part of a snapshot hierarchy, you can also choose to split this hierarchy from its parent volumes.
  See "Splitting an instant snapshot hierarchy" on page 413.

Creating and managing linked break-off snapshot volumes

Break-off snapshots are suitable for write-intensive volumes, such as database redo logs.

For linked break-off snapshots, you must prepare a volume that is to be used as the snapshot volume. This must be the same size as the volume for which the snapshot is being created, and it must also have the same region size.

See "Creating a volume for use as a full-sized instant or linked break-off snapshot" on page 393.

The attributes for a snapshot are specified as a tuple to the `vxsnap make` command. This command accepts multiple tuples. One tuple is required for each snapshot that is being created. Each element of a tuple is separated from the next by a slash character (/). Tuples are separated by white space.

To create and manage a linked break-off snapshot

1 Use the following command to link the prepared snapshot volume, *snapvol*, to the data volume:

```
# vxsnap [-g diskgroup] [-b] addmir volume mirvol=snapvol \
  [mirdg=snapdg]
```

The optional `mirdg` attribute can be used to specify the snapshot volume's current disk group, *snapdg*. The `-b` option can be used to perform the synchronization in the background. If the `-b` option is not specified, the command does not return until the link becomes `ACTIVE`.

For example, the following command links the prepared volume, `prepsnap`, in the disk group, `mysnapdg`, to the volume, `vol1`, in the disk group, `mydg`:

```
# vxsnap -g mydg -b addmir vol1 mirvol=prepsnap mirdg=mysnapdg
```

If the `-b` option is specified, you can use the `vxsnap snapwait` command to wait for the synchronization of the linked snapshot volume to complete, as shown in this example:

```
# vxsnap -g mydg snapwait vol1 mirvol=prepsnap mirdg=mysnapvoldg
```

2 To create a linked break-off snapshot, use the following form of the `vxsnap make` command.

```
# vxsnap [-g diskgroup] make   source=volume/snapvol=snapvol\
[/snapdg=snapdiskgroup]
```

The `snapdg` attribute must be used to specify the snapshot volume's disk group if this is different from that of the data volume.

For example, to use the prepared volume, `prepsnap`, as the snapshot for the volume, `vol1`, in the disk group, `mydg`, use the following command:

```
# vxsnap -g mydg make source=vol1/snapvol=prepsnap/snapdg=mysnapdg
```

3 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
```

The specified device must have a valid entry in the `/etc/vfstab file`.

4 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.

5 You now have the following choices of what to do with a linked break-off snapshot:

- Refresh the contents of the snapshot. This creates a new point-in-time image of the original volume ready for another backup. If synchronization was already in progress on the snapshot, this operation may result in large portions of the snapshot having to be resynchronized.
  See "Refreshing an instant snapshot" on page 409.
- This operation is not possible if the linked volume and snapshot are in different disk groups.Reattach the snapshot volume with the original volume.
  See "Reattaching a linked break-off snapshot volume" on page 410.
- Dissociate the snapshot volume entirely from the original volume. This may be useful if you want to use the copy for other purposes such as testing or report generation. If desired, you can delete the dissociated volume.
  See "Dissociating an instant snapshot" on page 412.
- If the snapshot is part of a snapshot hierarchy, you can also choose to split this hierarchy from its parent volumes.
  See "Splitting an instant snapshot hierarchy" on page 413.

Creating multiple instant snapshots

To make it easier to create snapshots of several volumes at the same time, the `vxsnap make` command accepts multiple tuples that define the source and snapshot volumes names as their arguments. For example, to create three instant snapshots, each with the same redundancy, from specified storage, the following form of the command can be used:

```
# vxsnap [-g diskgroup] make source=vol1/snapvol=snapvol1\
  source=vol2/snapvol=snapvol2 source=vol3/snapvol=snapvol3
```

The snapshot volumes (*snapvol1*, *snapvol2* and so on) must have been prepared in advance.

See "Creating a volume for use as a full-sized instant or linked break-off snapshot" on page 393.

The specified source volumes (*vol1*, *vol2* and so on) may be the same volume or they can be different volumes.

If all the snapshots are to be space-optimized and to share the same cache, the following form of the command can be used:

```
# vxsnap [-g diskgroup] make \
  source=vol1/newvol=snapvol1/cache=cacheobj \
  source=vol2/newvol=snapvol2/cache=cacheobj \
  source=vol3/newvol=snapvol3/cache=cacheobj \
  [alloc=storage_attributes]
```

The `vxsnap make` command also allows the snapshots to be of different types, have different redundancy, and be configured from different storage, as shown here:

```
# vxsnap [-g diskgroup] make source=vol1/snapvol=snapvol1 \
  source=vol2[/newvol=snapvol2]/cache=cacheobj\
[/alloc=storage_attributes2][/nmirror=number2]
  source=vol3[/newvol=snapvol3][/alloc=storage_attributes3]\
/nmirror=number3
```

In this example, *snapvol1* is a full-sized snapshot that uses a prepared volume, *snapvol2* is a space-optimized snapshot that uses a prepared cache, and *snapvol3* is a break-off full-sized snapshot that is formed from plexes of the original volume.

An example of where you might want to create mixed types of snapshots at the same time is when taking snapshots of volumes containing database redo logs and database tables:

```
# vxsnap -g mydg make \
  source=logv1/newvol=snplogv1/drl=sequential/nmirror=1 \
  source=logv2/newvol=snplogv2/drl=sequential/nmirror=1 \
  source=datav1/newvol=snpdatav1/cache=mydgcobj/drl=on \
  source=datav2/newvol=snpdatav2/cache=mydgcobj/drl=on
```

In this example, sequential DRL is enabled for the snapshots of the redo log volumes, and normal DRL is applied to the snapshots of the volumes that contain the database tables. The two space-optimized snapshots are configured to share the same cache object in the disk group. Also note that break-off snapshots are used for the redo logs as such volumes are write intensive.

Creating instant snapshots of volume sets

Volume set names can be used in place of volume names with the following `vxsnap` operations on instant snapshots: `addmir`, `dis`, `make`, `prepare`, `reattach`, `refresh`, `restore`, `rmmir`, `split`, `syncpause`, `syncresume`, `syncstart`, `syncstop`, `syncwait`, and `unprepare`.

The procedure for creating an instant snapshot of a volume set is the same as that for a standalone volume. However, there are certain restrictions if a full-sized instant snapshot is to be created from a prepared volume set. A full-sized instant snapshot of a volume set must itself be a volume set with the same number of volumes, and the same volume sizes and index numbers as the parent. For example, if a volume set contains three volumes with sizes 1GB, 2GB and 3GB, and indexes 0, 1 and 2 respectively, then the snapshot volume set must have three volumes with the same sizes matched to the same set of index numbers. The corresponding volumes in the parent and snapshot volume sets are also subject to the same restrictions as apply between standalone volumes and their snapshots.

You can use the `vxvset list` command to verify that the volume sets have identical characteristics as shown in this example:

```
# vxvset -g mydg list vset1

VOLUME       INDEX      LENGTH      KSTATE      CONTEXT
vol_0        0          204800      ENABLED     -
vol_1        1          409600      ENABLED     -
vol_2        2          614400      ENABLED     -

# vxvset -g mydg list snapvset1

VOLUME       INDEX      LENGTH      KSTATE      CONTEXT
svol_0       0          204800      ENABLED     -
svol_1       1          409600      ENABLED     -
svol_2       2          614400      ENABLED     -
```

A full-sized instant snapshot of a volume set can be created using a prepared volume set in which each volume is the same size as the corresponding volume in the parent volume set. Alternatively, you can use the `nmirrors` attribute to specify the number of plexes that are to be broken off provided that sufficient plexes exist for each volume in the volume set.

The following example shows how to prepare a source volume set, `vset1`, and an identical volume set, `snapvset1`, which is then used to create the snapshot:

```
# vxsnap -g mydg prepare vset1
# vxsnap -g mydg prepare snapvset1
# vxsnap -g mydg make source=vset1/snapvol=snapvset1
```

To create a full-sized third-mirror break-off snapshot, you must ensure that each volume in the source volume set contains sufficient plexes. The following example shows how to achieve this by using the `vxsnap` command to add the required number of plexes before breaking off the snapshot:

```
# vxsnap -g mydg prepare vset2
# vxsnap -g mydg addmir vset2 nmirror=1
# vxsnap -g mydg make source=vset2/newvol=snapvset2/nmirror=1
```

See "Adding snapshot mirrors to a volume" on page 407.

To create a space-optimized instant snapshot of a volume set, the commands are again identical to those for a standalone volume as shown in these examples:

```
# vxsnap -g mydg prepare vset3
# vxsnap -g mydg make source=vset3/newvol=snapvset3/cachesize=20m

# vxsnap -g mydg prepare vset4
# vxsnap -g mydg make source=vset4/newvol=snapvset4/cache=mycobj
```

Here a new cache object is created for the volume set, `vset3`, and an existing cache object, `mycobj`, is used for `vset4`.

See "About volume sets" on page 435.

Adding snapshot mirrors to a volume

If you are going to create a full-sized break-off snapshot volume, you can use the following command to add new snapshot mirrors to a volume:

```
# vxsnap [-b] [-g diskgroup] addmir volume|volume_set \
  [nmirror=N] [alloc=storage_attributes]
```

The volume must have been prepared using the `vxsnap prepare` command.

See "Preparing a volume for DRL and instant snapshots" on page 341.

If a volume set name is specified instead of a volume, the specified number of plexes is added to each volume in the volume set.

By default, the `vxsnap addmir` command adds one snapshot mirror to a volume unless you use the `nmirror` attribute to specify a different number of mirrors. The mirrors remain in the `SNAPATT` state until they are fully synchronized. The `-b` option can be used to perform the synchronization in the background. Once synchronized, the mirrors are placed in the `SNAPDONE` state.

For example, the following command adds 2 mirrors to the volume, `vol1`, on disks `mydg10` and `mydg11`:

```
# vxsnap -g mydg addmir vol1 nmirror=2 alloc=mydg10,mydg11
```

This command is similar in usage to the `vxassist snapstart` command, and supports the traditional third-mirror break-off snapshot model. As such, it does not provide an instant snapshot capability.

Once you have added one or more snapshot mirrors to a volume, you can use the `vxsnap make` command with either the `nmirror` attribute or the `plex` attribute to create the snapshot volumes.

Removing a snapshot mirror

To remove a single snapshot mirror from a volume, use this command:

```
# vxsnap [-g diskgroup] rmmir volume|volume_set
```

For example, the following command removes a snapshot mirror from the volume, `vol1`:

```
# vxsnap -g mydg rmmir vol1
```

This command is similar in usage to the `vxassist snapabort` command.

If a volume set name is specified instead of a volume, a mirror is removed from each volume in the volume set.

Removing a linked break-off snapshot volume

To remove a linked break-off snapshot volume from a volume, use this command:

```
# vxsnap [-g diskgroup] rmmir volume|volume_set mirvol=snapvol \
  [mirdg=snapdiskgroup]
```

The `mirvol` and optional `mirdg` attributes specify the snapshot volume, *snapvol*, and its disk group, *snapdiskgroup*. For example, the following command removes a linked snapshot volume, `prepsnap`, from the volume, `vol1`:

```
# vxsnap -g mydg rmmir vol1 mirvol=prepsnap mirdg=mysnapdg
```

Adding a snapshot to a cascaded snapshot hierarchy

To create a snapshot and push it onto a snapshot hierarchy between the original volume and an existing snapshot volume, specify the name of the existing snapshot volume as the value of the `infrontof` attribute to the `vxsnap make` command. The following example shows how to place the space-optimized snapshot, `thurs_bu`, of the volume, `dbvol`, in front of the earlier snapshot, `wed_bu`:

```
# vxsnap -g dbdg make source=dbvol/newvol=thurs_bu/\
infrontof=wed_bu/cache=dbdgcache
```

Similarly, the next snapshot that is taken, `fri_bu`, is placed in front of `thurs_bu`:

```
# vxsnap -g dbdg make source=dbvol/newvol=fri_bu/\
infrontof=thurs_bu/cache=dbdgcache
```

See "Cascaded snapshots" on page 381.

Refreshing an instant snapshot

Refreshing an instant snapshot replaces it with another point-in-time copy of a parent volume. To refresh one or more snapshots and make them immediately available for use, use the following command:

```
# vxsnap [-g diskgroup] refresh snapvolume|snapvolume_set \
  source=volume|volume_set [[snapvol2 source=vol2]...] \
  [syncing=yes|no]
```

If the source volume is not specified, the immediate parent of the snapshot is used. For full-sized instant snapshots, resynchronization is started by default. To disable resynchronization, specify the `syncing=no` attribute. This attribute is not supported for space-optimized snapshots.

Warning: The snapshot that is being refreshed must not be open to any application. For example, any file system configured on the volume must first be unmounted.

It is possible to refresh a volume from an unrelated volume provided that their sizes are compatible.

You can use the `vxsnap syncwait` command to wait for the synchronization of the snapshot volume to be completed, as shown here:

```
# vxsnap [-g diskgroup] syncwait snapvol
```

See "Controlling instant snapshot synchronization" on page 416.

Reattaching an instant snapshot

Note: This operation is not supported for space-optimized instant snapshots.

Using the following command, some or all plexes of an instant snapshot may be reattached to the specified original volume, or to a source volume in the snapshot hierarchy above the snapshot volume:

```
# vxsnap [-g diskgroup] reattach snapvolume|snapvolume_set \
  source=volume|volume_set [nmirror=number]
```

By default, all the plexes are reattached, which results in the removal of the snapshot. If required, the number of plexes to be reattached may be specified as the value assigned to the `nmirror` attribute.

Warning: The snapshot that is being reattached must not be open to any application. For example, any file system configured on the snapshot volume must first be unmounted.

It is possible to reattach a volume to an unrelated volume provided that their volume sizes and region sizes are compatible.

For example the following command reattaches one plex from the snapshot volume, `snapmyvol`, to the volume, `myvol`:

```
# vxsnap -g mydg reattach snapmyvol source=myvol nmirror=1
```

While the reattached plexes are being resynchronized from the data in the parent volume, they remain in the `SNAPTMP` state. After resynchronization is complete, the plexes are placed in the `SNAPDONE` state. You can use the `vxsnap snapwait` command (but not `vxsnap syncwait`) to wait for the resynchronization of the reattached plexes to complete, as shown here:

```
# vxsnap -g mydg snapwait myvol nmirror=1
```

If the volume and its snapshot have both been resized (to an identical smaller or larger size) before performing the reattachment, a fast resynchronization can still be performed. A full resynchronization is not required. Version 20 DCO volumes are resized proportionately when the associated data volume is resized. For version 0 DCO volumes, the FastResync maps stay the same size, but the region size is recalculated, and the locations of the dirty bits in the existing maps are adjusted. In both cases, new regions are marked as dirty in the maps.

Reattaching a linked break-off snapshot volume

Unlike other types of snapshot, the reattachment operation for linked break-off snapshot volumes does not return the plexes of the snapshot volume to the parent volume. The link relationship is re-established that makes the snapshot volume a mirror of the parent volume, and this allows the snapshot data to be

resynchronized. However, the snapshot volume is only readopted by its parent volume if they are both in the same disk group.

To reattach a linked break-off snapshot volume, use the following form of the `vxsnap reattach` command:

```
# vxsnap [-g snapdiskgroup] reattach snapvolume|snapvolume_set \
  source=volume|volume_set [sourcedg=diskgroup]
```

The `sourcedg` attribute must be used to specify the data volume's disk group if this is different from the snapshot volume's disk group, *snapdiskgroup*.

Warning: The snapshot that is being reattached must not be open to any application. For example, any file system configured on the snapshot volume must first be unmounted.

It is possible to reattach a volume to an unrelated volume provided that their sizes and region sizes are compatible.

For example the following command reattaches the snapshot volume, `prepsnap`, in the disk group, `snapdg`, to the volume, `myvol`, in the disk group, `mydg`:

```
# vxsnap -g snapdg reattach prepsnap source=myvol sourcedg=mydg
```

After resynchronization of the snapshot volume is complete, the link is placed in the `ACTIVE` state. You can use the `vxsnap snapwait` command (but not `vxsnap syncwait`) to wait for the resynchronization of the reattached volume to complete, as shown here:

```
# vxsnap -g snapdg snapwait myvol mirvol=prepsnap
```

Restoring a volume from an instant snapshot

It may sometimes be desirable to reinstate the contents of a volume from a backup or modified replica in a snapshot volume. The following command may be used to restore one or more volumes from the specified snapshots:

```
# vxsnap [-g diskgroup] restore volume|volume_set \
  source=snapvolume|snapvolume_set \
  [[volume2|volume_set2 \
  source=snapvolume2|snapvolume_set2]...]\
  [destroy=yes|no] [syncing=yes|no] [nmirror=number]
```

For a full-sized instant snapshot, some or all of its plexes may be reattached to the parent volume or to a specified source volume in the snapshot hierarchy above

the snapshot volume. If `destroy=yes` is specified, all the plexes of the full-sized instant snapshot are reattached and the snapshot volume is removed.

For a space-optimized instant snapshot, the cached data is used to recreate the contents of the specified volume. The space-optimized instant snapshot remains unchanged by the `restore` operation.

Warning: For this operation to succeed, the volume that is being restored and the snapshot volume must not be open to any application. For example, any file systems that are configured on either volume must first be unmounted.

It is not possible to restore a volume from an unrelated volume.

The `destroy` and `nmirror` attributes are not supported for space-optimized instant snapshots.

The following example demonstrates how to restore the volume, `myvol`, from the space-optimized snapshot, `snap3myvol`.

```
# vxsnap -g mydg restore myvol source=snap3myvol
```

Dissociating an instant snapshot

The following command breaks the association between a full-sized instant snapshot volume, *snapvol*, and its parent volume, so that the snapshot may be used as an independent volume:

```
# vxsnap [-f] [-g diskgroup] dis snapvolume|snapvolume_set
```

This operation fails if the snapshot, *snapvol*, has a snapshot hierarchy below it that contains unsynchronized snapshots. If this happens, the dependent snapshots must be fully synchronized from *snapvol*. When no dependent snapshots remain, *snapvol* may be dissociated. The snapshot hierarchy is then adopted by the parent volume of *snapvol*.

To be usable after dissociation, the snapshot volume and any snapshots in the hierarchy must have been fully synchronized.

See "Controlling instant snapshot synchronization" on page 416.

In addition, you cannot dissociate a snapshot if synchronization of any of the dependent snapshots in the hierarchy is incomplete. If an incomplete snapshot is dissociated, it is unusable and should be deleted.

See "Removing an instant snapshot" on page 413.

The following command dissociates the snapshot, `snap2myvol`, from its parent volume:

```
# vxsnap -g mydg dis snap2myvol
```

Warning: When applied to a volume set or to a component volume of a volume set, this operation can result in inconsistencies in the snapshot hierarchy in the case of a system crash or hardware failure. If the operation is applied to a volume set, the `-f` (force) option must be specified.

Removing an instant snapshot

When you have dissociated a full-sized instant snapshot, you can use the `vxedit` command to delete it altogether, as shown in this example:

```
# vxedit -g mydg -r rm snap2myvol
```

You can also use this command to remove a space-optimized instant snapshot from its cache.

See "Removing a cache" on page 420.

Splitting an instant snapshot hierarchy

Note: This operation is not supported for space-optimized instant snapshots.

The following command breaks the association between a snapshot hierarchy that has the snapshot volume, *snapvol*, at its head, and its parent volume, so that the snapshot hierarchy may be used independently of the parent volume:

```
# vxsnap [-f] [-g diskgroup] split snapvolume|snapvolume_set
```

The topmost snapshot volume in the hierarchy must have been fully synchronized for this command to succeed. Snapshots that are lower down in the hierarchy need not have been fully resynchronized.

See "Controlling instant snapshot synchronization" on page 416.

The following command splits the snapshot hierarchy under `snap2myvol` from its parent volume:

```
# vxsnap -g mydg split snap2myvol
```

Warning: When applied to a volume set or to a component volume of a volume set, this operation can result in inconsistencies in the snapshot hierarchy in the case of a system crash or hardware failure. If the operation is applied to a volume set, the `-f` (force) option must be specified.

Displaying instant snapshot information

The `vxsnap print` command may be used to display information about the snapshots that are associated with a volume.

```
# vxsnap [-g diskgroup] print [vol]
```

This command shows the percentage progress of the synchronization of a snapshot or volume. If no volume is specified, information about the snapshots for all the volumes in a disk group is displayed. The following example shows a volume, `vol1`, which has a full-sized snapshot, `snapvol1` whose contents have not been synchronized with `vol1`:

```
# vxsnap -g mydg print
NAME     SNAPOBJECT     TYPE    PARENT   SNAPSHOT   %DIRTY   %VALID

vol1     --             volume  --       --         --       100
         snapvol1_snp1  volume  --       snapvol1   1.30     --
snapvol1 vol1_snp1      volume  vol1     --         1.30     1.30
```

The `%DIRTY` value for `snapvol1` shows that its contents have changed by 1.30% when compared with the contents of `vol1`. As `snapvol1` has not been synchronized with `vol1`, the `%VALID` value is the same as the `%DIRTY` value. If the snapshot were partly synchronized, the `%VALID` value would lie between the `%DIRTY` value and 100%. If the snapshot were fully synchronized, the `%VALID` value would be 100%. The snapshot could then be made independent or moved into another disk group.

Additional information about the snapshots of volumes and volume sets can be obtained by using the `-n` option with the `vxsnap print` command:

```
# vxsnap [-g diskgroup] -n [-l] [-v] [-x] print [vol]
```

Alternatively, you can use the `vxsnap list` command, which is an alias for the `vxsnap -n print` command:

```
# vxsnap [-g diskgroup] [-l] [-v] [-x] list [vol]
```

The following output is an example of using this command on the disk group `dg1`:

```
# vxsnap -g dg -vx list

NAME    DG    OBJTYPE SNAPTYPE  PARENT  PARENTDG  SNAPDATE        CHANGE_DATA    SYNCED_DATA
vol     dg1   vol     -         -       -         -               -              10G (100%)
svol1   dg2   vol     fullinst  vol     dg1       2006/2/1 12:29  20M (0.2%)     60M (0.6%)
svol2   dg1   vol     mirbrk    vol     dg1       2006/2/1 12:29  120M (1.2%)    10G (100%)
svol3   dg2   vol     volbrk    vol     dg1       2006/2/1 12:29  105M (1.1%)    10G (100%)
svol21  dg1   vol     spaceopt  svol2   dg1       2006/2/1 12:29  52M (0.5%)     52M (0.5%)
vol-02  dg1   plex    snapmir   vol     dg1       -               -              56M (0.6%)
mvol    dg2   vol     mirvol    vol     dg1       -               -              58M (0.6%)
vset1   dg1   vset    -     -   -       -         -                              2G (100%)
v1      dg1   compvol -     -   -       -         -                              1G (100%)
v2      dg1   compvol -     -   -       -         -                              1G (100%)
svset1  dg1   vset    mirbrk    vset    dg1       2006/2/1 12:29  1G (50%)       2G (100%)
sv1     dg1   compvol mirbrk    v1      dg1       2006/2/1 12:29  512M (50%)     1G (100%)
sv2     dg1   compvol mirbrk    v2      dg1       2006/2/1 12:29  512M (50%)     1G (100%)
vol-03  dg1   plex    detmir    vol     dg1       -               20M (0.2%)     -
mvol2   dg2   vol     detvol    vol     dg1       -               20M (0.2%)     -
```

This shows that the volume `vol` has three full-sized snapshots, `svol1`, `svol2` and `svol3`, which are of types full-sized instant (`fullinst`), mirror break-off (`mirbrk`) and linked break-off (`volbrk`). It also has one snapshot plex (`snapmir`), `vol-02`, and one linked mirror volume (`mirvol`), `mvol`. The snapshot `svol2` itself has a space-optimized instant snapshot (`spaceopt`), `svol21`. There is also a volume set, `vset1`, with component volumes `v1` and `v2`. This volume set has a mirror break-off snapshot, `svset1`, with component volumes `sv1` and `sv2`. The last two entries show a detached plex, `vol-03`, and a detached mirror volume, `mvol2`, which have `vol` as their parent volume. These snapshot objects may have become detached due to an I/O error, or, in the case of the plex, by running the `vxplex det` command.

The `CHANGE_DATA` column shows the approximate difference between the current contents of the snapshot and its parent volume. This corresponds to the amount of data that would have to be resynchronized to make the contents the same again.

The `SYNCED_DATA` column shows the approximate progress of synchronization since the snapshot was taken.

The `-l` option can be used to obtain a longer form of the output listing instead of the tabular form.

The `-x` option expands the output to include the component volumes of volume sets.

See the vxsnap(1M) manual page for more information about using the vxsnap print and vxsnap list commands.

Controlling instant snapshot synchronization

Synchronization of the contents of a snapshot with its original volume is not possible for space-optimized instant snapshots.

By default, synchronization is enabled for the vxsnap reattach, refresh and restore operations on instant snapshots. Otherwise, synchronization is disabled unless you specify the syncing=yes attribute to the vxsnap command.

Table 9-1 shows the commands that are provided for controlling the synchronization manually.

Table 9-1 Commands for controlling instant snapshot synchronization

Command	Description
vxsnap [-g *diskgroup*] syncpause \ *vol*|*vol_set*	Pause synchronization of a volume.
vxsnap [-g *diskgroup*] syncresume \ *vol*|*vol_set*	Resume synchronization of a volume.
vxsnap [-b] [-g *diskgroup*] syncstart \ *vol*|*vol_set*	Start synchronization of a volume. The -b option puts the operation in the background.
vxsnap [-g *diskgroup*] syncstop \ *vol*|*vol_set*	Stop synchronization of a volume.
vxsnap [-g *diskgroup*] syncwait \ *vol*|*vol_set*	Exit when synchronization of a volume is complete. An error is returned if the *vol* or *vol_set* is invalid (for example, it is a space-optimized snapshot), or if the *vol* or *vol_set* is not being synchronized. **Note:** You cannot use this command to wait for synchronization of reattached plexes to complete.

The commands that are shown in Table 9-1 cannot be used to control the synchronization of linked break-off snapshots.

The `vxsnap snapwait` command is provided to wait for the link between new linked break-off snapshots to become ACTIVE, or for reattached snapshot plexes to reach the SNAPDONE state following resynchronization.

See "Creating and managing linked break-off snapshot volumes" on page 402.

See "Reattaching an instant snapshot" on page 409.

See "Reattaching a linked break-off snapshot volume" on page 410.

Improving the performance of snapshot synchronization

The following optional arguments to the `-o` option are provided to help optimize the performance of synchronization when using the `make`, `refresh`, `restore` and `syncstart` operations with full-sized instant snapshots:

`iosize=`*`size`*
: Specifies the size of each I/O request that is used when synchronizing the regions of a volume. Specifying a larger size causes synchronization to complete sooner, but with greater impact on the performance of other processes that are accessing the volume. The default size of 1m (1MB) is suggested as the minimum value for high-performance array and controller hardware. The specified value is rounded to a multiple of the volume's region size.

`slow=`*`iodelay`*
: Specifies the delay in milliseconds between synchronizing successive sets of regions as specified by the value of `iosize`. This can be used to change the impact of synchronization on system performance. The default value of *iodelay* is 0 milliseconds (no delay). Increasing this value slows down synchronization, and reduces the competition for I/O bandwidth with other processes that may be accessing the volume.

Options may be combined as shown in the following examples:

```
# vxsnap -g mydg -o iosize=2m,slow=100 make \
  source=myvol/snapvol=snap2myvol/syncing=on

# vxsnap -g mydg -o iosize=10m,slow=250 syncstart snap2myvol
```

Note: The `iosize` and `slow` parameters are not supported for space-optimized snapshots.

Listing the snapshots created on a cache

To list the space-optimized instant snapshots that have been created on a cache object, use the following command:

```
# vxcache [-g diskgroup] listvol cache_object
```

The snapshot names are printed as a space-separated list ordered by timestamp. If two or more snapshots have the same timestamp, these snapshots are sorted in order of decreasing size.

Tuning the autogrow attributes of a cache

The `highwatermark`, `autogrowby` and `maxautogrow` attributes determine how the VxVM cache daemon (`vxcached`) maintains the cache if the `autogrow` feature has been enabled and `vxcached` is running:

- When cache usage reaches the high watermark value, `highwatermark` (default value is 90 percent), `vxcached` grows the size of the cache volume by the value of `autogrowby` (default value is 20% of the size of the cache volume in blocks). The new required cache size cannot exceed the value of `maxautogrow` (default value is twice the size of the cache volume in blocks).
- When cache usage reaches the high watermark value, and the new required cache size would exceed the value of `maxautogrow`, `vxcached` deletes the oldest snapshot in the cache. If there are several snapshots with the same age, the largest of these is deleted.

If the `autogrow` feature has been disabled:

- When cache usage reaches the high watermark value, `vxcached` deletes the oldest snapshot in the cache. If there are several snapshots with the same age, the largest of these is deleted. If there is only a single snapshot, this snapshot is detached and marked as invalid.

Note: The `vxcached` daemon does not remove snapshots that are currently open, and it does not remove the last or only snapshot in the cache.

If the cache space becomes exhausted, the snapshot is detached and marked as invalid. If this happens, the snapshot is unrecoverable and must be removed. Enabling the `autogrow` feature on the cache helps to avoid this situation occurring. However, for very small caches (of the order of a few megabytes), it is possible for the cache to become exhausted before the system has time to respond and grow the cache. In such cases, you can increase the size of the cache manually.

See "Growing and shrinking a cache" on page 419.

Alternatively, you can use the `vxcache set` command to reduce the value of `highwatermark` as shown in this example:

```
# vxcache -g mydg set highwatermark=60 cobjmydg
```

You can use the `maxautogrow` attribute to limit the maximum size to which a cache can grow. To estimate this size, consider how much the contents of each source volume are likely to change between snapshot refreshes, and allow some additional space for contingency.

If necessary, you can use the `vxcache set` command to change other `autogrow` attribute values for a cache.

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

Warning: Ensure that the cache is sufficiently large, and that the `autogrow` attributes are configured correctly for your needs.

Growing and shrinking a cache

You can use the `vxcache` command to increase the size of the cache volume that is associated with a cache object:

```
# vxcache [-g diskgroup] growcacheto cache_object size
```

For example, to increase the size of the cache volume associated with the cache object, `mycache`, to 2GB, you would use the following command:

```
# vxcache -g mydg growcacheto mycache 2g
```

To grow a cache by a specified amount, use the following form of the command shown here:

```
# vxcache [-g diskgroup] growcacheby cache_object size
```

For example, the following command increases the size of `mycache` by 1GB:

```
# vxcache -g mydg growcacheby mycache 1g
```

You can similarly use the `shrinkcacheby` and `shrinkcacheto` operations to reduce the size of a cache.

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

Removing a cache

To remove a cache completely, including the cache object, its cache volume and all space-optimized snapshots that use the cache:

1. Run the following command to find out the names of the top-level snapshot volumes that are configured on the cache object:

```
# vxprint -g diskgroup -vne \
  "v_plex.pl_subdisk.sd_dm_name ~ /cache_object/"
```

   where *cache_object* is the name of the cache object.

2. Remove all the top-level snapshots and their dependent snapshots (this can be done with a single command):

```
# vxedit -g diskgroup -r rm snapvol ...
```

   where *snapvol* is the name of a top-level snapshot volume.

3. Stop the cache object:

```
# vxcache -g diskgroup stop cache_object
```

4. Finally, remove the cache object and its cache volume:

```
# vxedit -g diskgroup -r rm cache_object
```

Creating traditional third-mirror break-off snapshots

VxVM provides third-mirror break-off snapshot images of volume devices using `vxassist` and other commands.

To enhance the efficiency and usability of volume snapshots, turn on FastResync.

See "Enabling FastResync on a volume" on page 361.

If Persistent FastResync is required, you must associate a version 0 DCO with the volume.

See "Adding a version 0 DCO and DCO volume" on page 429.

A plex is required that is large enough to store the complete contents of the volume. Alternatively, you can use space-optimized instant snapshots.

See "Creating and managing space-optimized instant snapshots" on page 394.

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:

- Run vxassist snapstart to create a snapshot mirror.
- Run vxassist snapshot to create a snapshot volume.

The vxassist snapstart step creates a write-only backup plex which gets attached to and synchronized with the volume. When synchronized with the volume, the backup plex is ready to be used as a snapshot mirror. The end of the update procedure is indicated by the new snapshot mirror changing its state to SNAPDONE. This change can be tracked by the vxassist snapwait task, which waits until at least one of the mirrors changes its state to SNAPDONE. If the attach process fails, the snapshot mirror is removed and its space is released.

Note: If the snapstart procedure is interrupted, the snapshot mirror is automatically removed when the volume is started.

Once the snapshot mirror is synchronized, it continues being updated until it is detached. You can then select a convenient time at which to create a snapshot volume as an image of the existing volume. You can also ask users to refrain from using the system during the brief time required to perform the snapshot (typically less than a minute). The amount of time involved in creating the snapshot mirror is long in contrast to the brief amount of time that it takes to create the snapshot volume.

The online backup procedure is completed by running the vxassist snapshot command on a volume with a SNAPDONE mirror. This task detaches the finished snapshot (which becomes a normal mirror), creates a new normal volume and attaches the snapshot mirror to the snapshot volume. The snapshot then becomes a normal, functioning volume and the state of the snapshot is set to ACTIVE.

To back up a volume using the vxassist command

1 Create a snapshot mirror for a volume using the following command:

```
# vxassist [-b] [-g diskgroup] snapstart [nmirror=N] volume
```

For example, to create a snapshot mirror of a volume called `voldef`, use the following command:

```
# vxassist [-g diskgroup] snapstart voldef
```

The `vxassist snapstart` task creates a write-only mirror, which is attached to and synchronized from the volume to be backed up.

By default, VxVM attempts to avoid placing snapshot mirrors on a disk that already holds any plexes of a data volume. However, this may be impossible if insufficient space is available in the disk group. In this case, VxVM uses any available space on other disks in the disk group. If the snapshot plexes are placed on disks which are used to hold the plexes of other volumes, this may cause problems when you subsequently attempt to move a snapshot volume into another disk group.

See "Moving DCO volumes between disk groups" on page 255.

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 305.

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 424.

2 Choose a suitable time to create a snapshot. If possible, plan to take the snapshot at a time when users are accessing the volume as little as possible.

3 Create a snapshot volume using the following command:

```
# vxassist [-g diskgroup] snapshot [nmirror=N] volume snapshot
```

If required, use the `nmirror` attribute to specify the number of mirrors in the snapshot volume.

For example, to create a snapshot of `voldef`, use the following command:

```
# vxassist -g mydg snapshot voldef snapvoldef
```

The `vxassist snapshot` task detaches the finished snapshot mirror, creates a new volume, and attaches the snapshot mirror to it. This step should only take a few minutes. The snapshot volume, which reflects the original volume at the time of the snapshot, is now available for backing up, while the original volume continues to be available for applications and users.

If required, you can make snapshot volumes for several volumes in a disk group at the same time.

See "Creating multiple snapshots" on page 425.

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
```

The specified device must have a valid entry in the `/etc/vfstab file`.

5 If you require a backup of the data in the snapshot, use an appropriate utility or operating system command to copy the contents of the snapshot to tape, or to some other backup medium.

6 When the backup is complete, you have the following choices for what to do with the snapshot volume:

- Reattach some or all of the plexes of the snapshot volume with the original volume.
  See "Reattaching a snapshot volume" on page 425.
- If FastResync was enabled on the volume before the snapshot was taken, this speeds resynchronization of the snapshot plexes before the backup cycle starts again at step 3.
- Dissociate the snapshot volume entirely from the original volume
  See "Dissociating a snapshot volume" on page 427.
- This may be useful if you want to use the copy for other purposes such as testing or report generation.

- Remove the snapshot volume to save space with this command:

```
# vxedit [-g diskgroup] -rf rm snapshot
```

Dissociating or removing the snapshot volume loses the advantage of fast resynchronization if FastResync was enabled. If there are no further snapshot plexes available, any subsequent snapshots that you take require another complete copy of the original volume to be made.

Converting a plex into a snapshot plex

Note: A plex cannot be converted into a snapshot plex for layered volumes or for any volume that has an associated version 20 DCO volume.

It is recommended that the instant snapshot feature is used in preference to 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.

It is advisable to retain at least two plexes in a volume to maintain data redundancy.

To convert an existing plex into a snapshot plex for a volume on which Persistent FastResync is enabled, use the following command:

```
# vxplex [-g diskgroup] -o dcoplex=dcologplex convert \
  state=SNAPDONE plex
```

dcologplex is the name of an existing DCO plex that is to be associated with the new snapshot plex. You can use the `vxprint` command to find out the name of the DCO volume.

See "Adding a version 0 DCO and DCO volume" on page 429.

For example, to make a snapshot plex from the plex `trivol-03` in the 3-plex volume `trivol`, you would use the following command:

```
# vxplex -o dcoplex=trivol_dco-03 convert state=SNAPDONE \
  trivol-03
```

Here the DCO plex `trivol_dco_03` is specified as the DCO plex for the new snapshot plex.

To convert an existing plex into a snapshot plex in the SNAPDONE state for a volume on which Non-Persistent FastResync is enabled, use the following command:

```
# vxplex [-g diskgroup] convert state=SNAPDONE plex
```

A converted plex is in the SNAPDONE state, and can be used immediately to create a snapshot volume.

Note: The last complete regular plex in a volume, an incomplete regular plex, or a dirty region logging (DRL) log plex cannot be converted into a snapshot plex.

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, the first snapshot volume is named *SNAP-volume*, and each subsequent snapshot is named *SNAPnumber-volume*, where *number* is a unique serial number, and *volume* is the name of the volume for which the snapshot is being taken. This default pattern can be overridden by using the option `-o name=pattern`, as described on the `vxassist`(1M) manual page. For example, the pattern `SNAP%v-%d` reverses the order of the *number* and *volume* components in the name.

To snapshot all the volumes in a single disk group, specify the option `-o allvols` to `vxassist`:

```
# vxassist -g diskgroup -o allvols snapshot
```

This operation requires that all `snapstart` operations are complete on the volumes. It fails if any of the volumes in the disk group do not have a complete snapshot plex in the SNAPDONE state.

Reattaching a snapshot volume

The snapback operation merges a snapshot copy of a volume with the original volume. One or more snapshot plexes are detached from the snapshot volume

and re-attached to the original volume. The snapshot volume is removed if all its snapshot plexes are snapped back. This task resynchronizes the data in the volume so that the plexes are consistent.

The snapback operation cannot be applied to RAID-5 volumes unless they have been converted to a special layered volume layout by the addition of a DCO and DCO volume.

See "Adding a version 0 DCO and DCO volume" on page 429.

To enhance the efficiency of the snapback operation, enable FastResync on the volume before taking the snapshot

See "Enabling FastResync on a volume" on page 361.

To merge one snapshot plex with the original volume, use the following command:

```
# vxassist [-g diskgroup] snapback snapshot
```

where *snapshot* is the snapshot copy of the volume.

To merge all snapshot plexes in the snapshot volume with the original volume, use the following command:

```
# vxassist [-g diskgroup] -o allplexes snapback snapshot
```

To merge a specified number of plexes from the snapshot volume with the original volume, use the following command:

```
# vxassist [-g diskgroup] snapback nmirror=number snapshot
```

Here the `nmirror` attribute specifies the number of mirrors in the snapshot volume that are to be re-attached.

Once the snapshot plexes have been reattached and their data resynchronized, they are ready to be used in another `snapshot` operation.

By default, the data in the original volume is used to update the snapshot plexes that have been re-attached. To copy the data from the replica volume instead, use the following command:

```
# vxassist [-g diskgroup] -o resyncfromreplica snapback snapshot
```

Warning: Always unmount the snapshot volume (if this is mounted) before performing a snapback. In addition, you must unmount the file system corresponding to the primary volume before using the `resyncfromreplica` option.

Adding plexes to a snapshot volume

If you want to retain the existing plexes in a snapshot volume after a snapback operation, you can create additional snapshot plexes that are to be used for the snapback.

To add plexes to a snapshot volume

1. Use the following `vxprint` commands to discover the names of the snapshot volume's data change object (DCO) and DCO volume:

```
# DCONAME=`vxprint [-g diskgroup] -F%dco_name snapshot`
# DCOVOL=`vxprint [-g diskgroup] -F%log_vol $DCONAME`
```

2. Use the `vxassist mirror` command to create mirrors of the existing snapshot volume and its DCO volume:

```
# vxassist -g diskgroup mirror snapshot
# vxassist -g diskgroup mirror $DCOVOL
```

   The new plex in the DCO volume is required for use with the new data plex in the snapshot.

3. Use the `vxprint` command to find out the name of the additional snapshot plex:

```
# vxprint -g diskgroup snapshot
```

4. Use the `vxprint` command to find out the record ID of the additional DCO plex:

```
# vxprint -g diskgroup -F%rid $DCOVOL
```

5. Use the `vxedit` command to set the `dco_plex_rid` field of the new data plex to the name of the new DCO plex:

```
# vxedit -g diskgroup set dco_plex_rid=dco_plex_rid new_plex
```

   The new data plex is now ready to be used to perform a snapback operation.

Dissociating a snapshot volume

The link between a snapshot and its original volume can be permanently broken so that the snapshot volume becomes an independent volume. Use the following command to dissociate the snapshot volume, *snapshot*:

```
# vxassist snapclear snapshot
```

Displaying snapshot information

The `vxassist snapprint` command displays the associations between the original volumes and their respective replicas (snapshot copies):

```
# vxassist snapprint [volume]
```

Output from this command is shown in the following examples:

```
# vxassist -g mydg snapprint v1

V  NAME         USETYPE      LENGTH
SS SNAPOBJ      NAME         LENGTH     %DIRTY
DP NAME         VOLUME       LENGTH     %DIRTY

v  v1           fsgen        20480
ss SNAP-v1_snp  SNAP-v1      20480      4
dp v1-01        v1           20480      0
dp v1-02        v1           20480      0

v  SNAP-v1      fsgen        20480
ss v1_snp       v1           20480      0

# vxassist -g mydg snapprint v2

V  NAME         USETYPE      LENGTH
SS SNAPOBJ      NAME         LENGTH     %DIRTY
DP NAME         VOLUME       LENGTH     %DIRTY

v  v2           fsgen        20480
ss --           SNAP-v2      20480      0
dp v2-01        v2           20480      0

v  SNAP-v2      fsgen        20480
ss --           v2           20480      0
```

In this example, Persistent FastResync is enabled on volume `v1`, and Non-Persistent FastResync on volume `v2`. Lines beginning with `v`, `dp` and `ss` indicate a volume, detached plex and snapshot plex respectively. The `%DIRTY` field indicates the percentage of a snapshot plex or detached plex that is dirty with respect to the original volume. Notice that no snap objects are associated with volume `v2` or with its snapshot volume `SNAP-v2`.

See "How persistent FastResync works with snapshots" on page 72.

If a volume is specified, the `snapprint` command displays an error message if no FastResync maps are enabled for that volume.

Adding a version 0 DCO and DCO volume

The version 0 DCO log volume was introduced in VxVM 3.2. The version 0 layout supports traditional (third-mirror break-off) snapshots, but not full-sized or space-optimized instant snapshots.

See "Version 0 DCO volume layout" on page 70.

See "Version 20 DCO volume layout" on page 71.

See "Determining the DCO version number" on page 344.

To put Persistent FastResync into effect for a volume, a Data Change Object (DCO) and DCO volume must first be associated with that volume. When you have added a DCO object and DCO volume to a volume, you can then enable Persistent FastResync on the volume

See "Enabling FastResync on a volume" on page 361.

Note: You need a FastResync license to use the FastResync feature. Even if you do not have a license, you can configure a DCO object and DCO volume so that snap objects are associated with the original and snapshot volumes.

See "How persistent FastResync works with snapshots" on page 72.

To add a DCO object and DCO volume to an existing volume

1. Ensure that the disk group containing the existing volume has been upgraded to at least version 90. Use the following command to check the version of a disk group:

   ```
   # vxdg list diskgroup
   ```

   To upgrade a disk group to the latest version, use the following command:

   ```
   # vxdg upgrade diskgroup
   ```

   See "Upgrading a disk group" on page 264.

2. Use the following command to turn off Non-Persistent FastResync on the original volume if it is currently enabled:

   ```
   # vxvol [-g diskgroup] set fastresync=off volume
   ```

   If you are uncertain about which volumes have Non-Persistent FastResync enabled, use the following command to obtain a listing of such volumes:

   ```
   # vxprint [-g diskgroup] -F "%name" \
     -e "v_fastresync=on && !v_hasdcolog"
   ```

   Use the following command to add a DCO and DCO volume to the existing volume (which may already have dirty region logging (DRL) enabled):

   ```
   # vxassist [-g diskgroup] addlog volume logtype=dco \
     [ndcomirror=number] [dcolen=size] [storage_attributes]
   ```

   For non-layered volumes, the default number of plexes in the mirrored DCO volume is equal to the lesser of the number of plexes in the data volume or 2. For layered volumes, the default number of DCO plexes is always 2. If required, use the `ndcomirror` attribute to specify a different number. It is recommended that you configure as many DCO plexes as there are existing data and snapshot plexes in the volume. For example, specify `ndcomirror=3` when adding a DCO to a 3-way mirrored volume.

   The default size of each plex is 132 blocks. You can use the `dcolen` attribute to specify a different *size*. If specified, the size of the plex must be an integer multiple of 33 blocks from 33 up to a maximum of 2112 blocks.

   You can specify `vxassist`-style storage attributes to define the disks that can and/or cannot be used for the plexes of the DCO volume.

   See "Specifying storage for version 0 DCO plexes" on page 431.

Specifying storage for version 0 DCO plexes

If the disks that contain volumes and their snapshots are to be moved or split into different disk groups, the disks that contain their respective DCO plexes must be able to accompany them. By default, VxVM attempts to place version 0 DCO plexes on the same disks as the data plexes of the parent volume. However, this may be impossible if there is insufficient space available on those disks. In this case, VxVM uses any available space on other disks in the disk group. If the DCO plexes are placed on disks which are used to hold the plexes of other volumes, this may cause problems when you subsequently attempt to move volumes into other disk groups.

You can use storage attributes to specify explicitly which disks to use for the DCO plexes. If possible, specify the same disks as those on which the volume is configured.

For example, to add a DCO object and DCO volume with plexes on `mydg05` and `mydg06`, and a plex size of 264 blocks to the volume, `myvol`, in the disk group, `mydg`, use the following command:

```
# vxassist -g mydg addlog myvol logtype=dco dcolen=264 mydg05 mydg06
```

To view the details of the DCO object and DCO volume that are associated with a volume, use the `vxprint` command. The following is partial `vxprint` output for the volume named `vol1` (the TUTIL0 and PUTIL0 columns are omitted for clarity):

```
TY  NAME         ASSOC        KSTATE    LENGTH   PLOFFS   STATE  ...
v   vol1         fsgen        ENABLED   1024     -        ACTIVE
pl  vol1-01      vol1         ENABLED   1024     -        ACTIVE
sd  disk01-01    vol1-01      ENABLED   1024     0        -
pl  vol1-02      vol1         ENABLED   1024     -        ACTIVE
sd  disk02-01    vol1-02      ENABLED   1024     0        -
dc  vol1_dco     vol1         -         -        -        -
v   vol1_dcl     gen          ENABLED   132      -        ACTIVE
pl  vol1_dcl-01  vol1_dcl     ENABLED   132      -        ACTIVE
sd  disk03-01    vol1_dcl-01  ENABLED   132      0        -
pl  vol1_dcl-02  vol1_dcl     ENABLED   132      -        ACTIVE
sd  disk04-01    vol1_dcl-02  ENABLED   132        0        -
```

In this output, the DCO object is shown as `vol1_dco`, and the DCO volume as `vol1_dcl` with 2 plexes, `vol1_dcl-01` and `vol1_dcl-02`.

If required, you can use the `vxassist move` command to relocate DCO plexes to different disks. For example, the following command moves the plexes of the DCO volume, `vol1_dcl`, for volume `vol1` from `disk03` and `disk04` to `disk07` and `disk08`:

```
# vxassist -g mydg move vol1_dcl !disk03 !disk04 disk07 disk08
```

See "Moving DCO volumes between disk groups" on page 255.

See the vxassist(1M) manual page.

Removing a version 0 DCO and DCO volume

To dissociate a version 0 DCO object, DCO volume and any snap objects from a volume, use the following command:

```
# vxassist [-g diskgroup] remove log volume logtype=dco
```

This completely removes the DCO object, DCO volume and any snap objects. It also has the effect of disabling FastResync for the volume.

Alternatively, you can use the vxdco command to the same effect:

```
# vxdco [-g diskgroup] [-o rm] dis dco_obj
```

The default name of the DCO object, *dco_obj*, for a volume is usually formed by appending the string _dco to the name of the parent volume. To find out the name of the associated DCO object, use the vxprint command on the volume.

To dissociate, but not remove, the DCO object, DCO volume and any snap objects from the volume, myvol, in the disk group, mydg, use the following command:

```
# vxdco -g mydg dis myvol_dco
```

This form of the command dissociates the DCO object from the volume but does not destroy it or the DCO volume. If the -o rm option is specified, the DCO object, DCO volume and its plexes, and any snap objects are also removed.

Warning: Dissociating a DCO and DCO volume disables Persistent FastResync on the volume. A full resynchronization of any remaining snapshots is required when they are snapped back.

See the vxassist(1M) manual page.

See the vxdco(1M) manual pages.

Reattaching a version 0 DCO and DCO volume

If a version 0 DCO object and DCO volume are not removed by specifying the -o rm option to vxdco, they can be reattached to the parent volume using the following command:

```
# vxdco [-g diskgroup] att volume dco_obj
```

For example, to reattach the DCO object, `myvol_dco`, to the volume, `myvol`, use the following command:

```
# vxdco -g mydg att myvol myvol_dco
```

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

Chapter 10

Creating and administering volume sets

This chapter includes the following topics:

- About volume sets
- Creating a volume set
- Adding a volume to a volume set
- Listing details of volume sets
- Stopping and starting volume sets
- Removing a volume from a volume set
- Raw device node access to component volumes

About volume sets

Volume sets enable the use of the Multi-Volume Support and Dynamic Storage Tiering (DST) features 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.

Veritas Volume Manager (VxVM) provides the `vxvset` command to create and administer volume sets.

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 2048 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.
- Raw I/O from and to the component volumes of a volume set is supported under certain conditions.
  See "Raw device node access to component volumes" on page 439.
- Volume sets can be used in place of volumes with the following `vxsnap` operations on instant snapshots: `addmir`, `dis`, `make`, `prepare`, `reattach`, `refresh`, `restore`, `rmmir`, `split`, `syncpause`, `syncresume`, `syncstart`, `syncstop`, `syncwait`, and `unprepare`. The third-mirror break-off usage model for full-sized instant snapshots is supported for volume sets provided that sufficient plexes exist for each volume in the volume set.
  See "Creating instant snapshots of volume sets" on page 405.
- A full-sized snapshot of a volume set must itself be a volume set with the same number of volumes and the same volume index numbers as the parent. The corresponding volumes in the parent and snapshot volume sets are also subject to the same restrictions as apply between standalone volumes and their snapshots.

Creating a volume set

To create a volume set for use by Veritas File System (VxFS), use the following command:

```
# vxvset [-g diskgroup] -t vxfs make volset volume
```

Here *volset* is the name of the volume set, and *volume* is the name of the first volume in the volume set. The `-t` 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
```

Warning: The `-f` (force) option must be specified if the volume being added, or any volume in the volume set, is either a snapshot or the parent of a snapshot. Using this option can potentially cause inconsistencies in a snapshot hierarchy if any of the volumes involved in the operation is already in a snapshot chain.

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
```

When the final volume is removed, this deletes the volume set.

Warning: The `-f` (force) option must be specified if the volume being removed, or any volume in the volume set, is either a snapshot or the parent of a snapshot. Using this option can potentially cause inconsistencies in a snapshot hierarchy if any of the volumes involved in the operation is already in a snapshot chain.

Raw device node access to component volumes

To guard against accidental file system and data corruption, the device nodes of the component volumes are configured by default not to have raw and block entries in the `/dev/vx/rdsk/diskgroup` and `/dev/vx/dsk/diskgroup` directories. As a result, applications are prevented from directly reading from or writing to the component volumes of a volume set.

If some applications, such as the raw volume backup and restore feature of the Symantec NetBackup™ software, need to read from or write to the component volumes by accessing raw device nodes in the `/dev/vx/rdsk/diskgroup` directory, this is supported by specifying additional command-line options to the `vxvset` command. Access to the block device nodes of the component volumes of a volume set is unsupported.

Warning: Writing directly to or reading from the raw device node of a component volume of a volume set should only be performed if it is known that the volume's data will not otherwise change during the period of access.

All of the raw device nodes for the component volumes of a volume set can be created or removed in a single operation. Raw device nodes for any volumes added to a volume set are created automatically as required, and inherit the access mode of the existing device nodes.

Access to the raw device nodes for the component volumes can be configured to be read-only or read-write. This mode is shared by all the raw device nodes for the component volumes of a volume set. The read-only access mode implies that any writes to the raw device will fail, however writes using the `ioctl` interface or by VxFS to update metadata are not prevented. The read-write access mode allows direct writes via the raw device. The access mode to the raw device nodes of a volume set can be changed as required.

The presence of raw device nodes and their access mode is persistent across system reboots.

Note the following limitations of this feature:

- The disk group version must be 140 or greater.
- Access to the raw device nodes of the component volumes of a volume set is only supported for private disk groups; it is not supported for shared disk groups in a cluster.

Enabling raw device access when creating a volume set

To enable raw device access when creating a volume set, use the following form of the `vxvset make` command:

```
# vxvset [-g diskgroup] -o makedev=on \
  [-o compvol_access={read-only|read-write}] \
  [-o index] [-c "ch_addopt"] make vset vol [index]
```

The `-o makedev=on` option enables the creation of raw device nodes for the component volumes at the same time that the volume set is created. The default is setting is `off`.

If the `-o compvol_access=read-write` option is specified, direct writes are allowed to the raw device of each component volume. If the value is set to `read-only`, only reads are allowed from the raw device of each component volume.

If the `-o makedev=on` option is specified, but `-o compvol_access` is not specified, the default access mode is `read-only`.

If the `vxvset addvol` command is subsequently used to add a volume to a volume set, a new raw device node is created in */dev/vx/rdsk/diskgroup* if the value of the `makedev` attribute is currently set to `on`. The access mode is determined by the current setting of the `compvol_access` attribute.

The following example creates a volume set, `myvset1`, containing the volume, `myvol1`, in the disk group, `mydg`, with raw device access enabled in read-write mode:

```
# vxvset -g mydg -o makedev=on -o compvol_access=read-write \
  make myvset1 myvol1
```

Displaying the raw device access settings for a volume set

You can use the `vxprint -m` command to display the current settings for a volume set. If the `makedev` attribute is set to `on`, one of the following strings is displayed in the output:

`vset_devinfo=on:read-only`	Raw device nodes in read-only mode.
`vset_devinfo=on:read-write`	Raw device nodes in read-write mode.

A string is not displayed if `makedev` is set to `off`.

If the output from the `vxprint -m` command is fed to the `vxmake` command to recreate a volume set, the `vset_devinfo` attribute must set to `off`. Use the vxvset set command to re-enable raw device access with the desired access mode.

See "Controlling raw device access for an existing volume set" on page 441.

Controlling raw device access for an existing volume set

To enable or disable raw device node access for an existing volume set, use the following command:

```
# vxvset [-g diskgroup] [-f] set makedev={on|off} vset
```

The `makedev` attribute can be specified to the `vxvset set` command to create (`makedev=on`) or remove (`makedev=off`) the raw device nodes for the component volumes of a volume set. If any of the component volumes are open, the `-f` (force) option must be specified to set the attribute to `off`.

Specifying `makedev=off` removes the existing raw device nodes from the */dev/vx/rdsk/diskgroup* directory.

If the `makedev` attribute is set to `off`, and you use the `mknod` command to create the raw device nodes, you cannot read from or write to those nodes unless you set the value of `makedev` to `on`.

The syntax for setting the `compvol_access` attribute on a volume set is:

```
# vxvset [-g diskgroup] [-f] set \
  compvol_access={read-only|read-write} vset
```

The `compvol_access` attribute can be specified to the `vxvset set` command to change the access mode to the component volumes of a volume set. If any of the component volumes are open, the `-f` (force) option must be specified to set the attribute to `read-only`.

The following example sets the `makedev=on` and `compvol_access=read-only` attributes on a volume set, `myvset2`, in the disk group, `mydg`:

```
# vxvset -g mydg set makedev=on myvset2
```

The next example sets the `compvol_access=read-write` attribute on the volume set, `myvset2`:

```
# vxvset -g mydg set compvol_access=read-write myvset2
```

The final example removes raw device node access for the volume set, `myvset2`:

```
# vxvset -g mydg set makedev=off myvset2
```

Chapter 11

Configuring off-host processing

This chapter includes the following topics:

- About off-host processing solutions
- Implemention of off-host processing solutions

About off-host processing solutions

Off-host processing allows you to implement the following activities:

Data backup	As the requirement for 24 x 7 availability becomes essential for many businesses, organizations cannot afford the downtime involved in backing up critical data offline. By taking a snapshot of the data, and backing up from this snapshot, business-critical applications can continue to run without extended down time or impacted performance.
Decision support analysis and reporting	Because snapshots hold a point-in-time copy of a production database, a replica of the database can be set up using the snapshots. Operations such as decision support analysis and business reporting do not require access to up-to-the-minute information. This means that they can use a database copy that is running on a host other than the primary. When required, the database copy can quickly be synchronized with the data in the primary database.
Testing and training	Development or service groups can use snapshots as test data for new applications. Snapshot data provides developers, system testers and QA groups with a realistic basis for testing the robustness, integrity and performance of new applications.

Database error recovery	Logic errors caused by an administrator or an application program can compromise the integrity of a database. By restoring the database table files from a snapshot copy, the database can be recovered more quickly than by full restoration from tape or other backup media.

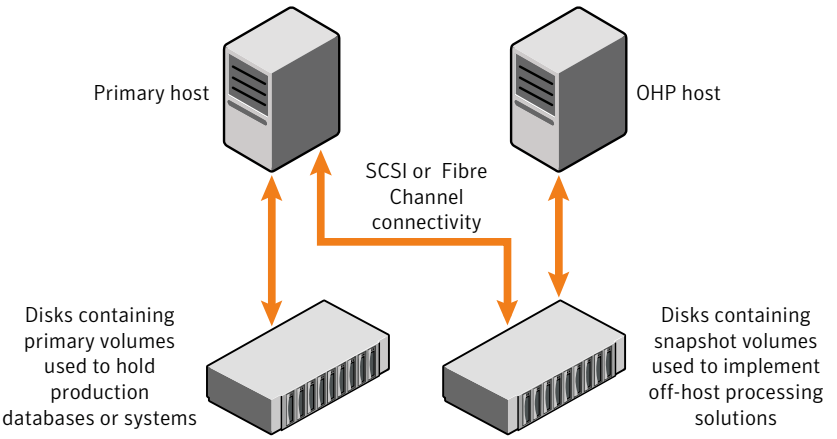
Off-host processing is made simpler by using linked break-off snapshots.

See "Linked break-off snapshot volumes" on page 380.

Implemention of off-host processing solutions

Figure 11-1 shows an example implementation of off-host processing.

Figure 11-1 Example implementation of off-host processing



By accessing snapshot volumes from a lightly loaded host (shown here as the OHP host), CPU- and I/O-intensive operations for online backup and decision support do not degrade the performance of the primary host that is performing the main production activity (such as running a database).If you also place the snapshot volumes on disks that are attached to different host controllers than the disks in the primary volumes, it is possible to avoid contending with the primary host for I/O resources.

The following sections describe how you can apply off-host processing to implement regular online backup of a volume in a private disk group, and to set up a replica of a production database for decision support. Two applications are outlined

See "Implementing off-host online backup" on page 445.

See "Implementing decision support" on page 449.

These applications use the Persistent FastResync feature of VxVM in conjunction with linked break-off snapshots.

A volume snapshot represents the data that exists in a volume at a given 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 in-memory data and the data 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.

Implementing off-host online backup

This section describes a procedure for implementing off-host online backup for a volume in a private disk group. The intention is to present an outline of how to set up a regular backup cycle. It is beyond the scope of this guide to describe how to configure a database to use this procedure, or how to perform the backup itself.

To back up a volume in a private disk group

1 Use the following command on the primary host to see if the volume is associated with a version 20 data change object (DCO) and DCO volume that allow instant snapshots and Persistent FastResync to be used with the volume:

```
# vxprint -g volumedg -F%instant volume
```

This command returns `on` if the volume can be used for instant snapshot operations; otherwise, it returns `off`.

2 Use the following command on the primary host to check whether FastResync is enabled on the volume:

```
# vxprint -g volumedg -F%fastresync volume
```

This command returns `on` if FastResync is enabled; otherwise, it returns `off`.

If FastResync is disabled, enable it using the following command on the primary host:

```
# vxvol -g volumedg set fastresync=on volume
```

3 On the primary host, create a new volume in a separate disk group for use as the snapshot volume.

See "Creating a volume for use as a full-sized instant or linked break-off snapshot" on page 393.

It is recommended that a snapshot disk group is dedicated to maintaining only those disks that are used for off-host processing.

4 On the primary host, link the snapshot volume in the snapshot disk group to the data volume:

```
# vxsnap -g volumedg -b addmir volume mirvol=snapvol \
  mirdg=snapvoldg
```

You can use the `vxsnap snapwait` command to wait for synchronization of the linked snapshot volume to complete:

```
# vxsnap -g volumedg snapwait volume mirvol=snapvol \
  mirdg=snapvoldg
```

This step sets up the snapshot volumes, and starts tracking changes to the original volumes.

When you are ready to create a backup, proceed to step 5.

5 On the primary host, suspend updates to the volume that contains the database tables. A database may have a hot backup mode that allows you to do this by temporarily suspending writes to its tables.

6 Create the snapshot volume, *snapvol*, by running the following command on the primary host:

```
# vxsnap -g volumedg make \
  source=volume/snapvol=snapvol/snapdg=snapvoldg
```

If a database spans more than one volume, you can specify all the volumes and their snapshot volumes using one command, as shown in this example:

```
# vxsnap -g dbasedg make \
  source=vol1/snapvol=snapvol1/snapdg=sdg \
  source=vol2/snapvol=snapvol2/snapdg=sdg \
  source=vol3/snapvol=snapvol3/snapdg=sdg
```

7 On the primary host, if you temporarily suspended updates to a volume in step 5, release all the database tables from hot backup mode.

8 On the primary host, deport the snapshot volume's disk group using the following command:

```
# vxdg deport snapvoldg
```

9 On the OHP host where the backup is to be performed, use the following command to import the snapshot volume's disk group:

```
# vxdg import snapvoldg
```

10 The snapshot volume is initially disabled following the join. Use the following commands on the OHP host to recover and restart the snapshot volume:

```
# vxrecover -g snapvoldg -m snapvol
# vxvol -g snapvoldg start snapvol
```

11 On the OHP host, back up the snapshot volume. If you need to remount the file system in the volume to back it up, first run `fsck` on the volume. The following are sample commands for checking and mounting a file system:

```
# fsck -F vxfs /dev/vx/rdsk/snapvoldg/snapvol
# mount -F vxfs /dev/vx/dsk/snapvoldg/snapvol mount_point
```

Back up the file system at this point, and then use the following command to unmount it.

```
# umount mount_point
```

12 On the OHP host, use the following command to deport the snapshot volume's disk group:

```
# vxdg deport snapvoldg
```

13 On the primary host, re-import the snapshot volume's disk group using the following command:

```
# vxdg import snapvoldg
```

14 The snapshot volume is initially disabled following the join. Use the following commands on the primary host to recover and restart the snapshot volume:

```
# vxrecover -g snapvoldg -m snapvol
# vxvol -g snapvoldg start snapvol
```

15 On the primary host, reattach the snapshot volume to its original volume using the following command:

```
# vxsnap -g snapvoldg reattach snapvol source=vol \
  sourcedg=volumedg
```

For example, to reattach the snapshot volumes `svol1`, `svol2` and `svol3`:

```
# vxsnap -g sdg reattach svol1 \
  source=vol1 sourcedg=dbasedg \
  svol2 source=vol2 sourcedg=dbasedg \
  svol3 source=vol3 sourcedg=dbasedg
```

You can use the `vxsnap snapwait` command to wait for synchronization of the linked snapshot volume to complete:

```
# vxsnap -g volumedg snapwait volume mirvol=snapvol
```

Repeat steps 5 through 15 each time that you need to back up the volume.

Implementing decision support

This section describes a procedure for implementing off-host decision support for a volume in a private disk group. The intention is to present an outline of how to set up a replica database. It is beyond the scope of this guide to describe how to configure a database to use this procedure.

To set up a replica database using the table files that are configured within a volume in a private disk group

1 Use the following command on the primary host to see if the volume is associated with a version 20 data change object (DCO) and DCO volume that allow instant snapshots and Persistent FastResync to be used with the volume:

```
# vxprint -g volumedg -F%instant volume
```

This command returns `on` if the volume can be used for instant snapshot operations; otherwise, it returns `off`.

If the volume was created under VxVM 4.0 or a later release, and it is not associated with a new-style DCO object and DCO volume, it must be prepared.

See "Preparing a volume for DRL and instant snapshots" on page 341.

If the volume was created before release 4.0 of VxVM, and has any attached snapshot plexes, or is associated with any snapshot volumes, it must be upgraded.

See "Upgrading existing volumes to use version 20 DCOs" on page 347.

2 Use the following command on the primary host to check whether FastResync is enabled on a volume:

```
# vxprint -g volumedg -F%fastresync volume
```

This command returns `on` if FastResync is enabled; otherwise, it returns `off`.

If FastResync is disabled, enable it using the following command on the primary host:

```
# vxvol -g volumedg set fastresync=on volume
```

3 Prepare the OHP host to receive the snapshot volume that contains the copy of the database tables. This may involve setting up private volumes to contain any redo logs, and configuring any files that are used to initialize the database.

4 On the primary host, create a new volume in a separate disk group for use as the snapshot volume.

See "Creating a volume for use as a full-sized instant or linked break-off snapshot" on page 393.

It is recommended that a snapshot disk group is dedicated to maintaining only those disks that are used for off-host processing.

5 On the primary host, link the snapshot volume in the snapshot disk group to the data volume:

```
# vxsnap -g volumedg -b addmir volume mirvol=snapvol \
  mirdg=snapvoldg
```

You can use the `vxsnap snapwait` command to wait for synchronization of the linked snapshot volume to complete:

```
# vxsnap -g volumedg snapwait volume mirvol=snapvol \
  mirdg=snapvoldg
```

This step sets up the snapshot volumes, and starts tracking changes to the original volumes.

When you are ready to create a replica database, proceed to 6.

6 On the primary host, suspend updates to the volume that contains the database tables. A database may have a hot backup mode that allows you to do this by temporarily suspending writes to its tables.

7 Create the snapshot volume, *snapvol*, by running the following command on the primary host:

```
# vxsnap -g volumedg make \
  source=volume/snapvol=snapvol/snapdg=snapvoldg
```

If a database spans more than one volume, you can specify all the volumes and their snapshot volumes using one command, as shown in this example:

```
# vxsnap -g dbasedg make \
  source=vol1/snapvol=snapvol1/snapdg=sdg \
  source=vol2/snapvol=snapvol2/snapdg=sdg \
  source=vol3/snapvol=snapvol3/snapdg=sdg
```

This step sets up the snapshot volumes ready for the backup cycle, and starts tracking changes to the original volumes.

8 On the primary host, if you temporarily suspended updates to a volume in step 6, release all the database tables from hot backup mode.

9 On the primary host, deport the snapshot volume's disk group using the following command:

```
# vxdg deport snapvoldg
```

10 On the OHP host where the replica database is to be set up, use the following command to import the snapshot volume's disk group:

```
# vxdg import snapvoldg
```

11 The snapshot volume is initially disabled following the join. Use the following commands on the OHP host to recover and restart the snapshot volume:

```
# vxrecover -g snapvoldg -m snapvol
# vxvol -g snapvoldg start snapvol
```

12 On the OHP host, check and mount the snapshot volume. The following are sample commands for checking and mounting a file system:

```
# fsck -F vxfs /dev/vx/rdsk/snapvoldg/snapvol
# mount -F vxfs /dev/vx/dsk/snapvoldg/snapvol mount_point
```

13 On the OHP host, use the appropriate database commands to recover and start the replica database for its decision support role.

At a later time, you can resynchronize the snapshot volume' s data with the primary database.

To refresh the snapshot plexes from the original volume

1 On the OHP host, shut down the replica database, and use the following command to unmount the snapshot volume:

```
# umount mount_point
```

2 On the OHP host, use the following command to deport the snapshot volume's disk group:

```
# vxdg deport snapvoldg
```

3 On the primary host, re-import the snapshot volume's disk group using the following command:

```
# vxdg import snapvoldg
```

4 The snapshot volume is initially disabled following the join. Use the following commands on the primary host to recover and restart the snapshot volume:

```
# vxrecover -g snapvoldg -m snapvol
# vxvol -g snapvoldg start snapvol
```

5 On the primary host, reattach the snapshot volume to its original volume using the following command:

```
# vxsnap -g snapvoldg reattach snapvol source=vol \
  sourcedg=volumedg
```

For example, to reattach the snapshot volumes `svol1`, `svol2` and `svol3`:

```
# vxsnap -g sdg reattach svol1 \
  source=vol1 sourcedg=dbasedg \
  svol2 source=vol2 sourcedg=dbasedg \
  svol3 source=vol3 sourcedg=dbasedg
```

You can use the `vxsnap snapwait` command to wait for synchronization of the linked snapshot volume to complete:

```
# vxsnap -g volumedg snapwait volume mirvol=snapvol
```

You can then resume the procedure from step 6.

Chapter 12

Administering hot-relocation

This chapter includes the following topics:

- About hot-relocation
- How hot-relocation works
- Configuring a system for hot-relocation
- Displaying spare disk information
- Marking a disk as a hot-relocation spare
- Removing a disk from use as a hot-relocation spare
- Excluding a disk from hot-relocation use
- Making a disk available for hot-relocation use
- Configuring hot-relocation to use only spare disks
- Moving relocated subdisks
- Modifying the behavior of hot-relocation

About hot-relocation

If a volume has a disk I/O failure (for example, the disk has an uncorrectable error), Veritas Volume Manager (VxVM) can detach the plex involved in the failure. I/O stops on that plex but continues on the remaining plexes of the volume.

If a disk fails completely, VxVM can detach the disk from its disk group. All plexes on the disk are disabled. If there are any unmirrored volumes on a disk when it is detached, those volumes are also disabled.

Apparent disk failure may not be due to a fault in the physical disk media or the disk controller, but may instead be caused by a fault in an intermediate or ancillary component such as a cable, host bus adapter, or power supply.

The hot-relocation feature in VxVM automatically detects disk failures, and notifies the system administrator and other nominated users of the failures by electronic mail. Hot-relocation also attempts to use spare disks and free disk space to restore redundancy and to preserve access to mirrored and RAID-5 volumes.

See "How hot-relocation works" on page 456.

If hot-relocation is disabled or you miss the electronic mail, you can use the `vxprint` command or the graphical user interface to examine the status of the disks. You may also see driver error messages on the console or in the system messages file.

Failed disks must be removed and replaced manually.

See "Removing and replacing disks" on page 144.

For more information about recovering volumes and their data after hardware failure, see the *Veritas Volume Manager Troubleshooting Guide*.

How hot-relocation works

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

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

Hot-relocation is only performed for redundant (mirrored or RAID-5) subdisks on a failed disk. Non-redundant subdisks on a failed disk are not relocated, but the system administrator is notified of their failure.

Hot-relocation is enabled by default and takes effect without the intervention of the system administrator when a failure occurs.

The hot-relocation daemon, `vxrelocd`, detects and reacts to VxVM events that signify the following types of failures:

Disk failure	This is normally detected as a result of an I/O failure from a VxVM object. VxVM attempts to correct the error. If the error cannot be corrected, VxVM tries to access configuration information in the private region of the disk. If it cannot access the private region, it considers the disk failed.
Plex failure	This is normally detected as a result of an uncorrectable I/O error in the plex (which affects subdisks within the plex). For mirrored volumes, the plex is detached.
RAID-5 subdisk failure	This is normally detected as a result of an uncorrectable I/O error. The subdisk is detached.

When `vxrelocd` detects such a failure, it performs the following steps:

- `vxrelocd` informs the system administrator (and other nominated users) by electronic mail of the failure and which VxVM objects are affected.
  See Partial disk failure mail messages.
  See Complete disk failure mail messages.
  See Modifying the behavior of hot-relocation.
- `vxrelocd` next determines if any subdisks can be relocated. `vxrelocd` looks for suitable space on disks that have been reserved as hot-relocation spares (marked `spare`) in the disk group where the failure occurred. It then relocates the subdisks to use this space.
- If no spare disks are available or additional space is needed, `vxrelocd` uses free space on disks in the same disk group, except those disks that have been excluded for hot-relocation use (marked `nohotuse`). When `vxrelocd` has relocated the subdisks, it reattaches each relocated subdisk to its plex.
- Finally, `vxrelocd` initiates appropriate recovery procedures. For example, recovery includes mirror resynchronization for mirrored volumes or data recovery for RAID-5 volumes. It also notifies the system administrator of the hot-relocation and recovery actions that have been taken.

If relocation is not possible, `vxrelocd` notifies the system administrator and takes no further action.

Warning: Hot-relocation does not guarantee the same layout of data or the same performance after relocation. An administrator should check whether any configuration changes are required after hot-relocation occurs.

Relocation of failing subdisks is not possible in the following cases:

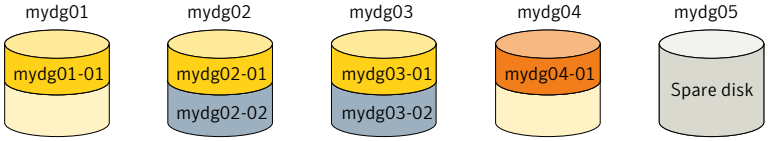
- The failing subdisks are on non-redundant volumes (that is, volumes of types other than mirrored or RAID-5).
- There are insufficient spare disks or free disk space in the disk group.
- The only available space is on a disk that already contains a mirror of the failing plex.
- The only available space is on a disk that already contains the RAID-5 log plex or one of its healthy subdisks. Failing subdisks in the RAID-5 plex cannot be relocated.
- If a mirrored volume has a dirty region logging (DRL) log subdisk as part of its data plex, failing subdisks belonging to that plex cannot be relocated.
- If a RAID-5 volume log plex or a mirrored volume DRL log plex fails, a new log plex is created elsewhere. There is no need to relocate the failed subdisks of the log plex.

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

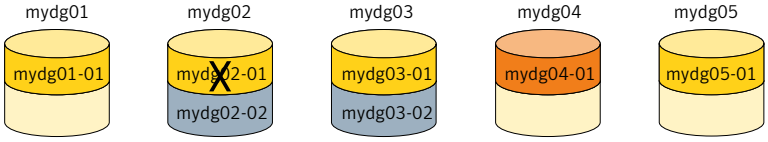
Figure 12-1 shows the hot-relocation process in the case of the failure of a single subdisk of a RAID-5 volume.

Figure 12-1 Example of hot-relocation for a subdisk in a RAID-5 volume

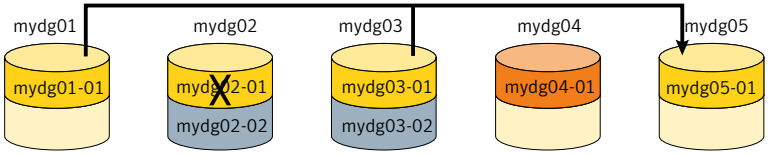
a Disk group contains five disks. Two RAID-5 volumes are configured across four of the disks. One spare disk is availavle for hot-relocation.



b Subdisk mydg02-01 in one RAID-5 volume fails. Hot-relocation replaces it with subdisk mydg05-01 that it has created on the spare disk, and then initiates recovery on the RAID-5 volume.



c RAID-5 recovery recreates subdisk mydg02-01's data and parity on subdisk mygd05-01 from the data and parity information remaining on subdisks mydg01-01 and mydg03-01.



Partial disk failure mail messages

If hot-relocation is enabled when a plex or disk is detached by a failure, mail indicating the failed objects is sent to `root`. If a partial disk failure occurs, the mail identifies the failed plexes. For example, if a disk containing mirrored volumes fails, you can receive mail information as shown in the following example:

```
To: root
Subject: Volume Manager failures on host teal
Failures have been detected by the Veritas Volume Manager:

failed plexes:
home-02
src-02
```

Mail can be sent to users other than `root`.

See "Modifying the behavior of hot-relocation" on page 473.

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 144.

Complete disk failure mail messages

If a disk fails completely and hot-relocation is enabled, the mail message lists the disk that failed and all plexes that use the disk. For example, you can receive mail as shown in this example display:

```
To: root
Subject: Volume Manager failures on host teal
```

```
Failures have been detected by the Veritas Volume Manager:

failed disks:
mydg02

failed plexes:
home-02
src-02
mkting-01

failing disks:
mydg02
```

This message shows that `mydg02` was detached by a failure. When a disk is detached, I/O cannot get to that disk. The plexes `home-02`, `src-02`, and `mkting-01` were also detached (probably because of the failure of the disk).

One possible cause of the problem could be a cabling error.

See "Partial disk failure mail messages" on page 459.

If the problem is not a cabling error, replace the disk.

See "Removing and replacing disks" on page 144.

How space is chosen for relocation

A spare disk must be initialized and placed in a disk group as a spare before it can be used for replacement purposes. If no disks have been designated as spares when a failure occurs, VxVM automatically uses any available free space in the disk group in which the failure occurs. If there is not enough spare disk space, a combination of spare space and free space is used.

The free space used in hot-relocation must not have been excluded from hot-relocation use. Disks can be excluded from hot-relocation use by using `vxdiskadm`, `vxedit` or the Veritas Enterprise Administrator (VEA).

You can designate one or more disks as hot-relocation spares within each disk group. Disks can be designated as spares by using `vxdiskadm`, `vxedit`, or the VEA. Disks designated as spares do not participate in the free space model and should not have storage space allocated on them.

When selecting space for relocation, hot-relocation preserves the redundancy characteristics of the VxVM object to which the relocated subdisk belongs. For example, hot-relocation ensures that subdisks from a failed plex are not relocated to a disk containing a mirror of the failed plex. If redundancy cannot be preserved

using any available spare disks and/or free space, hot-relocation does not take place. If relocation is not possible, the system administrator is notified and no further action is taken.

From the eligible disks, hot-relocation attempts to use the disk that is “closest” to the failed disk. The value of “closeness” depends on the controller, target, and disk number of the failed disk. A disk on the same controller as the failed disk is closer than a disk on a different controller. A disk under the same target as the failed disk is closer than one on a different target.

Hot-relocation tries to move all subdisks from a failing drive to the same destination disk, if possible.

If the failing disk is a root disk, hot-relocation only works if it can relocate all of the file systems to the same disk. If none are found, the system administrator is notified through email.

When hot-relocation takes place, the failed subdisk is removed from the configuration database, and VxVM ensures that the disk space used by the failed subdisk is not recycled as free space.

Configuring a system for hot-relocation

By designating spare disks and making free space on disks available for use by hot relocation, you can control how disk space is used for relocating subdisks in the event of a disk failure. If the combined free space and space on spare disks is not sufficient or does not meet the redundancy constraints, the subdisks are not relocated.

Find out which disks are spares or are excluded from hot-relocation.

See “Displaying spare disk information” on page 463.

You can prepare for hot-relocation by designating one or more disks per disk group as hot-relocation spares.

See “Marking a disk as a hot-relocation spare” on page 463.

If required, you can remove a disk from use as a hot-relocation spare

See “Removing a disk from use as a hot-relocation spare” on page 464.

If no spares are available at the time of a failure or if there is not enough space on the spares, free space on disks in the same disk group as where the failure occurred is automatically used, unless it has been excluded from hot-relocation use.

See “Excluding a disk from hot-relocation use” on page 465.

See “Making a disk available for hot-relocation use” on page 466.

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 467.

After a successful relocation, remove and replace the failed disk.

See "Removing and replacing disks" on page 144.

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.

To use vxdiskadm to designate a disk as a hot-relocation spare

1. Select `Mark a disk as a spare for a disk group` from the `vxdiskadm` main menu.
2. At the following prompt, enter a disk media name (such as `mydg01`):

   ```
   Enter disk name [<disk>,list,q,?] mydg01
   ```

   The following notice is displayed when the disk has been marked as spare:

   ```
   VxVM NOTICE V-5-2-219 Marking of mydg01 in mydg as a spare disk
   is complete.
   ```

3. At the following prompt, indicate whether you want to add more disks as spares (`y`) or return to the `vxdiskadm` main menu (`n`):

   ```
   Mark another disk as a spare? [y,n,q,?] (default: n)
   ```

   Any VM disk in this disk group can now use this disk as a spare in the event of a failure. If a disk fails, hot-relocation should automatically occur (if possible). You should be notified of the failure and relocation through electronic mail. After successful relocation, you may want to replace the failed disk.

Removing a disk from use as a hot-relocation spare

While a disk is designated as a spare, the space on that disk is not used for the creation of VxVM objects within its disk group. If necessary, you can free a spare disk for general use by removing it from the pool of hot-relocation disks.

To remove a spare from the hot-relocation pool, use the following command:

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

where *diskname* is the disk media name.

For example, to make `mydg01` available for normal use in the disk group, `mydg`, use the following command:

```
# vxedit -g mydg set spare=off mydg01
```

To use vxdiskadm to remove a disk from the hot-relocation pool

1. Select `Turn off the spare flag on a disk` from the `vxdiskadm` main menu.
2. At the following prompt, enter the disk media name of a spare disk (such as `mydg01`):

   ```
   Enter disk name [<disk>,list,q,?] mydg01
   ```

   The following confirmation is displayed:

   ```
   VxVM NOTICE V-5-2-143 Disk mydg01 in mydg no longer marked as
   a spare disk.
   ```

3. At the following prompt, indicate whether you want to disable more spare disks (`y`) or return to the `vxdiskadm` main menu (`n`):

   ```
   Turn off spare flag on another disk? [y,n,q,?] (default: n)
   ```

Excluding a disk from hot-relocation use

To exclude a disk from hot-relocation use, use the following command:

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

where *diskname* is the disk media name.

To use vxdiskadm to exclude a disk from hot-relocation use

1. Select `Exclude a disk from hot-relocation use` from the `vxdiskadm` main menu.
2. At the following prompt, enter the disk media name (such as `mydg01`):

   ```
   Enter disk name [<disk>,list,q,?] mydg01
   ```

   The following confirmation is displayed:

   ```
   VxVM INFO V-5-2-925 Excluding mydg01 in mydg from hot-
   relocation use is complete.
   ```

3. At the following prompt, indicate whether you want to add more disks to be excluded from hot-relocation (`y`) or return to the `vxdiskadm` main menu (`n`):

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

Making a disk available for hot-relocation use

Free space is used automatically by hot-relocation in case spare space is not sufficient to relocate failed subdisks. You can limit this free space usage by hot-relocation by specifying which free disks should not be touched by hot-relocation. If a disk was previously excluded from hot-relocation use, you can undo the exclusion and add the disk back to the hot-relocation pool.

To make a disk available for hot-relocation use, use the following command:

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

To use vxdiskadm to make a disk available for hot-relocation use

1. Select `Make a disk available for hot-relocation use` from the `vxdiskadm` main menu.
2. At the following prompt, enter the disk media name (such as `mydg01`):

   ```
   Enter disk name [<disk>,list,q,?] mydg01
   ```

   The following confirmation is displayed:

   ```
   V-5-2-932 Making mydg01 in mydg available for hot-relocation
   use is complete.
   ```

3. At the following prompt, indicate whether you want to add more disks to be excluded from hot-relocation (`y`) or return to the `vxdiskadm` main menu (`n`):

   ```
   Make another disk available for hot-relocation use? [y,n,q,?]
   (default: n)
   ```

Configuring hot-relocation to use only spare disks

If you want VxVM to use only spare disks for hot-relocation, add the following line to the file `/etc/default/vxassist`:

```
spare=only
```

If not enough storage can be located on disks marked as spare, the relocation fails. Any free space on non-spare disks is not used.

Moving relocated subdisks

When hot-relocation occurs, subdisks are relocated to spare disks and/or available free space within the disk group. The new subdisk locations may not provide the same performance or data layout that existed before hot-relocation took place. You can move the relocated subdisks (after hot-relocation is complete) to improve performance.

You can also move the relocated subdisks 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.

See "Removing and replacing disks" on page 144.

Once this is done, you can move a relocated subdisk back to the original disk as described in the following sections.

Warning: During subdisk move operations, RAID-5 volumes are not redundant.

Moving relocated subdisks using vxdiskadm

When a disk has replaced following a failure, you can use the `vxdiskadm` command move the hot-relocated subdisks back to the disk where they originally resided.

To move the relocated subdisks using vxdiskadm

1. Select `Unrelocate subdisks back to a disk` from the `vxdiskadm` main menu.
2. This option prompts for the original disk media name first.

   Enter the disk media name where the hot-relocated subdisks originally resided at the following prompt:

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

   If there are no hot-relocated subdisks in the system, `vxdiskadm` displays `Currently there are no hot-relocated disks`, and asks you to press Return to continue.

3 You are next asked if you want to move the subdisks to a destination disk other than the original disk.

```
Unrelocate to a new disk [y,n,q,?] (default: n)
```

4 If moving subdisks to their original offsets is not possible, you can choose to unrelocate the subdisks forcibly to the specified disk, but not necessarily to the same offsets.

```
Use -f option to unrelocate the subdisks if moving to the exact
offset fails? [y,n,q,?] (default: n)
```

5 If you entered `y` at step 4 to unrelocate the subdisks forcibly, enter `y` or press Return at the following prompt to confirm the operation:

```
Requested operation is to move all the subdisks which were
hot-relocated from mydg10 back to mydg10 of disk group mydg.
Continue with operation? [y,n,q,?] (default: y)
```

A status message is displayed at the end of the operation.

```
VxVM INFO V-5-2-954 Unrelocate to disk mydg10 is complete.
```

As an alternative to this procedure, use either the `vxassist` command or the `vxunreloc` command directly.

See "Moving relocated subdisks using vxassist" on page 469.

See "Moving relocated subdisks using vxunreloc" on page 470.

Moving relocated subdisks using vxassist

You can use the `vxassist` command to move and unrelocate subdisks. For example, to move the relocated subdisks on `mydg05` belonging to the volume `home` back to `mydg02`, enter the following command:

```
# vxassist -g mydg move home !mydg05 mydg02
```

Here, `!mydg05` specifies the current location of the subdisks, and `mydg02` specifies where the subdisks should be relocated.

If the volume is enabled, subdisks within detached or disabled plexes, and detached log or RAID-5 subdisks, are moved without recovery of data.

If the volume is not enabled, subdisks within STALE or OFFLINE plexes, and stale log or RAID-5 subdisks, are moved without recovery. If there are other subdisks within a non-enabled volume that require moving, the relocation fails.

For enabled subdisks in enabled plexes within an enabled volume, data is moved to the new location, without loss of either availability or redundancy of the volume.

Moving relocated subdisks using vxunreloc

VxVM hot-relocation allows the system to automatically react to I/O failures on a redundant VxVM object at the subdisk level and then take necessary action to make the object available again. This mechanism detects I/O failures in a subdisk, relocates the subdisk, and recovers the plex associated with the subdisk. After the disk has been replaced, `vxunreloc` allows you to restore the system back to the configuration that existed before the disk failure. `vxunreloc` allows you to move the hot-relocated subdisks back onto a disk that was replaced due to a failure.

When `vxunreloc` is invoked, you must specify the disk media name where the hot-relocated subdisks originally resided. When `vxunreloc` moves the subdisks, it moves them to the original offsets. If you try to unrelocate to a disk that is smaller than the original disk that failed,`vxunreloc` does nothing except return an error.

vxunreloc provides an option to move the subdisks to a different disk from where they were originally relocated. It also provides an option to unrelocate subdisks to a different offset as long as the destination disk is large enough to accommodate all the subdisks.

If `vxunreloc` cannot replace the subdisks back to the same original offsets, a force option is available that allows you to move the subdisks to a specified disk without using the original offsets.

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

The examples in the following sections demonstrate the use of `vxunreloc`.

Moving hot-relocated subdisks back to their original disk

Assume that `mydg01` failed and all the subdisks were relocated. After `mydg01` is replaced, `vxunreloc` can be used to move all the hot-relocated subdisks back to `mydg01`.

```
# vxunreloc -g mydg mydg01
```

Moving hot-relocated subdisks back to a different disk

The `vxunreloc` utility provides the `-n` option to move the subdisks to a different disk from where they were originally relocated.

Assume that mydg01 failed, and that all of the subdisks that resided on it were hot-relocated to other disks. vxunreloc provides an option to move the subdisks to a different disk from where they were originally relocated. After the disk is repaired, it is added back to the disk group using a different name, for example, mydg05. If you want to move all the hot-relocated subdisks back to the new disk, the following command can be used:

```
# vxunreloc -g mydg -n mydg05 mydg01
```

The destination disk should have at least as much storage capacity as was in use on the original disk. If there is not enough space, the unrelocate operation will fail and none of the subdisks will be moved.

Forcing hot-relocated subdisks to accept different offsets

By default, vxunreloc attempts to move hot-relocated subdisks to their original offsets. However, vxunreloc fails if any subdisks already occupy part or all of the area on the destination disk. In such a case, you have two choices:

- Move the existing subdisks somewhere else, and then re-run vxunreloc.
- Use the -f option provided by vxunreloc to move the subdisks to the destination disk, but leave it to vxunreloc to find the space on the disk. As long as the destination disk is large enough so that the region of the disk for storing subdisks can accommodate all subdisks, all the hot-relocated subdisks will be unrelocated without using the original offsets.

Assume that mydg01 failed and the subdisks were relocated and that you want to move the hot-relocated subdisks to mydg05 where some subdisks already reside. You can use the force option to move the hot-relocated subdisks to mydg05, but not to the exact offsets:

```
# vxunreloc -g mydg -f -n mydg05 mydg01
```

Examining which subdisks were hot-relocated from a disk

If a subdisk was hot relocated more than once due to multiple disk failures, it can still be unrelocated back to its original location. For instance, if mydg01 failed and a subdisk named mydg01-01 was moved to mydg02, and then mydg02 experienced disk failure, all of the subdisks residing on it, including the one which was hot-relocated to it, will be moved again. When mydg02 was replaced, a vxunreloc operation for mydg02 will do nothing to the hot-relocated subdisk mydg01-01. However, a replacement of mydg01 followed by a vxunreloc operation, moves mydg01-01 back to mydg01 if vxunreloc is run immediately after the replacement.

After the disk that experienced the failure is fixed or replaced, vxunreloc can be used to move all the hot-relocated subdisks back to the disk. When a subdisk is hot-relocated, its original disk-media name and the offset into the disk are saved in the configuration database. When a subdisk is moved back to the original disk or to a new disk using vxunreloc, the information is erased. The original disk-media name and the original offset are saved in the subdisk records. To print all of the subdisks that were hot-relocated from mydg01 in the mydg disk group, use the following command:

```
# vxprint -g mydg -se 'sd_orig_dmname="mydg01"'
```

Restarting vxunreloc after errors

vxunreloc moves subdisks in three phases:

- vxunreloc creates as many subdisks on the specified destination disk as there are subdisks to be unrelocated. The string UNRELOC is placed in the comment field of each subdisk record.
  Creating the subdisk is an all-or-nothing operation. If vxunreloc cannot create all the subdisks successfully, none are created, and vxunreloc exits.
- vxunreloc moves the data from each subdisk to the corresponding newly created subdisk on the destination disk.
- When all subdisk data moves have been completed successfully, vxunreloc sets the comment field to the null string for each subdisk on the destination disk whose comment field is currently set to UNRELOC.

The comment fields of all the subdisks on the destination disk remain marked as UNRELOC until phase 3 completes. If its execution is interrupted, vxunreloc can subsequently re-use subdisks that it created on the destination disk during a previous execution, but it does not use any data that was moved to the destination disk.

If a subdisk data move fails, vxunreloc displays an error message and exits. Determine the problem that caused the move to fail, and fix it before re-executing vxunreloc.

If the system goes down after the new subdisks are created on the destination disk, but before all the data has been moved, re-execute vxunreloc when the system has been rebooted.

Warning: Do not modify the string UNRELOC in the comment field of a subdisk record.

Modifying the behavior of hot-relocation

Hot-relocation is turned on as long as the `vxrelocd` process is running. You should normally leave hot-relocation turned on so that you can take advantage of this feature if a failure occurs. However, if you choose to disable hot-relocation (perhaps because you do not want the free space on your disks to be used for relocation), you can prevent `vxrelocd` from starting at system startup time by editing the startup file that invokes `vxrelocd`. This file is `/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:

1. To prevent `vxrelocd` starting, comment out the entry that invokes it in the startup file:

   ```
   # nohup vxrelocd root &
   ```

2. By default, `vxrelocd` sends electronic mail to `root` when failures are detected and relocation actions are performed. You can instruct `vxrelocd` to notify additional users by adding the appropriate user names as shown here:

   ```
   nohup vxrelocd root user1 user2 &
   ```

3. To reduce the impact of recovery on system performance, you can instruct `vxrelocd` to increase the delay between the recovery of each region of the volume, as shown in the following example:

   ```
   nohup vxrelocd -o slow[=IOdelay] root &
   ```

   where the optional *IOdelay* value indicates the desired delay in milliseconds. The default value for the delay is 250 milliseconds.

   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.

Chapter 13

Administering cluster functionality

This chapter includes the following topics:

- About the cluster functionality of VxVM
- Overview of cluster volume management
- Cluster initialization and configuration
- Dirty region logging in cluster environments
- Multiple host failover configurations
- Administering VxVM in cluster environments

About the cluster functionality of VxVM

A cluster consists of a number of hosts or nodes that share a set of disks. The following are the main benefits of cluster configurations:

Availability	If one node fails, the other nodes can still access the shared disks. When configured with suitable software, mission-critical applications can continue running by transferring their execution to a standby node in the cluster. This ability to provide continuous uninterrupted service by switching to redundant hardware is commonly termed failover. Failover is transparent to users and high-level applications for database and file-sharing. You must configure cluster management software, such as Veritas Cluster Server (VCS), to monitor systems and services, and to restart applications on another node in the event of either hardware or software failure. VCS also allows you to perform general administration tasks such as making nodes join or leave a cluster.
Off-host processing	Clusters can reduce contention for system resources by performing activities such as backup, decision support and report generation on the more lightly loaded nodes of the cluster. This allows businesses to derive enhanced value from their investment in cluster systems.

The cluster functionality of Veritas Volume Manager (CVM) allows up to 16 nodes in a cluster to simultaneously access and manage a set of disks under VxVM control (VM disks). The same logical view of disk configuration and any changes to this is available on all the nodes. When the cluster functionality is enabled, all the nodes in the cluster can share VxVM objects. This chapter discusses the cluster functionality that is provided with VxVM.

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

Products such as Veritas Storage Foundation Cluster File System (SFCFS), and Veritas Cluster Server (VCS) are separately licensed, and are not included with Veritas Volume Manager. See the documentation provided with those products for more information about them.

The Dynamic Multipathing (DMP) feature of VxVM can be used in a clustered environment.

See “DMP in a clustered environment” on page 169.

Campus cluster configurations (also known as stretch cluster or remote mirror configurations) can also be configured and administered.

See “About sites and remote mirrors” on page 511.

Overview of cluster volume management

In recent years, tightly-coupled cluster systems have become increasingly popular in the realm of enterprise-scale mission-critical data processing. The primary advantage of clusters is protection against hardware failure. Should the primary node fail or otherwise become unavailable, applications can continue to run by transferring their execution to standby nodes in the cluster. This ability to provide continuous availability of service by switching to redundant hardware is commonly termed failover.

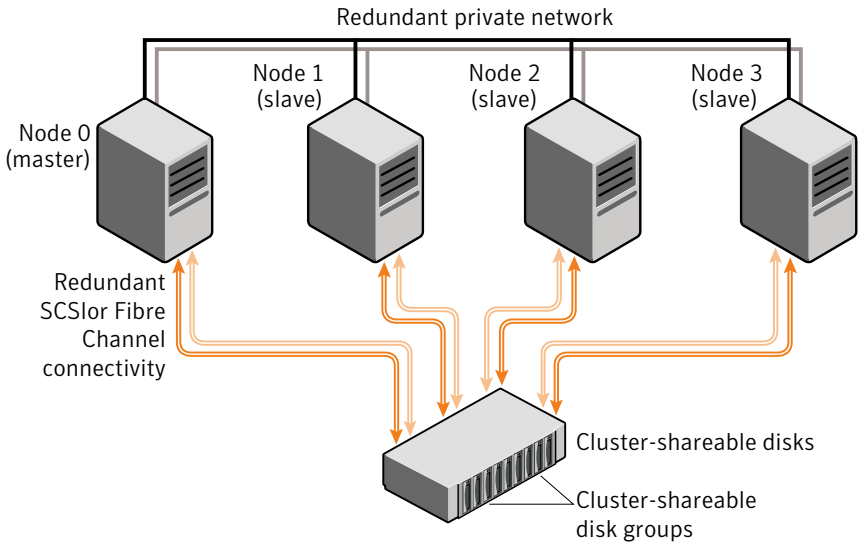
Another major advantage of clustered systems is their ability to reduce contention for system resources caused by activities such as backup, decision support and report generation. Businesses can derive enhanced value from their investment in cluster systems by performing such operations on lightly loaded nodes in the cluster rather than on the heavily loaded nodes that answer requests for service. This ability to perform some operations on the lightly loaded nodes is commonly termed load balancing.

The cluster functionality of VxVM works together with the cluster monitor daemon that is provided by VCS or by the host operating system. When configured correctly, the cluster monitor informs VxVM of changes in cluster membership. Each node starts up independently and has its own cluster monitor plus its own copies of the operating system and VxVM with support for cluster functionality. When a node joins a cluster, it gains access to shared disk groups and volumes. When a node leaves a cluster, it no longer has access to these shared objects. A node joins a cluster when the cluster monitor is started on that node.

Warning: The cluster functionality of VxVM is supported only when used in conjunction with a cluster monitor that has been configured correctly to work with VxVM.

Figure 13-1 shows a simple cluster arrangement consisting of four nodes with similar or identical hardware characteristics (CPUs, RAM and host adapters), and configured with identical software (including the operating system).

Figure 13-1 Example of a 4-node cluster



Node 0 is configured as the master node and nodes 1, 2 and 3 are configured as slave nodes. The nodes are fully connected by a private network and they are also separately connected to shared external storage (either disk arrays or JBODs: just a bunch of disks) via SCSI or Fibre Channel.

In this example, each node has two independent paths to the disks, which are configured in one or more cluster-shareable disk groups. Multiple paths provide resilience against failure of one of the paths, but this is not a requirement for cluster configuration. Disks may also be connected by single paths.

The private network allows the nodes to share information about system resources and about each other's state. Using the private network, any node can recognize which other nodes are currently active, which are joining or leaving the cluster, and which have failed. The private network requires at least two communication channels to provide redundancy against one of the channels failing. If only one channel were used, its failure would be indistinguishable from node failure—a condition known as network partitioning.

To the cluster monitor, all nodes are the same. VxVM objects configured within shared disk groups can potentially be accessed by all nodes that join the cluster. However, the cluster functionality of VxVM requires that one node act as the master node; all other nodes in the cluster are slave nodes. Any node is capable of being the master node, and it is responsible for coordinating certain VxVM activities.

You must run commands that configure or reconfigure VxVM objects on the master node. Tasks that must be initiated from the master node include setting up shared disk groups, creating and reconfiguring volumes, and performing snapshot operations.

VxVM determines that the first node to join a cluster performs the function of master node. If the master node leaves a cluster, one of the slave nodes is chosen to be the new master.

Private and shared disk groups

The following types of disk groups are defined:

Private disk group	Belongs to only one node. A private disk group is only imported by one system. Disks in a private disk group may be physically accessible from one or more systems, but access is restricted to one system only. The boot disk group (usually aliased by the reserved disk group name `bootdg`) is always a private disk group.
Shared disk group	Can be shared by all nodes. A shared (or cluster-shareable) disk group is imported by all cluster nodes. Disks in a shared disk group must be physically accessible from all systems that may join the cluster.

In a cluster, most disk groups are shared. Disks in a shared disk group are accessible from all nodes in a cluster, allowing applications on multiple cluster nodes to simultaneously access the same disk. A volume in a shared disk group can be simultaneously accessed by more than one node in the cluster, subject to licensing and disk group activation mode restrictions.

You can use the `vxdg` command to designate a disk group as cluster-shareable.

See "Importing disk groups as shared" on page 504.

When a disk group is imported as cluster-shareable for one node, each disk header is marked with the cluster ID. As each node subsequently joins the cluster, it recognizes the disk group as being cluster-shareable and imports it. As system administrator, you can also import or deport a shared disk group at any time; the operation takes place in a distributed fashion on all nodes.

Each physical disk is marked with a unique disk ID. When cluster functionality for VxVM starts on the master, it imports all shared disk groups (except for any that have the `noautoimport` attribute set). When a slave tries to join a cluster, the master sends it a list of the disk IDs that it has imported, and the slave checks to see if it can access them all. If the slave cannot access one of the listed disks, it abandons its attempt to join the cluster. If it can access all of the listed disks, it imports the same shared disk groups as the master and joins the cluster. When a

node leaves the cluster, it deports all its imported shared disk groups, but they remain imported on the surviving nodes.

Reconfiguring a shared disk group is performed with the cooperation of all nodes. Configuration changes to the disk group happen simultaneously on all nodes and the changes are identical. Such changes are atomic in nature, which means that they either occur simultaneously on all nodes or not at all.

Whether all members of the cluster have simultaneous read and write access to a cluster-shareable disk group depends on its activation mode setting.

See "Activation modes of shared disk groups" on page 480.

The data contained in a cluster-shareable disk group is available as long as at least one node is active in the cluster. The failure of a cluster node does not affect access by the remaining active nodes. Regardless of which node accesses a cluster-shareable disk group, the configuration of the disk group looks the same.

Warning: Applications running on each node can access the data on the VM disks simultaneously. VxVM does not protect against simultaneous writes to shared volumes by more than one node. It is assumed that applications control consistency (by using Veritas Cluster File System or a distributed lock manager, for example).

Activation modes of shared disk groups

A shared disk group must be activated on a node in order for the volumes in the disk group to become accessible for application I/O from that node. The ability of applications to read from or to write to volumes is dictated by the activation mode of a shared disk group. Valid activation modes for a shared disk group are `exclusivewrite`, `readonly`, `sharedread`, `sharedwrite`, and `off` (inactive).

Disk group activation was a new feature in Veritas Volume Manager 3.0. To maintain compatibility with previous releases, the default activation mode for shared disk groups is `shared-write`.

Special uses of clusters, such as high availability (HA) applications and off-host backup, can use disk group activation to explicitly control volume access from different nodes in the cluster

Table 13-1 describes the activation modes.

Table 13-1 Activation modes for shared disk groups

Activation mode	Description
`exclusivewrite (ew)`	The node has exclusive write access to the disk group. No other node can activate the disk group for write access.

Table 13-1 Activation modes for shared disk groups *(continued)*

Activation mode	Description
`readonly (ro)`	The node has read access to the disk group and denies write access for all other nodes in the cluster. The node has no write access to the disk group. Attempts to activate a disk group for either of the write modes on other nodes fail.
`sharedread (sr)`	The node has read access to the disk group. The node has no write access to the disk group, however other nodes can obtain write access.
`sharedwrite (sw)`	The node has write access to the disk group.
`off`	The node has neither read nor write access to the disk group. Query operations on the disk group are permitted.

Table 13-2 summarizes the allowed and conflicting activation modes for shared disk groups.

Table 13-2 Allowed and conflicting activation modes

Disk group activated in cluster as...	Attempt to activate disk group on another node as...			
	exclusive-write	readonly	sharedread	sharedwrite
exclusivewrite	Fails	Fails	Succeeds	Fails
readonly	Fails	Succeeds	Succeeds	Fails
sharedread	Succeeds	Succeeds	Succeeds	Succeeds
sharedwrite	Fails	Fails	Succeeds	Succeeds

To place activation modes under user control, create a defaults file `/etc/default/vxdg` containing the following lines:

```
enable_activation=true
default_activation_mode=activation-mode
```

The *activation-mode* is one of `exclusivewrite`, `readonly`, `sharedread`, `sharedwrite`, or `off`.

When a shared disk group is created or imported, it is activated in the specified mode. When a node joins the cluster, all shared disk groups accessible from the node are activated in the specified mode.

The activation mode of a disk group controls volume I/O from different nodes in the cluster. It is not possible to activate a disk group on a given node if it is activated in a conflicting mode on another node in the cluster. When enabling activation using the defaults file, it is recommended that this file be made identical on all nodes in the cluster. Otherwise, the results of activation are unpredictable.

If the defaults file is edited while the `vxconfigd` daemon is already running, the `vxconfigd` process must be restarted for the changes in the defaults file to take effect.

If the default activation mode is anything other than `off`, an activation following a cluster join, or a disk group creation or import can fail if another node in the cluster has activated the disk group in a conflicting mode.

To display the activation mode for a shared disk group, use the `vxdg list` *`diskgroup`* command.

See "Listing shared disk groups" on page 502.

You can also use the `vxdg` command to change the activation mode on a shared disk group.

See "Changing the activation mode on a shared disk group" on page 506.

It is also possible to configure a volume so that it can only be opened by a single node in a cluster.

See "Creating volumes with exclusive open access by a node" on page 507.

See "Setting exclusive open access to a volume by a node" on page 508.

Connectivity policy of shared disk groups

A shared disk group provides concurrent read and write access to the volumes that it contains for all nodes in a cluster. A shared disk group can only be created on the master node. This has the following advantages and implications:

- All nodes in the cluster see exactly the same configuration.
- Only the master node can change the configuration.
- Any changes on the master node are automatically coordinated and propagated to the slave nodes in the cluster.
- Any failures that require a configuration change must be sent to the master node so that they can be resolved correctly.

- As the master node resolves failures, all the slave nodes are correctly updated. This ensures that all nodes have the same view of the configuration.

The practical implication of this design is that I/O failure on any node results in the configuration of all nodes being changed. This is known as the global detach policy. However, in some cases, it is not desirable to have all nodes react in this way to I/O failure. To address this, an alternate way of responding to I/O failures, known as the local detach policy, was introduced.

The local detach policy is intended for use with shared mirrored volumes in a cluster. This policy prevents I/O failure on a single slave node from causing a plex to be detached. This would require the plex to be resynchronized when it is subsequently reattached. The local detach policy is available for disk groups that have a version number of 70 or greater.

For small mirrored volumes, non-mirrored volumes, volumes that use hardware mirrors, and volumes in private disk groups, there is no benefit in configuring the local detach policy. In most cases, it is recommended that you use the default global detach policy.

The detach policies have no effect if the master node loses access to all copies of the configuration database and logs in a disk group. If this happened in releases prior to 4.1, the master node always disabled the disk group. Release 4.1 introduces the disk group failure policy, which allows you to change this behavior for critical disk groups. This policy is only available for disk groups that have a version number of 120 or greater.

See "Global detach policy" on page 483.

See "Local detach policy" on page 484.

See "Disk group failure policy" on page 485.

See "Guidelines for choosing detach and failure policies" on page 486.

Global detach policy

Warning: The global detach policy must be selected when Dynamic MultiPathing (DMP) is used to manage multipathing on Active/Passive arrays, This ensures that all nodes correctly coordinate their use of the active path.

The global detach policy is the traditional and default policy for all nodes on the configuration. If there is a read or write I/O failure on a slave node, the master node performs the usual I/O recovery operations to repair the failure, and the plex is detached cluster-wide. All nodes remain in the cluster and continue to perform I/O, but the redundancy of the mirrors is reduced. When the problem

that caused the I/O failure has been corrected, the mirrors that were detached must be recovered before the redundancy of the data can be restored.

Local detach policy

Warning: Do not use the local detach policy if you use the VCS agents that monitor the cluster functionality of Veritas Volume Manager, and which are provided with Veritas Storage Foundation™ for Cluster File System HA and Veritas Storage Foundation for databases HA. These agents do not notify VCS about local failures.

The local detach policy is designed to support failover applications in large clusters where the redundancy of the volume is more important than the number of nodes that can access the volume. If there is a write failure on a slave node, the master node performs the usual I/O recovery operations to repair the failure, and additionally contacts all the nodes to see if the disk is still acceptable to them. If the write failure is not seen by all the nodes, I/O is stopped for the node that first saw the failure, and the application using the volume is also notified about the failure.

If required, configure the cluster management software to move the application to a different node, and/or remove the node that saw the failure from the cluster. The volume continues to return write errors, as long as one mirror of the volume has an error. The volume continues to satisfy read requests as long as one good plex is available.

If the reason for the I/O error is corrected and the node is still a member of the cluster, it can resume performing I/O from/to the volume without affecting the redundancy of the data.

The `vxdg` command can be used to set the disk detach policy on a shared disk group.

See "Setting the disk detach policy on a shared disk group" on page 507.

Table 13-3 summarizes the effect on a cluster of I/O failure to the disks in a mirrored volume.

Table 13-3 Cluster behavior under I/O failure to a mirrored volume for different disk detach policies

Type of I/O failure	Local (diskdetpolicy=local)	Global (diskdetpolicy=global)
Failure of path to one disk in a volume for a single node	Reads fail only if no plexes remain available to the affected node. Writes to the volume fail.	The plex is detached, and I/O from/to the volume continues. An I/O error is generated if no plexes remain.
Failure of paths to all disks in a volume for a single node	I/O fails for the affected node.	The plex is detached, and I/O from/to the volume continues. An I/O error is generated if no plexes remain.
Failure of one or more disks in a volume for all nodes.	The plex is detached, and I/O from/to the volume continues. An I/O error is generated if no plexes remain.	The plex is detached, and I/O from/to the volume continues. An I/O error is generated if no plexes remain.

Disk group failure policy

The local detach policy by itself is insufficient to determine the desired behavior if the master node loses access to all disks that contain copies of the configuration database and logs. In this case, the disk group is disabled. As a result, the other nodes in the cluster also lose access to the volume. In release 4.1, the disk group failure policy is introduced to determine the behavior of the master node in such cases.

Table 13-4 shows how the behavior of the master node changes according to the setting of the failure policy.

Table 13-4 Behavior of master node for different failure policies

Type of I/O failure	Leave (dgfailpolicy=leave)	Disable (dgfailpolicy=dgdisable)
Master node loses access to all copies of the logs.	The master node panics with the message “klog update failed” for a failed kernel-initiated transaction, or “cvm config update failed” for a failed user-initiated transaction.	The master node disables the disk group.

The behavior of the master node under the disk group failure policy is independent of the setting of the disk detach policy. If the disk group failure policy is set to

leave, all nodes panic in the unlikely case that none of them can access the log copies.

The vxdg command can be used to set the failure policy on a shared disk group.

See "Setting the disk group failure policy on a shared disk group" on page 507.

Guidelines for choosing detach and failure policies

In most cases it is recommended that you use the global detach policy, and particularly if any of the following conditions apply:

- If you are using the VCS agents that monitor the cluster functionality of Veritas Volume Manager, and which are provided with Veritas Storage Foundation™ for Cluster File System HA and Veritas Storage Foundation for databases HA. These agents do not notify VCS about local failures.
- When an array is seen by DMP as Active/Passive. The local detach policy causes unpredictable behavior for Active/Passive arrays.
- For clusters with four or fewer nodes. With a small number of nodes in a cluster, it is preferable to keep all nodes actively using the volumes, and to keep the applications running on all the nodes.
- If only non-mirrored, small mirrored, or hardware mirrored volumes are configured. This avoids the system overhead of the extra messaging that is required by the local detach policy.

The local detach policy may be suitable in the following cases:

- When large mirrored volumes are configured. Resynchronizing a reattached plex can degrade system performance. The local detach policy can avoid the need to detach the plex at all. (Alternatively, the dirty region logging (DRL) feature can be used to reduce the amount of resynchronization that is required.)
- For clusters with more than four nodes. Keeping an application running on a particular node is less critical when there are many nodes in a cluster. It may be possible to configure the cluster management software to move an application to a node that has access to the volumes. In addition, load balancing may be able to move applications to a different volume from the one that experienced the I/O problem. This preserves data redundancy, and other nodes may still be able to perform I/O from/to the volumes on the disk.

If you have a critical disk group that you do not want to become disabled in the case that the master node loses access to the copies of the logs, set the disk group failure policy to leave. This prevents I/O failure on the master node disabling the disk group. However, critical applications running on the master node fail if they lose access to the other shared disk groups. In such a case, it may be preferable to set the policy to dgdisable, and to allow the disk group to be disabled.

The default settings for the detach and failure policies are `global` and `dgdisable` respectively. You can use the `vxdg` command to change both the detach and failure policies on a shared disk group, as shown in this example:

```
# vxdg -g diskgroup set diskdetpolicy=local dgfailpolicy=leave
```

Effect of disk connectivity on cluster reconfiguration

The detach policy, previous I/O errors, or access to disks are not considered when a new master node is chosen. When the master node leaves a cluster, the node that takes over as master of the cluster may already have seen I/O failures for one or more disks. Under the local detach policy, if a node was affected before reconfiguration, and this node then becomes the master, the failure is treated differently from the global detach policy case.

See "Connectivity policy of shared disk groups" on page 482.

Some failure scenarios do not result in a disk group failure policy being invoked, but can potentially impact the cluster. For example, if the local disk detach policy is in effect, and the new master node has a failed plex, this results in all nodes detaching the plex because the new master is unaffected by the policy.

The detach policy does not change the requirement that a node joining a cluster must have access to all the disks in all shared disk groups. Similarly, a node that is removed from the cluster because of an I/O failure cannot rejoin the cluster until this requirement is met.

Limitations of shared disk groups

Only raw device access may be performed via the cluster functionality of VxVM. It does not support shared access to file systems in shared volumes unless the appropriate software, such as Veritas Cluster File System, is installed and configured.

Note: The boot disk group (usually aliased as `bootdg`) cannot be made cluster-shareable. It must be private.

The cluster functionality of VxVM does not support RAID-5 volumes, or task monitoring for cluster-shareable disk groups. These features can, however, be used in private disk groups that are attached to specific nodes of a cluster.

If you have RAID-5 volumes in a private disk group that you wish to make shareable, you must first relayout the volumes as a supported volume type such as `stripe-mirror` or `mirror-stripe`. Online relayout of shared volumes is supported provided that it does not involve RAID-5 volumes.

If a shared disk group contains RAID-5 volumes, deport it and then reimport the disk group as private on one of the cluster nodes. Reorganize the volumes into layouts that are supported for shared disk groups, and then deport and reimport the disk group as shared.

Cluster initialization and configuration

Before any nodes can join a new cluster for the first time, you must supply certain configuration information during cluster monitor setup. This information is normally stored in some form of cluster monitor configuration database. The precise content and format of this information depends on the characteristics of the cluster monitor. The information required by VxVM is as follows:

- cluster ID
- node IDs
- network addresses of nodes
- port addresses

When a node joins the cluster, this information is automatically loaded into VxVM on that node at node startup time.

Note: To make effective use of the cluster functionality of VxVM requires that you configure a cluster monitor (such as provided by Sun Java™ System Cluster software, or by GAB (Group Membership and Atomic Broadcast) in VCS). For a VCS environment, use the `vxcvmconfig` command on any node to configure the cluster to use the cluster functionality of VxVM. The `vxcvmconfig` command is not included with Veritas Volume Manager.

The cluster monitor startup procedure effects node initialization, and brings up the various cluster components (such as VxVM with cluster support, the cluster monitor, and a distributed lock manager) on the node. Once this is complete, applications may be started. The cluster monitor startup procedure must be invoked on each node to be joined to the cluster.

For VxVM in a cluster environment, initialization consists of loading the cluster configuration information and joining the nodes in the cluster. The first node to join becomes the master node, and later nodes (slaves) join to the master. If two nodes join simultaneously, VxVM chooses the master. Once the join for a given node is complete, that node has access to the shared disk groups and volumes.

Cluster reconfiguration

Cluster reconfiguration occurs if a node leaves or joins a cluster. Each node's cluster monitor continuously watches the other cluster nodes. When the membership of the cluster changes, the cluster monitor informs VxVM for it to take appropriate action.

During cluster reconfiguration, VxVM suspends I/O to shared disks. I/O resumes when the reconfiguration completes. Applications may appear to freeze for a short time during reconfiguration.

If other operations, such as VxVM operations or recoveries, are in progress, cluster reconfiguration can be delayed until those operations have completed. Volume reconfigurations do not take place at the same time as cluster reconfigurations. Depending on the circumstances, an operation may be held up and restarted later. In most cases, cluster reconfiguration takes precedence. However, if the volume reconfiguration is in the commit stage, it completes first.

See "Volume reconfiguration" on page 492.

See "vxclust utility" on page 489.

See "vxclustadm utility" on page 490.

vxclust utility

`vxclust` is used when Sun Java System Cluster software acts as the cluster monitor.

Every time there is a cluster reconfiguration, every node currently in the cluster runs the `vxclust` utility at each of several well-orchestrated steps. The cluster monitor facilities ensure that the same step is executed on all nodes at the same time. A given step only starts when the previous one has completed on all nodes. At each step in the reconfiguration, the `vxclust` utility determines what the cluster functionality of VxVM should do next. After informing VxVM of its next action, the `vxclust` utility waits for the outcome (success, failure, or retry) and communicates that to the cluster monitor.

If a node does not respond to a the `vxclust` utility request within a specific timeout period, that node aborts. The `vxclust` utility then decides whether to restart the reconfiguration or give up, depending on the circumstances. If the cause of the reconfiguration is a local, uncorrectable error, `vxclust` gives up. If a node cannot complete an operation because another node has left, the surviving node times out. In this case, the `vxclust` utility requests a reconfiguration with the expectation that another node will leave. If no other node leaves, the `vxclust` utility causes the local node to leave.

If a reconfiguration step fails, the `vxclust` utility returns an error to the cluster monitor. The cluster monitor may decide to abort the node, causing its immediate

departure from the cluster. Any I/O in progress to the shared disk fails and access to the shared disks is stopped.

`vxclust` decides what actions to take when it is informed of changes in the cluster. If a new master node is required (due to failure of the previous master), `vxclust` determines which node becomes the new master.

vxclustadm utility

The `vxclustadm` command provides an interface to the cluster functionality of VxVM when VCS is used as the cluster monitor. It is also called during cluster startup and shutdown. In the absence of a cluster monitor, `vxclustadm` can also be used to activate or deactivate the cluster functionality of VxVM on any node in a cluster.

The `startnode` keyword to `vxclustadm` starts cluster functionality on a cluster node by passing cluster configuration information to the VxVM kernel. In response to this command, the kernel and the VxVM configuration daemon, `vxconfigd`, perform initialization.

The `stopnode` keyword stops cluster functionality on a node. It waits for all outstanding I/O to complete and for all applications to close shared volumes.

The `abortnode` keyword terminates cluster activity on a node. It does not wait for outstanding I/O to complete nor for applications to close shared volumes.

The `reinit` keyword allows nodes to be added to or removed from a cluster without stopping the cluster. Before running this command, the cluster configuration file must have been updated with information about the supported nodes in the cluster.

The `nidmap` keyword prints a table showing the mapping between node IDs in VxVM's cluster-support subsystem and node IDs in the cluster monitor. It also prints the state of the node in the cluster.

The `nodestate` keyword reports the state of a cluster node and also the reason for the last abort of the node as shown in this example:

```
# /etc/vx/bin/vxclustadm nodestate

state: out of cluster
reason: user initiated stop
```

Table 13-5 lists the various reasons that may be given for a node abort.

Table 13-5 Node abort messages

Reason	Description
`cannot find disk on slave node`	Missing disk or bad disk on the slave node.
`cannot obtain configuration data`	The node cannot read the configuration data due to an error such as disk failure.
`cluster device open failed`	Open of a cluster device failed.
`clustering license mismatch with master node`	Clustering license does not match that on the master node.
`clustering license not available`	Clustering license cannot be found.
`connection refused by master`	Join of a node refused by the master node.
`disk in use by another cluster`	A disk belongs to a cluster other than the one that a node is joining.
`join timed out during reconfiguration`	Join of a node has timed out due to reconfiguration taking place in the cluster.
`klog update failed`	Cannot update kernel log copies during the join of a node.
`master aborted during join`	Master node aborted while another node was joining the cluster.
`minor number conflict`	Minor number conflicts exist between private disk groups and shared disk groups that are being imported.
`protocol version out of range`	Cluster protocol version mismatch or unsupported version.
`recovery in progress`	Volumes that were opened by the node are still recovering.
`transition to role failed`	Changing the role of a node to be the master failed.
`user initiated abort`	Node is out of cluster due to an abort initiated by the cluster monitor.
`user initiated stop`	Node is out of cluster due to a stop initiated by the user or by the cluster monitor.
`vxconfigd is not enabled`	The VxVM configuration daemon is not enabled.

See the vxclustadm(1M) manual page.

Volume reconfiguration

Volume reconfiguration is the process of creating, changing, and removing VxVM objects such as disk groups, volumes and plexes. In a cluster, all nodes co-operate to perform such operations. The vxconfigd daemons play an active role in volume reconfiguration. For reconfiguration to succeed, a vxconfigd daemon must be running on each of the nodes.

See "vxconfigd daemon" on page 493.

A volume reconfiguration transaction is initiated by running a VxVM utility on the master node. The utility contacts the local vxconfigd daemon on the master node, which validates the requested change. For example, vxconfigd rejects an attempt to create a new disk group with the same name as an existing disk group. The vxconfigd daemon on the master node then sends details of the changes to the vxconfigd daemons on the slave nodes. The vxconfigd daemons on the slave nodes then perform their own checking. For example, each slave node checks that it does not have a private disk group with the same name as the one being created; if the operation involves a new disk, each node checks that it can access that disk. When the vxconfigd daemons on all the nodes agree that the proposed change is reasonable, each notifies its kernel. The kernels then co-operate to either commit or to abandon the transaction. Before the transaction can be committed, all of the kernels ensure that no I/O is underway. The master node is responsible both for initiating the reconfiguration, and for coordinating the commitment of the transaction. The resulting configuration changes appear to occur simultaneously on all nodes.

If a vxconfigd daemon on any node goes away during reconfiguration, all nodes are notified and the operation fails. If any node leaves the cluster, the operation fails unless the master has already committed it. If the master node leaves the cluster, the new master node, which was previously a slave node, completes or fails the operation depending on whether or not it received notification of successful completion from the previous master node. This notification is performed in such a way that if the new master does not receive it, neither does any other slave.

If a node attempts to join a cluster while a volume reconfiguration is being performed, the result of the reconfiguration depends on how far it has progressed. If the kernel has not yet been invoked, the volume reconfiguration is suspended until the node has joined the cluster. If the kernel has been invoked, the node waits until the reconfiguration is complete before joining the cluster.

When an error occurs, such as when a check on a slave fails or a node leaves the cluster, the error is returned to the utility and a message is sent to the console on the master node to identify on which node the error occurred.

vxconfigd daemon

The VxVM configuration daemon, `vxconfigd`, maintains the configuration of VxVM objects. It receives cluster-related instructions from the `vxclust` utility under Sun Java System Cluster software, or from the kernel when running VCS. A separate copy of `vxconfigd` runs on each node, and these copies communicate with each other over a network. When invoked, a VxVM utility communicates with the `vxconfigd` daemon running on the same node; it does not attempt to connect with `vxconfigd` daemons on other nodes. During cluster startup, the `vxclust` utility(SunCluster) or the kernel (for VCS) prompts `vxconfigd` to begin cluster operation and indicates whether it is a master node or a slave node.

When a node is initialized for cluster operation, the `vxconfigd` daemon is notified that the node is about to join the cluster and is provided with the following information from the cluster monitor configuration database:

- cluster ID
- node IDs
- master node ID
- role of the node
- network address of the `vxconfigd` daemon on each node (if applicable)

On the master node, the `vxconfigd` daemon sets up the shared configuration by importing shared disk groups, and informs the `vxclust` utility (for SunCluster) or the kernel (for VCS) when it is ready for the slave nodes to join the cluster.

On slave nodes, the `vxconfigd` daemon is notified when the slave node can join the cluster. When the slave node joins the cluster, the `vxconfigd` daemon and the VxVM kernel communicate with their counterparts on the master node to set up the shared configuration.

When a node leaves the cluster, the kernel notifies the `vxconfigd` daemon on all the other nodes. The master node then performs any necessary cleanup. If the master node leaves the cluster, the kernels select a new master node and the `vxconfigd` daemons on all nodes are notified of the choice.

The `vxconfigd` daemon also participates in volume reconfiguration.

See "Volume reconfiguration" on page 492.

vxconfigd daemon recovery

In a cluster, the `vxconfigd` daemons on the slave nodes are always connected to the `vxconfigd` daemon on the master node. If the `vxconfigd` daemon is stopped, volume reconfiguration cannot take place. Other nodes can join the cluster if the `vxconfigd` daemon is not running on the slave nodes.

If the `vxconfigd` daemon stops, different actions are taken depending on which node this occurred:

- If the `vxconfigd` daemon is stopped on the master node, the `vxconfigd` daemons on the slave nodes periodically attempt to rejoin to the master node. Such attempts do not succeed until the `vxconfigd` daemon is restarted on the master. In this case, the `vxconfigd` daemons on the slave nodes have not lost information about the shared configuration, so that any displayed configuration information is correct.
- If the `vxconfigd` daemon is stopped on a slave node, the master node takes no action. When the `vxconfigd` daemon is restarted on the slave, the slave `vxconfigd` daemon attempts to reconnect to the master daemon and to re-acquire the information about the shared configuration. (Neither the kernel view of the shared configuration nor access to shared disks is affected.) Until the `vxconfigd` daemon on the slave node has successfully reconnected to the `vxconfigd` daemon on the master node, it has very little information about the shared configuration and any attempts to display or modify the shared configuration can fail. For example, shared disk groups listed using the `vxdg list` command are marked as `disabled`; when the rejoin completes successfully, they are marked as `enabled`.
- If the `vxconfigd` daemon is stopped on both the master and slave nodes, the slave nodes do not display accurate configuration information until `vxconfigd` is restarted on the master and slave nodes, and the daemons have reconnected.

If the `vxclust` utility (for SunCluster) or the CVM agent (for VCS) determines that the `vxconfigd` daemon has stopped on a node, `vxconfigd` is restarted automatically.

Warning: The `-r` reset option to `vxconfigd` restarts the `vxconfigd` daemon and recreates all states from scratch. This option cannot be used to restart `vxconfigd` while a node is joined to a cluster because it causes cluster information to be discarded.

It may sometimes be necessary to restart `vxconfigd` manually in a VCS controlled cluster to resolve a VxVM issue.

To restart vxconfigd manually

1 Use the following command to disable failover on any service groups that contain VxVM objects:

```
# hagrp -freeze group
```

2 Enter the following command to stop and restart the VxVM configuration daemon on the affected node:

```
# vxconfigd -k
```

3 Use the following command to re-enable failover for the service groups that you froze in step 1:

```
# hagrp -unfreeze group
```

Node shutdown

Although it is possible to shut down the cluster on a node by invoking the shutdown procedure of the node's cluster monitor, this procedure is intended for terminating cluster components after stopping any applications on the node that have access to shared storage. VxVM supports clean node shutdown, which allows a node to leave the cluster gracefully when all access to shared volumes has ceased. The host is still operational, but cluster applications cannot be run on it.

The cluster functionality of VxVM maintains global state information for each volume. This enables VxVM to determine which volumes need to be recovered when a node crashes. When a node leaves the cluster due to a crash or by some other means that is not clean, VxVM determines which volumes may have writes that have not completed and the master node resynchronizes these volumes. It can use dirty region logging (DRL) or FastResync if these are active for any of the volumes.

Clean node shutdown must be used after, or in conjunction with, a procedure to halt all cluster applications. Depending on the characteristics of the clustered application and its shutdown procedure, a successful shutdown can require a lot of time (minutes to hours). For instance, many applications have the concept of draining, where they accept no new work, but complete any work in progress before exiting. This process can take a long time if, for example, a long-running transaction is active.

When the VxVM shutdown procedure is invoked, it checks all volumes in all shared disk groups on the node that is being shut down. The procedure then either continues with the shutdown, or fails for one of the following reasons:

- If all volumes in shared disk groups are closed, VxVM makes them unavailable to applications. Because all nodes are informed that these volumes are closed on the leaving node, no resynchronization is performed.
- If any volume in a shared disk group is open, the shutdown procedure fails. The shutdown procedure can be repeatedly retried until it succeeds. There is no timeout checking in this operation—it is intended as a service that verifies that the clustered applications are no longer active.

Once shutdown succeeds, the node has left the cluster. It is not possible to access the shared volumes until the node joins the cluster again.

Since shutdown can be a lengthy process, other reconfiguration can take place while shutdown is in progress. Normally, the shutdown attempt is suspended until the other reconfiguration completes. However, if it is already too far advanced, the shutdown may complete first.

Node abort

If a node does not leave a cluster cleanly, this is because it crashed or because some cluster component made the node leave on an emergency basis. The ensuing cluster reconfiguration calls the VxVM abort function. This procedure immediately attempts to halt all access to shared volumes, although it does wait until pending I/O from or to the disk completes.

I/O operations that have not yet been started are failed, and the shared volumes are removed. Applications that were accessing the shared volumes therefore fail with errors.

After a node abort or crash, shared volumes must be recovered, either by a surviving node or by a subsequent cluster restart, because it is very likely that there are unsynchronized mirrors.

Cluster shutdown

If all nodes leave a cluster, shared volumes must be recovered when the cluster is next started if the last node did not leave cleanly, or if resynchronization from previous nodes leaving uncleanly is incomplete.

Dirty region logging in cluster environments

Dirty region logging (DRL) is an optional property of a volume that provides speedy recovery of mirrored volumes after a system failure. DRL is supported in cluster-shareable disk groups. This section provides a brief overview of how DRL behaves in a cluster environment.

In a cluster environment, the VxVM implementation of DRL differs slightly from the normal implementation.

A dirty region log on a system without cluster support has a recovery map and a single active map. A dirty region log in a cluster, however, has one recovery map and one active map for each node in the cluster.

The dirty region log size in clusters is typically larger than in non-clustered systems, as it must accommodate a recovery map plus active maps for each node in the cluster. The size of each map within the dirty region log is one or more whole blocks. The `vxassist` command automatically allocates a sufficiently large dirty region log for the size of the volume and the number of nodes.

It is possible to reimport a non-shared disk group (and its volumes) as a shared disk group in a cluster environment. However, the dirty region logs of the imported disk group may be considered invalid and a full recovery may result.

If a shared disk group is imported as a private disk group on a system without cluster support, VxVM considers the logs of the shared volumes to be invalid and conducts a full volume recovery. After the recovery completes, VxVM uses DRL.

The cluster functionality of VxVM can perform a DRL recovery on a non-shared volume. However, if such a volume is moved to a VxVM system with cluster support and imported as shared, the dirty region log is probably too small to accommodate maps for all the cluster nodes. VxVM then marks the log invalid and performs a full recovery anyway. Similarly, moving a DRL volume from a two-node cluster to a four-node cluster can result in too small a log size, which the cluster functionality of VxVM handles with a full volume recovery. In both cases, you must allocate a new log of sufficient size.

See "Dirty region logging" on page 62.

How DRL works in a cluster environment

When one or more nodes in a cluster crash, DRL must handle the recovery of all volumes that were in use by those nodes when the crashes occurred. On initial cluster startup, all active maps are incorporated into the recovery map during the volume start operation.

Nodes that crash (that is, leave the cluster as dirty) are not allowed to rejoin the cluster until their DRL active maps have been incorporated into the recovery maps on all affected volumes. The recovery utilities compare a crashed nod'es active maps with the recovery map and make any necessary updates before the node can rejoin the cluster and resume I/O to the volume (which overwrites the active map). During this time, other nodes can continue to perform I/O.

VxVM tracks which nodes have crashed. If multiple node recoveries are underway in a cluster at a given time, their respective recoveries and recovery map updates

can compete with each other. VxVM tracks changes in the state of DRL recovery and prevents I/O collisions.

The master node performs volatile tracking of DRL recovery map updates for each volume, and prevents multiple utilities from changing the recovery map simultaneously.

Multiple host failover configurations

Outside the context of clustering functionality, VxVM disk groups can be imported (made available) from only one host at any given time. When a host imports a disk group as private, the volumes and configuration of that disk group become accessible to the host. If the administrator or system software wants to privately use the same disk group from another host, the host that already has the disk group imported (importing host) must deport (give up access to) the disk group. Once deported, the disk group can be imported by another host.

If two hosts are allowed to access a disk group concurrently without proper synchronization, such as that provided by the Oracle Parallel Server, the configuration of the disk group, and possibly the contents of volumes, can be corrupted. Similar corruption can also occur if a file system or database on a raw disk partition is accessed concurrently by two hosts, so this problem in not limited to Veritas Volume Manager.

Import lock

When a host in a non-clustered environment imports a disk group, an import lock is written on all disks in that disk group. The import lock is cleared when the host deports the disk group. The presence of the import lock prevents other hosts from importing the disk group until the importing host has deported the disk group.

Specifically, when a host imports a disk group, the import normally fails if any disks within the disk group appear to be locked by another host. This allows automatic re-importing of disk groups after a reboot (autoimporting) and prevents imports by another host, even while the first host is shut down. If the importing host is shut down without deporting the disk group, the disk group can only be imported by another host by clearing the host ID lock first (discussed later).

The import lock contains a host ID (in Veritas Volume Manager, this is the host name) reference to identify the importing host and enforce the lock. Problems can therefore arise if two hosts have the same host ID.

Since Veritas Volume Manager uses the host name as the host ID (by default), it is advisable to change the host name of one machine if another machine shares

its host name. To change the host name, use the `vxdctl hostid new_hostname` command.

Failover

The import locking scheme works well in an environment where disk groups are not normally shifted from one system to another. However, consider a setup where two hosts, Node A and Node B, can access the drives of a disk group. The disk group is first imported by Node A, but the administrator wants to access the disk group from Node B if Node A crashes. Such a failover scenario can be used to provide manual high availability to data, where the failure of one node does not prevent access to data. Failover can be combined with a "high availability" monitor to provide automatic high availability to data: when Node B detects that Node A has crashed or shut down, Node B imports (fails over) the disk group to provide access to the volumes.

Veritas Volume Manager can support failover, but it relies on the administrator or on an external high-availability monitor to ensure that the first system is shut down or unavailable before the disk group is imported to another system.

See "Moving disk groups between systems" on page 238.

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

Corruption of disk group configuration

If `vxdg import` is used with `-C` (clears locks) and/or `-f` (forces import) to import a disk group that is still in use from another host, disk group configuration corruption is likely to occur. Volume content corruption is also likely if a file system or database is started on the imported volumes before the other host crashes or shuts down.

If this kind of corruption occurs, you must probably rebuild your configuration from scratch and reload all volumes in the disk group from a backup. To backup and rebuild the configuration, if nothing has changed, use `vxprint -mspvd` and store the output which can be fed to `vxmake` to restore the layouts. There are typically numerous configuration copies for each disk group, but corruption nearly always affects all configuration copies, so redundancy does not help in this case.

Disk group configuration corruption usually shows up as missing or duplicate records in the configuration databases. This can result in a variety of `vxconfigd` error messages

```
VxVM vxconfigd ERROR
V-5-1-569 Disk group group,Disk disk:
Cannot auto-import group: reason
```

where the *reason* can describe errors such as:

```
Association not resolved
Association count is incorrect
Duplicate record in configuration
Configuration records are inconsistent
```

These errors are typically reported in association with specific disk group configuration copies, but usually apply to all copies. The following is usually displayed along with the error:

```
Disk group has no valid configuration copies
```

If you use the Veritas Cluster Server product, all disk group failover issues can be managed correctly. VCS includes a high availability monitor and includes failover scripts for VxVM, VxFS, and for several popular databases.

The `-t` option to `vxdg` prevents automatic re-imports on reboot and is necessary when used with a host monitor (such as VCS) that controls imports itself, rather than relying on automatic imports by Veritas Volume Manager.

See the *Veritas Volume Manager Troubleshooting Guide*.

Administering VxVM in cluster environments

The following sections describe the administration of VxVM's cluster functionality.

Note: Most VxVM commands require superuser or equivalent privileges.

Requesting node status and discovering the master node

The `vxdctl` utility controls the operation of the `vxconfigd` volume configuration daemon. The `-c` option can be used to request cluster information and to find out which node is the master. To determine whether the `vxconfigd` daemon is enabled and/or running, use the following command:

```
vxdctl -c mode
```

Table 13-6 shows the various messages that may be output according to the current status of the cluster node.

Table 13-6 Cluster status messages

Status message	Description

`mode: enabled:`
`cluster active - MASTER`
`master: mozart`
	The node is the master.

`mode: enabled:`
`cluster active - SLAVE`
`master: mozart`
	The node is a slave.

`mode: enabled:`
`cluster active - role not set`
`master: mozart`
`state: joining`
`reconfig: master update`
	The node has not yet been assigned a role, and is in the process of joining the cluster.

`mode: enabled:`
`cluster active - SLAVE`
`master: mozart`
`state: joining`
	The node is configured as a slave, and is in the process of joining the cluster.

`mode: enabled:` `cluster inactive`	The cluster is not active.

If the `vxconfigd` daemon is disabled, no cluster information is displayed.

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

Determining if a disk is shareable

The `vxdisk` utility manages VxVM disks. To use the `vxdisk` utility to determine whether a disk is part of a cluster-shareable disk group, use the following command:

```
# vxdisk list accessname
```

where *accessname* is the disk access name (or device name).

For example, a portion of the output from this command (for the device `c4t1d0`) is shown here:

```
Device:    c4t1d0
devicetag: c4t1d0
```

```
type:      auto
clusterid: cvm2
disk:      name=shdg01 id=963616090.1034.cvm2
timeout:   30
group:     name=shdg id=963616065.1032.cvm2
flags:     online ready autoconfig shared imported
...
```

Note that the `clusterid` field is set to `cvm2` (the name of the cluster), and the `flags` field includes an entry for `shared`. When a node is not joined to the cluster, the `flags` field contains the `autoimport` flag instead of `imported`.

Listing shared disk groups

`vxdg` can be used to list information about shared disk groups. To display information for all disk groups, use the following command:

```
# vxdg list
```

Example output from this command is displayed here:

```
NAME        STATE              ID
rootdg      enabled            774215886.1025.teal
group2      enabled,shared     774575420.1170.teal
group1      enabled,shared     774222028.1090.teal
```

Shared disk groups are designated with the flag `shared`.

To display information for shared disk groups only, use the following command:

```
# vxdg -s list
```

Example output from this command is as follows:

```
NAME     STATE            ID
group2   enabled,shared   774575420.1170.teal
group1   enabled,shared   774222028.1090.teal
```

To display information about one specific disk group, use the following command:

```
# vxdg list diskgroup
```

The following is example output for the command `vxdg list group1` on the master:

```
Group:     group1
dgid:      774222028.1090.teal
```

```
import-id: 32768.1749
flags:     shared
version:   140
alignment: 8192 (bytes)
ssb:       on
local-activation: exclusive-write
cluster-actv-modes: node0=ew node1=off
detach-policy: local
private_region_failure: leave
copies:    nconfig=2 nlog=2
config:    seqno=0.1976 permlen=1456 free=1448 templen=6 loglen=220
config disk c1t0d0s2 copy 1 len=1456 state=clean online
config disk c1t0d0s2 copy 1 len=1456 state=clean online
log disk c1t0d0s2 copy 1 len=220
log disk c1t0d0s2 copy 1 len=220
```

Note that the `flags` field is set to `shared`. The output for the same command when run on a slave is slightly different. The `local-activation` and `cluster-actv-modes` fields display the activation mode for this node and for each node in the cluster respectively. The `detach-policy` and `private_region_failure` fields indicate how the cluster behaves in the event of loss of connectivity to the disks, and to the configuration and log copies on the disks.

Creating a shared disk group

Shared disk groups can only be created on the master node.

If the cluster software has been run to set up the cluster, a shared disk group can be created using the following command:

```
# vxdg -s init diskgroup [diskname=]devicename
```

where *diskgroup* is the disk group name, *diskname* is the administrative name chosen for a VM disk, and *devicename* is the device name (or disk access name).

Warning: The operating system cannot tell if a disk is shared. To protect data integrity when dealing with disks that can be accessed by multiple systems, use the correct designation when adding a disk to a disk group. VxVM allows you to add a disk that is not physically shared to a shared disk group if the node where the disk is accessible is the only node in the cluster. However, this means that other nodes cannot join the cluster. Furthermore, if you attempt to add the same disk to different disk groups (private or shared) on two nodes at the same time, the results are undefined. Perform all configuration on one node only, and preferably on the master node.

Forcibly adding a disk to a disk group

Disks can only be forcibly added to a shared disk group on the master node.

If VxVM does not add a disk to an existing disk group because that disk is not attached to the same nodes as the other disks in the disk group, you can forcibly add the disk using the following command:

```
# vxdg -f adddisk -g diskgroup [diskname=]devicename
```

Warning: Only use the force option(`-f`) if you are fully aware of the consequences such as possible data corruption.

Importing disk groups as shared

Shared disk groups can only be imported on the master node.

Disk groups can be imported as shared using the `vxdg -s import` command. If the disk groups are set up before the cluster software is run, the disk groups can be imported into the cluster arrangement using the following command:

```
# vxdg -s import diskgroup
```

where *diskgroup* is the disk group name or ID. On subsequent cluster restarts, the disk group is automatically imported as shared. Note that it can be necessary to deport the disk group (using the `vxdg deport diskgroup` command) before invoking the `vxdg` utility.

Forcibly importing a disk group

You can use the `-f` option to the `vxdg` command to import a disk group forcibly.

Warning: The force option(`-f`) must be used with caution and only if you are fully aware of the consequences such as possible data corruption.

When a cluster is restarted, VxVM can refuse to auto-import a disk group for one of the following reasons:

- A disk in the disk group is no longer accessible because of hardware errors on the disk. In this case, use the following command to forcibly reimport the disk group:

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

- Some of the nodes to which disks in the disk group are attached are not currently in the cluster, so the disk group cannot access all of its disks. In this case, a forced import is unsafe and must not be attempted because it can result in inconsistent mirrors.

Converting a disk group from shared to private

Shared disk groups can only be deported on the master node.

To convert a shared disk group to a private disk group, first deport it on the master node using this command:

```
# vxdg deport diskgroup
```

Then reimport the disk group on any cluster node using this command:

```
# vxdg import diskgroup
```

Moving objects between shared disk groups

You can only move objects between shared disk groups on the master node. You cannot move objects between private and shared disk groups.

You can use the `vxdg move` command to move a self-contained set of VxVM objects such as disks and top-level volumes between disk groups. In a cluster, you can move such objects between private disk groups on any cluster node where those disk groups are imported.

See "Moving objects between disk groups" on page 257.

Splitting shared disk groups

You can use the `vxdg split` command to remove a self-contained set of VxVM objects from an imported disk group, and move them to a newly created disk group.

See "Splitting disk groups" on page 260.

Splitting a private disk group creates a private disk group, and splitting a shared disk group creates a shared disk group. You can split a private disk group on any cluster node where that disk group is imported. You can only split a shared disk group or create a shared target disk group on the master node.

See "Moving objects between disk groups" on page 257.

Joining shared disk groups

You cannot join a private disk group and a shared disk group.

You can use the `vxdg join` command to merge the contents of two imported disk groups. In a cluster, you can join two private disk groups on any cluster node where those disk groups are imported.

If the source disk group and the target disk group are both shared, you must perform the join on the master node.

See "Joining disk groups" on page 261.

Changing the activation mode on a shared disk group

The activation mode for access by a cluster node to a shared disk group is set directly on that node.

The activation mode of a shared disk group can be changed using the following command:

```
# vxdg -g diskgroup set activation=mode
```

The activation mode is one of `exclusivewrite` or `ew`, `readonly` or `ro`, `sharedread` or `sr`, `sharedwrite` or `sw`, or `off`.

If you use this command to change the activation mode of a shared disk group, you must first change the activation mode to `off` before setting it to any other value, as shown here:

```
# vxdg -g myshdg set activation=off
# vxdg -g myshdg set activation=readonly
```

See "Activation modes of shared disk groups" on page 480.

Setting the disk detach policy on a shared disk group

The disk detach policy for a shared disk group can only be set on the master node.

The `vxdg` command may be used to set either the `global` or `local` disk detach policy for a shared disk group:

```
# vxdg -g diskgroup set diskdetpolicy=global|local
```

The default disk detach policy is `global`.

See "Connectivity policy of shared disk groups" on page 482.

Setting the disk group failure policy on a shared disk group

The disk group failure policy for a shared disk group can only be set on the master node.

The `vxdg` command may be used to set either the `dgdisable` or `leave` failure policy for a shared disk group:

```
# vxdg -g diskgroup set dgfailpolicy=dgdisable|leave
```

The default failure policy is `dgdisable`.

See "Disk group failure policy" on page 485.

Creating volumes with exclusive open access by a node

All shared volumes, including those with exclusive open access, can only be created on the master node.

When using the `vxassist` command to create a volume, you can use the `exclusive=on` attribute to specify that the volume may only be opened by one node in the cluster at a time. For example, to create the mirrored volume `volmir` in the disk group `dskgrp`, and configure it for exclusive open, use the following command:

```
# vxassist -g dskgrp make volmir 5g layout=mirror exclusive=on
```

Multiple opens by the same node are also supported. Any attempts by other nodes to open the volume fail until the final close of the volume by the node that opened it.

Specifying `exclusive=off` instead means that more than one node in a cluster can open a volume simultaneously.

Setting exclusive open access to a volume by a node

Exclusive open access on a volume can only be set on the master node. Ensure that none of the nodes in the cluster have the volume open when setting this attribute.

You can set the `exclusive=on` attribute with the vxvol command to specify that an existing volume may only be opened by one node in the cluster at a time.

For example, to set exclusive open on the volume `volmir` in the disk group `dskgrp`, use the following command:

```
# vxvol -g dskgrp set exclusive=on volmir
```

Multiple opens by the same node are also supported. Any attempts by other nodes to open the volume fail until the final close of the volume by the node that opened it.

Specifying `exclusive=off` instead means that more than one node in a cluster can open a volume simultaneously.

Displaying the cluster protocol version

The following command displays the cluster protocol version running on a node:

```
# vxdctl list
```

This command produces output similar to the following:

```
Volboot file
version: 3/1
seqno: 0.19
cluster protocol version: 80
hostid: giga
entries:
```

You can also check the existing cluster protocol version using the following command:

```
# vxdctl protocolversion
```

This produces output similar to the following:

```
Cluster running at protocol 80
```

Displaying the supported cluster protocol version range

The following command displays the maximum and minimum protocol version supported by the node and the current protocol version:

```
# vxdctl support
```

This command produces out put similar to the following:

```
Support information:
vxconfigd_vrsn:     21
dg_minimum:         20
dg_maximum:         140
kernel:             15
protocol_minimum:   80
protocol_maximum:   80
protocol_current:   80
```

You can also use the following command to display the maximum and minimum cluster protocol version supported by the current Veritas Volume Manager release:

```
# vxdctl protocolrange
```

This produces output similar to the following:

```
minprotoversion: 80, maxprotoversion: 80
```

Upgrading the cluster protocol version

The cluster protocol version can only be updated on the master node.

After all the nodes in the cluster have been updated with a new cluster protocol, you can upgrade the entire cluster using the following command on the master node:

```
# vxdctl upgrade
```

Recovering volumes in shared disk groups

Volumes can only be recovered on the master node.

The `vxrecover` utility is used to recover plexes and volumes after disk replacement. When a node leaves a cluster, it can leave some mirrors in an inconsistent state. The `vxrecover` utility can be used to recover such volumes. The `-c` option to `vxrecover` causes it to recover all volumes in shared disk groups. The `vxconfigd` daemon automatically calls the `vxrecover` utility with the `-c` option when necessary.

Warning: While the `vxrecover` utility is active, there can be some degradation in system performance.

Obtaining cluster performance statistics

The `vxstat` utility returns statistics for specified objects. In a cluster environment, `vxstat` gathers statistics from all of the nodes in the cluster. The statistics give the total usage, by all nodes, for the requested objects. If a local object is specified, its local usage is returned.

You can optionally specify a subset of nodes using the following form of the command:

```
# vxstat -g diskgroup -n node[,node...]
```

where *node* is an integer. If a comma-separated list of nodes is supplied, the `vxstat` utility displays the sum of the statistics for the nodes in the list.

For example, to obtain statistics for node 2, volume `vol1`,use the following command:

```
# vxstat -g group1 -n 2 vol1
```

This command produces output similar to the following:

```
           OPERATIONS    BLOCKS           AVG TIME(ms)
TYP  NAME  READ   WRITE  READ     WRITE  READ   WRITE
vol  vol1  2421   0      600000   0      99.0   0.0
```

To obtain and display statistics for the entire cluster, use the following command:

```
# vxstat -b
```

The statistics for all nodes are summed. For example, if node 1 performed 100 I/O operations and node 2 performed 200 I/O operations, `vxstat -b` displays a total of 300 I/O operations.

Chapter 14

Administering sites and remote mirrors

This chapter includes the following topics:

- About sites and remote mirrors
- Configuring Remote Mirror sites
- Configuring sites for hosts
- Configuring sites for storage
- Changing the site name
- Configuring site-based allocation on a disk group
- Configuring site consistency on a disk group
- Configuring site consistency on a volume
- Setting the siteread policy on a volume
- Site-based allocation of storage to volumes
- Making an existing disk group site consistent
- Fire drill — testing the configuration
- Failure scenarios and recovery procedures

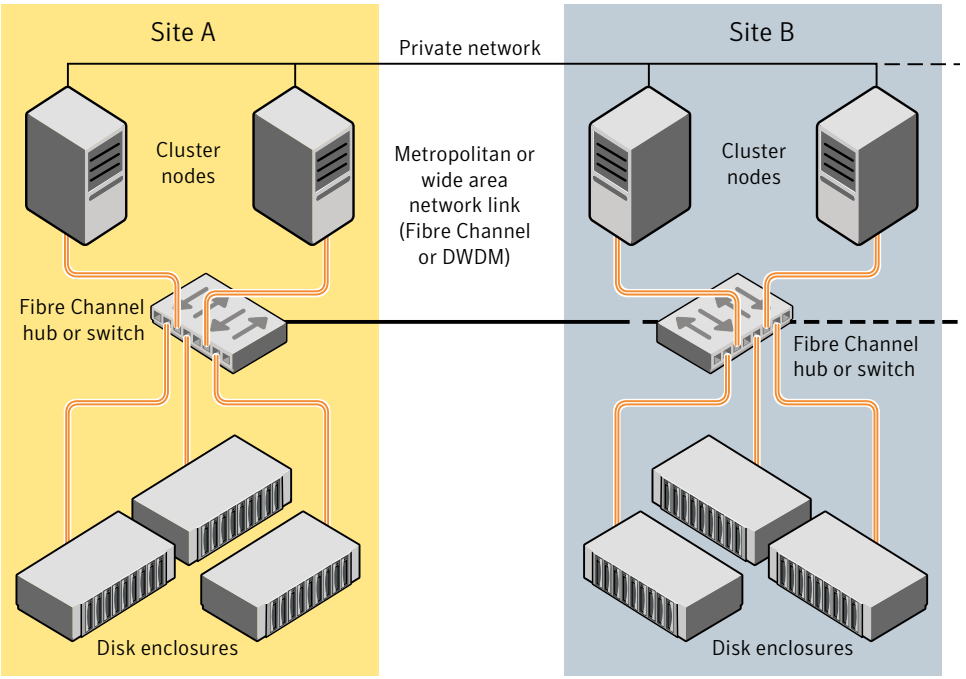
About sites and remote mirrors

In a Remote Mirror configuration (also known as a campus cluster or stretch cluster) the hosts and storage of a cluster that would usually be located in one

place, are instead divided between two or more sites. These sites are typically connected via a redundant high-capacity network that provides access to storage and private link communication between the cluster nodes.

Figure 14-1 shows a typical two-site remote mirror configuration.

Figure 14-1 Example of a two-site remote mirror configuration



If a disk group is configured across the storage at the sites, and inter-site communication is disrupted, there is a possibility of a serial split brain condition arising if each site continues to update the local disk group configuration copies.

See "Handling conflicting configuration copies" on page 244.

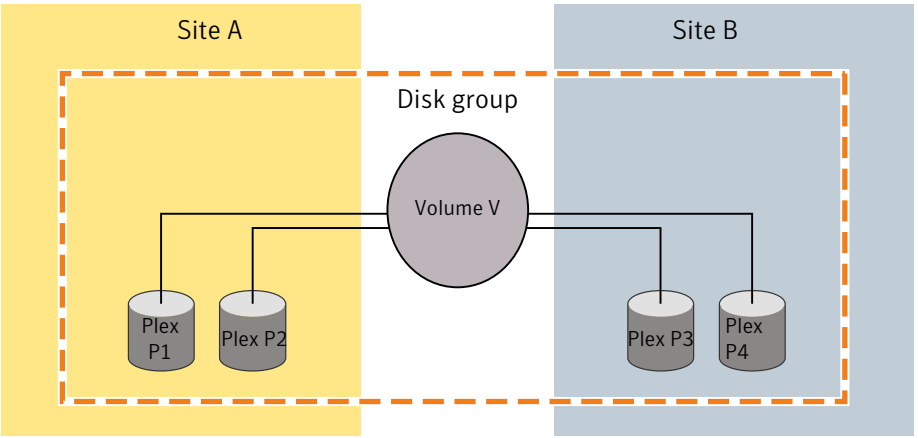
VxVM provides mechanisms for dealing with the serial split brain condition, monitoring the health of a remote mirror, and testing the robustness of the cluster against various types of failure (also known as fire drill).

For applications and services to function correctly at a site when other sites have become inaccessible, at least one complete plex of each volume must be configured at each site (site-based allocation), and the consistency of the data in the plexes at each site must be ensured (site consistency).

By tagging disks with site names, storage can be allocated from the correct location when creating, resizing or relocating a volume, and when changing a volume's layout.

Figure 14-2 shows an example of a site-consistent volume with two plexes configured at each of two sites.

Figure 14-2 Site-consistent volume with two plexes at each of two sites



The storage for plexes P1 and P2 is allocated storage that is tagged as belonging to site A, and the storage for plexes P3 and P4 is allocated storage that is tagged as belonging to site B.

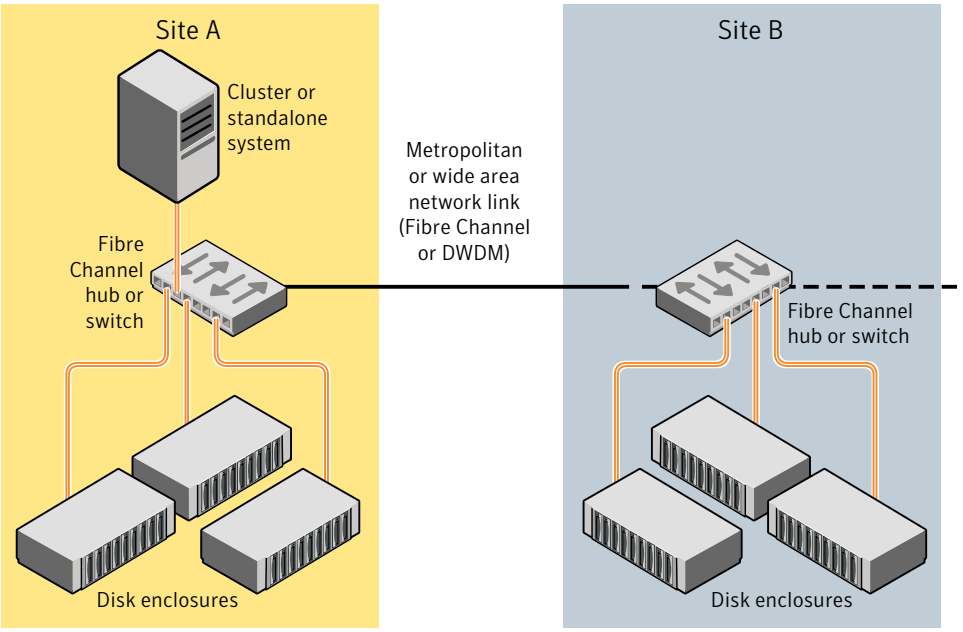
Although not shown in this figure, DCO log volumes are also mirrored across the sites, and disk group configuration copies are distributed across the sites.

Site consistency means that the data in the plexes for a volume must be consistent at each site. The site consistency of a volume is ensured by detaching a site when its last complete plex fails at that site. If a site fails, all its plexes are detached and the site is said to be detached. If site consistency is not on, only the plex that fails is detached. The remaining volumes and their plexes on that site are not detached.

To enhance read performance, VxVM will service reads from the plexes at the local site where an application is running if the `siteread` read policy is set on a volume. Writes are written to plexes at all sites.

Figure 14-3 shows a configuration with remote storage only that is also supported.

Figure 14-3 Example of a two-site configuration with remote storage only



Configuring Remote Mirror sites

Note: The Remote Mirror feature requires that the Site Awareness license has been installed on all hosts at all sites that participate in the configuration.

This section describes setting up a new disk group. To configure an existing disk group as a Remote Mirror configuration, additional steps may be required.

See "Making an existing disk group site consistent" on page 523.

Setting up a new disk group for a Remote Mirror configuration requires the following steps:

- Define the site name for each host that can access the disk group.
  See "Configuring sites for hosts" on page 515.
- Create the disk group with storage from each site.
- Register a site record to the disk group, for each site.
  See "Configuring site-based allocation on a disk group" on page 517.
- Set up automatic site tagging for the disk group, if required.

See “Configuring automatic site tagging for a disk group” on page 516.

- Assign a site name to the disks or enclosures. You can set site tags at the disk level or at the enclosure level. If you specify one or more enclosures, the site tag applies to the disks in that enclosure that are within the disk group.
  See “Configuring site tagging for disks or enclosures” on page 516.
- Turn on site consistency for the disk group.
  See “Configuring site consistency on a disk group” on page 518.

Configuring sites for hosts

To set up a Remote Mirror configuration, specify to which site each host belongs.

Use the following command to set the site name for each host:

```
# vxdctl set site=sitename
```

The name that has been assigned to a site is stored in the `/etc/vx/volboot` file.

To display the site name for a host, use the following command:

```
# vxdctl list | grep siteid
siteid: building1
```

To remove the site name from a host, use the following command:

```
# vxdctl [-F] unset site
```

The `-F` option is required if any imported disk groups are registered to the site.

Configuring sites for storage

For a Remote Mirror configuration, each disk or enclosure in the disk group is assigned to one of the sites in the configuration. Use site tags to specify to which site each storage device in the disk group belongs.

Assign a site tag in one of the following ways:

- Using automatic site tagging on a disk group.
  See “Configuring automatic site tagging for a disk group” on page 516.
- Manually tagging one or more disks or enclosures.
  The disks or enclosures need not be included in a disk group.
  See “Configuring site tagging for disks or enclosures” on page 516.

Configuring automatic site tagging for a disk group

Configure automatic site tagging if you want disks or LUNs to inherit the tag from the enclosure. After you turn on automatic site tagging for a disk group, assign the site names to the enclosures in the disk group. Any disks or LUNs added to that disk group inherit the tag from the enclosure to which they belong.

To configure automatic site tagging for a disk group

1. Set the autotagging policy to **on** for the disk group. Automatic tagging is the default setting, so this step is only required if the autotagging policy was previously disabled.

   To turn on autotagging, use the following command:

   ```
   # vxdg [-g diskgroup] set autotagging=on
   ```

2. Assign the site name to an enclosure within the disk group, using the following command:

   ```
   # vxdg [-g diskgroup] settag encl:encl_name site=sitename
   ```

To display the setting for automatic site tagging for a disk group

- To determine whether automatic site tagging is on for a disk group, use the following command:

  ```
  # vxprint -g diskgroup -F"%autotagging" diskgroup
  ```

To list the site tags for a disk group

- To list the site tags for a disk group, use the following command:

  ```
  # vxdg [-g diskgroup] listtag site=sitename
  ```

To remove a site tag from an enclosure or a disk group

- To remove a site tag from a disk group, use the following command:

  ```
  # vxdg [-g diskgroup] rmtag [encl:encl_name] site=sitename
  ```

Configuring site tagging for disks or enclosures

To set up a Remote Mirror configuration, specify to which site each storage device in the disk group belongs. Assign a site tag to one or more disks or enclosures. If the disk or enclosure does not belong to a disk group, you must use this method to assign a site tag.

To tag disks or enclosures with a site name

- Assign a site name to one or more disks or enclosures, using the following command:

```
# vxdisk [-g diskgroup] settag site=sitename \
  disk disk1...|encl:encl_name encl:encl_name1...
```

  where the disks can be specified either by the disk access name or the disk media name.

To display the disks or enclosures registered to a site

- To check which disks or enclosures are registered to a site, use the following command:

```
# vxdisk [-g diskgroup] listtag
```

To remove the site tag from a disk or enclosure

- To remove the site tag from a disk or enclosure, use the following command:

```
# vxdisk rmtag site=sitename \
disk disk1...|encl:encl_name encl:encl_name1...
```

Changing the site name

You can change the site name, or tag, that is used to identify each site in a Remote Mirror configuration. Renaming the site changes the site record in the disk group. The site name is also changed for all of the disks and enclosures that are tagged with the existing site name.

After you rename a site, you may need to explicitly change the site name for each host that belongs to that site.

See "Configuring sites for hosts" on page 515.

To rename the site

- Specify the new site name as follows:

```
# vxdg [-g diskgroup] renamesite old_sitename new_sitename
```

Configuring site-based allocation on a disk group

Site-based allocation requires that each volume has at least one plex at each site that is registered in the disk group. When you set the site-based allocation

requirement for a disk group, each volume created has the `allsites` attribute set to `on`, by default. The `allsites` attribute indicates that the volume must have at least one plex on each site that is registered to the disk group. For new volumes, the read policy is set to `siteread`.

If site-based allocation is not required, or is not possible (as is the case for RAID-5 volumes), specify the `allsites=off` attribute to the `vxassist` command.

Before setting site-based allocation on a disk group, be sure to meet the following requirements:

- The Site Awareness license must be installed on all the hosts in the Remote Mirror configuration
- Each existing volume in the disk group must have at least one plex at each site. If this condition is not met, the command to turn on the site-based allocation fails. If the `-f` option is specified, the command does not fail, but instead it sets the `allsites` attribute for the volume to `off`.

To turn on the site-based allocation requirement on a disk group

- Use the `vxdg addsite` command for each site at which site-based allocation is required:

  ```
  # vxdg -g diskgroup [-f] addsite sitename
  ```

To remove the site-based allocation requirement on a disk group

- To remove the site-based allocation requirement from a site, use this command:

  ```
  # vxdg -g diskgroup [-f] rmsite sitename
  ```

  The `-f` option allows the requirement to be removed if the site is detached or offline.

  The site name is not removed from the disks. If required, use the `vxdisk rmtag` command to remove the site tag.

  See "Configuring Remote Mirror sites" on page 514.

Configuring site consistency on a disk group

Site consistency means that the data in the plexes for a volume must be consistent at each site. A site-consistent volume has the same number of plexes, or mirrors, on each site in the disk group. A site-consistent volume must have at least one plex, or mirror, on each site in the disk group. The site consistency of a volume is ensured by detaching a site when its last complete plex fails at that site. If a

site fails, all its plexes are detached and the site is said to be detached. Turn on this behavior by setting the `siteconsistent` attribute to `on`.

If the `siteconsistent` attribute is set to `off`, only the plex that fails is detached. The remaining volumes and their plexes on that site are not detached.

If you turn on the site consistency requirement for a disk group, each new volume created in the disk group inherits the site consistency of the disk group, by default. Setting the `siteconsistent` attribute on a disk group does not affect existing volumes in the disk group. You can also control the site consistency on individual volumes.

See "Configuring site consistency on a volume" on page 520.

Before setting site consistency on a disk group, be sure to meet the following requirements:

- Site-based allocation must be configured for a disk group before site consistency is turned on.
  See "Configuring site-based allocation on a disk group" on page 517.
- All the disks in a disk group must be registered to one of the sites before you can set the `siteconsistent` attribute on the disk group.

To turn on the site consistency requirement for a disk group

- Turn on the site consistency requirement for a disk group by using the `vxdg` command:

```
# vxdg -g diskgroup set siteconsistent=on
```

To view the setting for the site consistency requirement

- To verify whether site consistency has been enabled for a disk group, use the following command:

```
# vxdg list diskgroup | grep siteconsistent

flags: siteconsistent
```

To turn off the site consistency requirement

- To turn off the site consistency requirement for a disk group, use the following command:

```
# vxdg -g diskgroup set siteconsistent=off
```

Configuring site consistency on a volume

To set the site consistency requirement when creating a volume, specify the `siteconsistent` attribute to the `vxassist make` command, for example:

```
# vxassist [-g diskgroup] make volume size \
  nmirror=4 siteconsistent={on|off}
```

By default, a volume inherits the value that is set on its disk group.

By default, creating a site-consistent volume also creates an associated version 20 DCO volume, and enables Persistent FastResync on the volume. This allows faster recovery of the volume during the reattachment of a site.

To turn on the site consistency requirement for an existing volume, use the following form of the `vxvol` command:

```
# vxvol [-g diskgroup] set siteconsistent=on volume
```

To turn off the site consistency requirement for a volume, use the following command:

```
# vxvol [-g diskgroup] set siteconsistent=off volume
```

The `siteconsistent` attribute and the `allsites` attribute must be set to `off` for RAID-5 volumes in a site-consistent disk group.

Setting the siteread policy on a volume

If the Site Awareness license is installed on all the hosts in the Remote Mirror configuration, the disk group is configured for site consistency with several sites enabled, and the `allsites=on` attribute is specified for a volume, the default read policy is `siteread`.

If required, you can use the following command to set the `siteread` policy on a volume:

```
# vxvol [-g diskgroup] rdpol siteread volume
```

This command has no effect if a site name has not been set for the host.

See “Changing the read policy for mirrored volumes” on page 358.

Site-based allocation of storage to volumes

The `vxassist` command can be used to create volumes only from storage that exists at a specified site, as shown in this example:

```
# vxassist -g diskgroup make volume size site:site1 \
  [allsites={on|off}] [siteconsistent={on|off}]
```

The storage class `site` is used in similar way to other storage classes with the `vxassist` command, such as `enclr`, `ctlr` and `disk`.

See "Mirroring across targets, controllers or enclosures" on page 318.

If the Site Awareness license is installed on all the hosts in the Remote Mirror configuration, and site consistency is enabled on a volume, the `vxassist` command attempts to allocate storage across the sites that are registered to a disk group. If not enough storage is available at all sites, the command fails unless you also specify the `allsites=off` attribute.

By default, the `allsites` attribute is set to `on` for volume in a site-consistent disk group. The `allsites` and `siteconsistent` attributes must be set to `off` for RAID-5 volumes in a site-consistent disk group.

In a similar way to mirroring across controllers, you can also ensure that plexes are created at all sites that are registered for a disk group:

```
# vxassist -g diskgroup make volume size mirror=site
```

The `allsites` and `siteconsistent` attributes can be combined to create a non-site-consistent mirrored volume with plexes only at some of the sites:

```
# vxassist -g diskgroup make volume size mirror=site \
  site:site1 site:site2 ... allsites=off siteconsistent=off
```

a non-site-consistent mirrored volume with plexes at all of the sites:

```
# vxassist -g diskgroup make volume size mirror=site \
  allsites=on siteconsistent=off
```

a site-consistent mirrored volume with plexes only at some of the sites:

```
# vxassist -g diskgroup make volume size mirror=site \
  site:site1 site:site2 ... allsites=off siteconsistent=on
```

or a site-consistent mirrored volume with plexes at all of the sites:

```
# vxassist -g diskgroup make volume size mirror=site \
  allsites=on siteconsistent=on
```

Specifying the number of mirrors ensures that each mirror is created on a different site:

```
# vxassist -g diskgroup make volume size mirror=site \
  nmirror=2 site:site1 site:site2 [allsites={on|off}] \
  [siteconsistent={on|off}]
```

If a volume is intended to be site consistent, the number of mirrors that are specified must be equal to the number of sites.

Examples of storage allocation by specifying sites

Table 14-1 shows examples of how to use sites with the `vxassist` command to allocate storage. These examples assume that the disk group, `ccdg`, has been enabled for site consistency with disks configured at two sites, `site1` and `site2`.

Table 14-1 Examples of storage allocation by specifying sites

Command	Description
`# vxassist -g ccdg make vol 2g \` `  nmirror=2`	Create a volume with one mirror at each site. The `nmirror` keyword is optional. If the `nmirror` keyword is specified, it must equal the number of sites.

`# vxassist -g ccdg -o ordered \`
`  make vol 2g \`
`  layout=mirror-stripe ncol=3 \`
`  ccdg01 ccdg02 ccdg03 ccdg09 \`
`  ccdg10 ccdg11`
	Create a mirrored-stripe volume specifying allocation order to validate redundancy across the sites. The named disks must be tagged with the appropriate site name, and there must be sufficient disks at each site to create the volume.

`# vxassist -g ccdg make vol 2g \` `  nmirror=2 ccdg01 ccdg09`	Create a volume with one mirror on each of the named disks. The named disks must be tagged with the appropriate site name, and there must be sufficient disks at each site to create the volume.

`# vxassist -g ccdg make vol 2g \`
`  nmirror=2 siteconsistent=off \`
`  allsites=off`
	Create a mirrored volume that is not site consistent. Both mirrors can be allocated from any available storage in the disk group.

Table 14-1 Examples of storage allocation by specifying sites *(continued)*

Command	Description
`# vxassist -g ccdg make vol 2g \ nmirror=2 site:site2 \ siteconsistent=off \ allsites=off`	Create a mirrored volume that is not site consistent. Both mirrors are allocated from any available storage in the disk group that is tagged as belonging to `site2`.
`# vxassist -g ccdg make vol 2g \ nmirror=2 !site:site1 \ siteconsistent=off \ allsites=off`	Create a mirrored volume that is not site consistent. Both mirrors are allocated from any available storage in the disk group that is tagged as not belonging to `site1`.
`# vxassist -g ccdg mirror vol \ site:site1`	Add a mirror at a specified site. The command fails if there is insufficient storage available at the site.
`# vxassist -g ccdg remove \ mirror vol site:site1`	Remove a mirror from a volume at a specified site. If the volume is site consistent, the command fails if this would remove the last remaining plex at a site.
`# vxassist -g ccdg growto vol 4g`	Grow a volume. If the volume is site consistent, the command fails if there is insufficient storage available at each site.

Making an existing disk group site consistent

To make an existing disk group site consistent

1 Ensure that the disk group is updated to at least version 140, by running the `vxdg upgrade` command on it:

```
# vxdg upgrade diskgroup
```

2 On each host that can access the disk group, define the site name:

```
# vxdctl set site=sitename
```

3 Tag all the disks in the disk group with the appropriate site name:

```
# vxdisk [-g diskgroup] settag site=sitename disk1 disk2
```

Or, to tag all the disks in a specified enclosure, use the following command:

```
# vxdisk [-g diskgroup] settag site=sitename
  encl:encl_name
```

4 Use the `vxdg move` command to move any unsupported RAID-5 volumes to another disk group. Alternatively, use the `vxassist convert` commands to convert the volumes to a supported layout such as `mirror` or `mirror-stripe`. You can use the `site` and `mirror=site` storage allocation attribute to ensure that the plexes are created on the correct storage.

5 Use the `vxevac` command to ensue that the volumes have equal number of plexes at each site. You can use the `site` and `mirror=site` storage allocation attribute to ensure that the plexes are created on the correct storage.

6 Register a site record for each site with the disk group:

```
# vxdg -g diskgroup addsite sitename
```

7 Turn on site consistency for the disk group:

```
# vxdg -g diskgroup set siteconsistent=on
```

8 Turn on site consistency for each existing volume in the disk group:

```
# vxvol [-g diskgroup] set siteconsistent=on volume ...
```

Fire drill — testing the configuration

Warning: To avoid potential loss of service or data, it is recommended that you do not use these procedures on a live system.

After validating the consistency of the volumes and disk groups at your sites, you should validate the procedures that you will use in the event of the various possible types of failure. A fire drill lets you test that a site can be brought up cleanly during recovery from a disaster scenario such as site failure.

Simulating site failure

To simulate the failure of a site, use the following command to detach all the devices at a specified site:

```
# vxdg -g diskgroup [-f] detachsite sitename
```

The `-f` option must be specified if any plexes configured on storage at the site are currently online.

After the site is detached, the application should run correctly on the available site. This step verifies that the primary site is fine. Continue the fire drill by verifying the secondary site.

Verifying the secondary site

After detaching the site from primary site, verify whether the application starts correctly on a secondary site. The fire drill ensures that the application can run on the secondary if disaster strikes the primary site. These procedures assume that the application is running correctly before the fire drill operation begins.

To verify the secondary site, import the detached site on a different host using the following command:

```
vxdg -o site=sitename import dgname
```

Then start the application. If the application runs correctly on the secondary site, this step verifies the integrity of the secondary site.

Recovery from simulated site failure

Use the following commands to reattach a site and recover the disk group:

```
# vxdg -g diskgroup [-o overridessb] reattachsite sitename
# vxrecover -g diskgroup
```

It may be necessary to specify the `-o overridessb` option if a serial split-brain condition is indicated.

Failure scenarios and recovery procedures

Table 14-2 lists the possible failure scenarios and recovery procedures for the Remote Mirror feature.

Table 14-2 Failure scenarios and recovery procedures

Failure scenario	Recovery procedure
Disruption of network link between sites.	See "Recovering from a loss of site connectivity" on page 526.
Failure of hosts at a site.	See "Recovering from host failure" on page 526.
Failure of storage at a site.	See "Recovering from storage failure" on page 526.
Failure of both hosts and storage at a site.	See "Recovering from site failure" on page 527.

Recovering from a loss of site connectivity

If the network links between the sites are disrupted, the application environments may continue to run in parallel, and this may lead to inconsistencies between the disk group configuration copies at the sites. When connectivity between the sites is restored, a serial split-brain condition may then exist between the sites. One site must be chosen as having the preferred version of the disk group configuration copies. The configuration copies at the other sites can then be updated from these copies.

Use the following commands to reattach a site and recover the disk group:

```
# vxdg -g diskgroup -o overridessb reattachsite sitename
# vxrecover -g diskgroup
```

In the case that the host systems are configured at a single site with only storage at the remote sites, the usual resynchronization mechanism of VxVM is used to recover the remote plexes when the storage comes back on line.

Recovering from host failure

If one or more cluster nodes fail at a site, but the storage remains online, this is handled either by VCS failover in the case of the Storage Foundation HA product, or by node takeover in the case that the node was the master for a shared disk group as supported by the Storage Foundation Cluster File System software.

Recovering from storage failure

If storage fails at a site, the plexes that are configured on that storage are detached locally if a site-consistent volume still has other mirrors available at the site. The hot-relocation feature of VxVM will attempt to recreate the failed plexes on other available storage in the disk group. If no plexes of a site-consistent volume remain

in operation at a site, and hot-relocation cannot recreate the plexes at that site, the site is detached. Because site connectivity has not been lost, applications running on hosts at the site can still access data at the other sites.

When the storage comes back online, the `vxsited` reattaches the site automatically.

See "Automatic site reattachment" on page 527.

If the `vxsited` is not running, use the following commands to reattach a site and recover the disk group:

```
# vxdg -g diskgroup reattachsite sitename
# vxrecover -g diskgroup
```

Recovering from site failure

If all the hosts and storage fail at a site, use the following commands to reattach the site after it comes back online, and to recover the disk group:

```
# vxdg -g diskgroup [-o overridessb] reattachsite sitename
# vxrecover -g diskgroup
```

The `-o overridessb` option is only required if a serial split-brain condition is indicated. A serial split-brain condition may happen if the site was brought back up while the private network link was inoperative. This option updates the configuration database on the reattached site with the consistent copies at the other sites.

Automatic site reattachment

The automatic site reattachment daemon, `vxsited`, provides automatic reattachment of sites. The `vxsited` daemon uses the `vxnotify` mechanism to monitor storage coming back online on a site after a previous failure, and to restore redundancy of mirrors across sites.

If the hot-relocation daemon, `vxrelocd`, is running, `vxsited` attempts to reattach the site, and allows `vxrelocd` to try to use the available disks in the disk group to relocate the failed subdisks. If `vxrelocd` succeeds in relocating the failed subdisks, it starts the recovery of the plexes at the site. When all the plexes have been recovered, the plexes are put into the ACTIVE state, and the state of the site is set to ACTIVE.

If `vxrelocd` is not running, `vxsited` reattaches a site only when all the disks at that site become accessible. After reattachment succeeds, `vxsited` sets the site state to ACTIVE, and initiates recovery of the plexes. When all the plexes have been recovered, the plexes are put into the ACTIVE state.

Note: `vxsited` does not try to reattach a site that you have explicitly detached by using the `vxdg detachsite` command.

The automatic site reattachment feature is enabled by default. The `vxsited` daemon uses email to notify `root` of any attempts to reattach sites and to initiate recovery of plexes at those sites.

To send mail to other users, add the user name to the line that starts `vxsited` in the `/lib/svc/method/vxvm-recover` startup script and run the `svcadm refresh vxvm/vxvm-recover` command (for Solaris 10 onward), or `/etc/init.d/vxvm-recover` and reboot the system (for OS releases before Solaris 10).

If you do not want a site to be recovered automatically, kill the `vxsited` daemon, and prevent it from restarting. To kill the daemon, run the following command from the command line:

```
# ps -afe
```

Locate the process table entry for `vxsited`, and kill it by specifying its process ID:

```
# kill -9 PID
```

If there is no entry in the process table for `vxsited`, the automatic site reattachment feature is disabled.

To prevent the automatic site reattachment feature from being restarted, comment out the line that starts `vxsited` in the `/lib/svc/method/vxvm-recover` startup script and run the `svcadm refresh vxvm/vxvm-recover` command (for Solaris 10 onward), or `/etc/init.d/vxvm-recover` (for OS releases before Solaris 10).

Chapter 15

Using Storage Expert

This chapter includes the following topics:

- About Storage Expert
- How Storage Expert works
- Before using Storage Expert
- Running Storage Expert
- Identifying configuration problems using Storage Expert

About Storage Expert

System administrators often find that gathering and interpreting data about large and complex configurations can be a difficult task. Veritas Storage Expert 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 “How Storage Expert works” on page 530.

See “Before using Storage Expert” on page 530.

See “Running Storage Expert” on page 530.

See “Identifying configuration problems using Storage Expert” on page 533.

See “Rule definitions and attributes” on page 539.

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.

Running Storage Expert

The executable rule files are located in the directory, /opt/VRTS/vxse/vxvm. It is recommended that this directory be added to the PATH variable.

The rules are invoked using the following command-line syntax:

```
# rulename [options] keyword [attribute=value ...]
```

Each of the rules performs a different function.

See "Rule definitions and attributes" on page 539.

The following options may be specified:

-d defaults_file	Specifies an alternate defaults file.
-g diskgroup	Specifies verbose output format.
-v	Specifies the disk group to be examined.

One of the following keywords must be specified:

`check`	Lists the default values used by the rule's attributes.
`info`	Describes what the rule does.
`list`	Lists the attributes of the rule that you can set.
`run`	Runs the rule.

See "Rule definitions and attributes" on page 539.

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 - TUNEABLES 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 - TUNEABLES
------------------------------------------------------------
VxVM vxse:vxse_stripes2 INFO V-5-1-5546
```

```
too_wide_stripe - (16) 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.

See "Rule definitions and attributes" on page 539.

To alter the behavior of rules, you can change the value of their attributes.

See "Setting rule attributes" on page 532.

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 532.

You can set Storage Expert to run as a `cron` job to notify administrators, and to archive reports automatically.

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:

- A value specified on the command line.
- A value specified in a user-defined defaults file.
- A value in the `/etc/default/vxse` file that has not been commented out.
- 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 name

See "Rule definitions and attributes" on page 539.

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 dirty region log (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.

See "Preparing a volume for DRL and instant snapshots" on page 341.

Checking for large mirrored volumes without a mirrored dirty region log (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.

See "Preparing a volume for DRL and instant snapshots" on page 341.

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.

See "Adding a RAID-5 log" on page 351.

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.

See "Adding a RAID-5 log" on page 351.

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.

See "Creating a disk group" on page 221.

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 the 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.

See "Upgrading a disk group" on page 264.

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 "Displaying disk group information" on page 220.

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`.

See "Importing a disk group" on page 226.

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.

See "Adding a disk to a disk group" on page 223.

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 336.

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

See "Reattaching plexes" on page 289.

See "Starting a volume" on page 335.

See 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.

See "Converting between layered and non-layered volumes" on page 370.

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 363.

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 363.

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 363.

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 “About hot-relocation” on page 455.

Hardware failures

VxVM maintains information about failed disks and disabled controllers.

Checking for disk failures and disabled controllers (vxse_dc_failures)

Rule `vxse_dc_failures` can be used to discover if the system has any failed disks or disabled controllers.

Rootability

The root disk can be put under VxVM control and mirrored.

Checking the validity of root mirrors (vxse_rootmir)

Rule `vxse_rootmir` can be used to confirm that the root mirrors are set up correctly.

See “Rootability” on page 128.

System name

The system name that is known to VxVM can be checked for consistency.

Checking the system name (vxse_host)

Rule `vxse_host` can be used to confirm that the system name 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

You can use the `info` keyword to show a description of a rule.

See “Discovering what a rule does” on page 531.

Table 15-1 lists the available rule definitions, and rule attributes and their default values.

Table 15-1 Rule definitions in Storage Expert

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.

Table 15-1 Rule definitions in Storage Expert *(continued)*

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

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 532.

Table 15-2 lists the available rule attributes and their default values.

Table 15-2 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.

Table 15-2 Rule attributes and default attribute values *(continued)*

Rule	Attribute	Default value	Description
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.
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.

Table 15-2 Rule attributes and default attribute values *(continued)*

Rule	Attribute	Default value	Description
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.
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.

Chapter 16

Performance monitoring and tuning

This chapter includes the following topics:

- Performance guidelines
- RAID-5
- Performance monitoring
- Tuning VxVM

Performance guidelines

Veritas Volume Manager (VxVM) can improve overall system performance by optimizing the layout of data storage on the available hardware. 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, which 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.

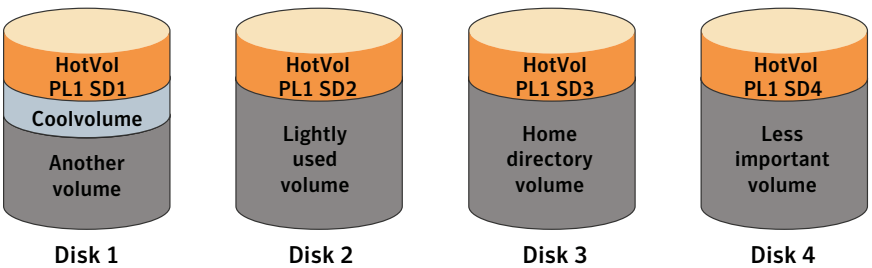
Striping

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.

Figure 16-1 shows an example of a single volume (`HotVol`) that has been identified as a data-access bottleneck.

Figure 16-1 Use of striping for optimal data access



This volume is striped across four disks, leaving the remaining space on these disks free for use by less-heavily used volumes.

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 48.

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.

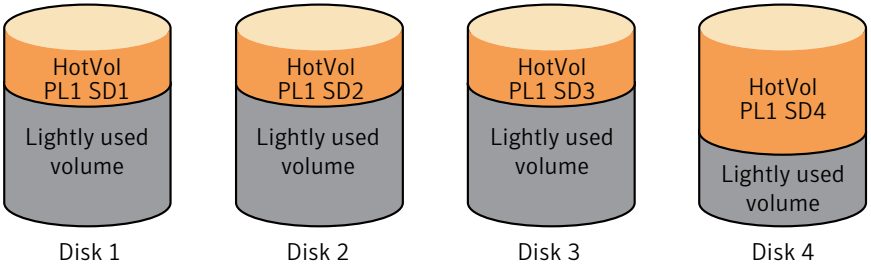
Volume read policies

To help optimize performance for different types of volumes, VxVM allows any one of several different types of read policy to be set on data plexes.

See "Changing the read policy for mirrored volumes" on page 358.

Figure 16-2 shows an example where the read policy of the mirrored-stripe volume labeled `Hot Vol` is set to `prefer` for the striped plex `PL1`.

Figure 16-2 Use of mirroring and striping for improved performance



The `prefer` policy distributes the load when reading across the otherwise lightly-used disks in `PL1`, as opposed to the single disk in plex `PL2`. (`HotVol` is an example of a mirrored-stripe volume in which one data plex is striped and the other data plex is concatenated.)

To improve performance for read-intensive workloads, you can attach up to 32 data plexes to the same volume. However, this 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.

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

Printing volume statistics

Use the `vxstat` command to access information about activity on volumes, plexes, subdisks, and disks under VxVM control, and to print summary statistics to the standard output. These statistics represent VxVM activity from the time the system initially booted or from the last time the counters were reset to zero. If no VxVM object name is specified, statistics from all volumes in the configuration database are reported.

VxVM records the following I/O statistics:

- count of operations
- number of blocks transferred (one operation can involve more than one block)

- average operation time (which reflects the total time through the VxVM interface and is not suitable for comparison against other statistics programs)

These statistics are recorded for logical I/O including reads, writes, atomic copies, verified reads, verified writes, plex reads, and plex writes for each volume. As a result, one write to a two-plex volume results in at least five operations: one for each plex, one for each subdisk, and one for the volume. Also, one read that spans two subdisks shows at least four reads—one read for each subdisk, one for the plex, and one for the volume.

VxVM also maintains other statistical data. For each plex, it records read and write failures. For volumes, it records corrected read and write failures in addition to read and write failures.

To reset the statistics information to zero, use the `-r` option. This can be done for all objects or for only those objects that are specified. Resetting just prior to an operation makes it possible to measure the impact of that particular operation.

The following is an example of output produced using the `vxstat` command:

```
                OPERATIONS          BLOCKS             AVG TIME(ms)
TYP  NAME         READ    WRITE      READ     WRITE     READ      WRITE
vol  blop            0        0         0         0      0.0        0.0
vol  foobarvol       0        0         0         0      0.0        0.0
vol  rootvol     73017   181735    718528   1114227     26.8       27.9
vol  swapvol     13197    20252    105569    162009     25.8      397.0
vol  testvol         0        0         0         0      0.0        0.0
```

Additional volume statistics are available for RAID-5 configurations.

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

Using performance data

When you have gathered performance data, you can use it to determine how to configure your system to use resources most effectively. The following sections provide an overview of how you can use this data.

Using I/O statistics

Examination of the I/O statistics can suggest how to reconfigure your system. You should examine two primary statistics: volume I/O activity and disk I/O activity.

Before obtaining statistics, reset the counters for all existing statistics using the `vxstat -r` command. This eliminates any differences between volumes or disks

due to volumes being created, and also removes statistics from boot time (which are not usually of interest).

After resetting the counters, allow the system to run during typical system activity. Run the application or workload of interest on the system to measure its effect. When monitoring a system that is used for multiple purposes, try not to exercise any one application more than usual. When monitoring a time-sharing system with many users, let statistics accumulate for several hours during the normal working day.

To display volume statistics, enter the `vxstat` command with no arguments. The following is a typical display of volume statistics:

```
                  OPERATIONS          BLOCKS         AVG TIME(ms)
TYP  NAME      READ    WRITE      READ     WRITE     READ    WRITE
vol  archive    865      807      5722      3809     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 typical output:

```
V   NAME        RVG/VSET/CO  KSTATE     STATE     LENGTH      READPOL      REFPLEX      UTYPE
PL  NAME        VOLUME       KSTATE     STATE     LENGTH      LAYOUT       NCOL/WDTH    MODE
SD  NAME        PLEX         DISK       DISKOFFS  LENGTH      [COL/]OFF    DEVICE       MODE

v   archive     -            ENABLED    ACTIVE    20480       SELECT       -            fsgen
```

```
pl  archive-01 archive      ENABLED    ACTIVE    20480       CONCAT       -            RW
sd  mydg03-03  archive-01   mydg03     0         40960       0            c1t2d0       ENA
```

The subdisks line (beginning `sd`) indicates that the volume `archive` is on disk `mydg03`. To move the volume off `mydg03`, use the following command:

```
# 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`.

The Veritas Enterprise Administrator (VEA) has a graphical user interface (GUI), which provides an easier way to move pieces of volumes between disks. You may find that approach 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.

Warning: Striping a volume, or splitting a volume across multiple disks, increases the chance that a disk failure results in failure of that volume. For example, if five volumes are striped across the same five disks, then failure of any one of the five disks requires that all five volumes be restored from a backup. If each volume were on a separate disk, only one volume would need to be restored. Use mirroring or RAID-5 to reduce the chance that a single disk failure results in failure of a large number of volumes.

Note that file systems and databases typically shift their use of allocated space over time, so this position-specific information on a volume is often not useful. Databases are reasonable candidates for moving to non-busy disks if the space used by a particularly busy index or table can be identified.

Examining the ratio of reads to writes helps to identify volumes that can be mirrored to improve their performance. If the read-to-write ratio is high, mirroring can increase performance as well as reliability. The ratio of reads to writes where mirroring can improve performance depends greatly on the disks, the disk controller, whether multiple controllers can be used, and the speed of the system bus. If a particularly busy volume has a high ratio of reads to writes, it is likely that mirroring can significantly improve performance of that volume.

Using I/O tracing

I/O statistics provide the data for basic performance analysis; I/O traces serve for more detailed analysis. With an I/O trace, focus is narrowed to obtain an event trace for a specific workload. This helps to explicitly identify the location and size of a hot spot, as well as which application is causing it.

Using data from I/O traces, real work loads on disks can be simulated and the results traced. By using these statistics, you can anticipate system limitations and plan for additional resources.

See “Gathering and displaying I/O statistics” on page 188.

See “Specifying the I/O policy” on page 197.

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.

See "Changing the values of tunables" on page 555.

Tuning guidelines for large systems

On smaller systems (with fewer 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` command to select from, and also allow for the creation of larger volumes. Smaller disk groups do not require as large a configuration database and so can exist with smaller private regions. Very large disk groups can eventually exhaust the private region size in the disk group 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 of 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 maximum of 4. Distribute the copies across separate controllers or targets to enhance the effectiveness of this redundancy.

To set the number of configuration copies for a new disk group, use the `nconfig` operand with the `vxdg init` command.

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

You can also change the number of copies for an existing group by using the `vxedit set` command. For example, to configure five configuration copies for the disk group, `bigdg`, use the following command:

```
# vxedit set nconfig=5 bigdg
```

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

Changing the values of tunables

Tunables can be modified by editing the file `/kernel/drv/vxio.conf` for most VxVM kernel tunables. The system must be shut down and rebooted for the change to take effect.

Warning: If you modify `/kernel/drv/vxio.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;
```

Warning: 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` are listed in the output under the "System properties." heading for the `vxio` 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>
   .
   .
   .
```

For more information, see the `prtconf`(1M) and `driver.conf`(4) manual pages.

DMP tunables are set online (without requiring a reboot) by using the `vxdmpadm` command as shown here:

```
# vxdmpadm settune dmp_tunable=value
```

The values of these tunables can be displayed by using this command:

```
# vxdmpadm gettune [dmp_tunable]
```

Tunable parameters for VxVM

Table 16-1 lists the kernel tunable parameters for VxVM.

Table 16-1 Kernel tunable parameters for VxVM

Parameter	Description
vol_checkpt_default	The interval at which utilities performing recoveries or resynchronization operations load the current offset into the kernel as a checkpoint. A system failure during such operations does not require a full recovery, but can continue from the last reached checkpoint. The default value is 20480 sectors (10MB). Increasing this size reduces the overhead of checkpoints on recovery operations at the expense of additional recovery following a system failure during a recovery.
vol_default_iodelay	The count in clock ticks for which utilities pause if they have been directed to reduce the frequency of issuing I/O requests, but have not been given a specific delay time. This tunable is used by utilities performing operations such as resynchronizing mirrors or rebuilding RAID-5 columns. The default value is 50 ticks. Increasing this value results in slower recovery operations and consequently lower system impact while recoveries are being performed.

Table 16-1 Kernel tunable parameters for VxVM *(continued)*

Parameter	Description
vol_fmr_logsz	The maximum size in kilobytes of the bitmap that Non-Persistent FastResync uses to track changed blocks in a volume. The number of blocks in a volume that are mapped to each bit in the bitmap depends on the size of the volume, and this value changes if the size of the volume is changed. For example, if the volume size is 1 gigabyte and the system block size is 512 bytes, a value for this tunable of 4 yields a map that contains 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 this tunable on the master node overrides the tunable values on the slave nodes, if these values are different. Because the value of a shared volume can change, the value of this tunable is retained for the life of the volume. In configurations which have thousands of mirrors with attached snapshot plexes, the total memory overhead can represent a significantly higher overhead in memory consumption than is usual for VxVM. The default value is 4KB. The maximum and minimum permitted values are 1KB and 8KB. **Note:** The value of this tunable does not have any effect on Persistent FastResync.
vol_kmsg_resend_period	This is an obsolete tunable parameter. Use vol_kmsg_resend_period_usecs instead. If specified in /kernel/drv/vxio.conf, the value is internally converted to microseconds, and applied to vol_kmsg_resend_period_usecs instead.

Table 16-1 Kernel tunable parameters for VxVM *(continued)*

Parameter	Description
vol_kmsg_resend_period_usecs	The value in microseconds of the kernel message (KMSG) resend period that is used by the clustering functionality of VxVM with SunCluster. The default value is 3000000 microseconds (3 seconds). This tunable should be used instead of vol_kmsg_resend_period from release 5.0 onward as it allows finer granularity to be applied to performance tuning.
vol_kmsg_send_period	This is an obsolete tunable parameter. Use vol_kmsg_send_period_usecs instead. If specified in /kernel/drv/vxio.conf, the value is internally converted to microseconds, and applied to vol_kmsg_send_period_usecs instead.
vol_kmsg_send_period_usecs	The value in microseconds of the kernel message (KMSG) send period that is used by the clustering functionality of VxVM with SunCluster. The default value is 1000000 microseconds (1 second). This tunable should be used instead of vol_kmsg_send_period from release 5.0 onward as it allows finer granularity to be applied to performance tuning.
vol_max_vol	The maximum number of volumes that can be created on the system. The minimum and maximum permitted values are 1 and the maximum number of minor numbers representable on the system. The default value is 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 is 2048 sectors (1MB). The value of voliomem_maxpool_sz must be at least 10 times greater than the value of vol_maxio. If DRL sequential logging is configured, the value of voldrl_min_regionsz must be set to at least half the value of vol_maxio.

Table 16-1 Kernel tunable parameters for VxVM *(continued)*

Parameter	Description
vol_maxioctl	The maximum size of data that can be passed into VxVM via an ioctl call. Increasing this limit allows larger operations to be performed. Decreasing the limit is not generally recommended, because some utilities depend upon performing operations of a certain size and can fail unexpectedly if they issue oversized ioctl requests. The default value is 32768 bytes (32KB).
vol_maxparallelio	The number of I/O operations that the vxconfigd daemon is permitted to request from the kernel in a single VOL_VOLDIO_READ per VOL_VOLDIO_WRITE ioctl call. The default value is 256. This value should not be changed.
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 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 this tunable is 20% of the smaller of physical memory or kernel virtual memory. It is inadvisable to go over this limit, because deadlock is likely to occur. If stripes are larger than the value of this tunable, full stripe I/O requests are broken up, which prevents full-stripe read/writes. This throttles the volume I/O throughput for sequential I/O or larger I/O requests. This tunable limits the size of an I/O request at a higher level in VxVM than the level of an individual disk. For example, for an 8 by 64KB stripe, a value of 256KB only allows I/O requests that use half the disks in the stripe; thus, it cuts potential throughput in half. If you have more columns or you have used a larger interleave factor, then your relative performance is worse. This tunable must be set, as a minimum, to the size of your largest stripe (RAID-0 or RAID-5).

Table 16-1 Kernel tunable parameters for VxVM *(continued)*

Parameter	Description
`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 63.
`voldrl_max_drtregs`	The maximum number of dirty regions that can exist on the system for non-sequential DRL on volumes. A larger value may result in improved system performance at the expense of recovery time. This tunable can be used to regulate the worse-case recovery time for the system following a failure. The default value is 2048.
`voldrl_max_seq_dirty`	The maximum number of dirty regions allowed for sequential DRL. This is useful for volumes that are usually written to sequentially, such as database logs. Limiting the number of dirty regions allows for faster recovery if a crash occurs. The default value is 3.
`voldrl_min_regionsz`	The minimum number of sectors for a dirty region logging (DRL) volume region. With DRL, VxVM logically divides a volume into a set of consecutive regions. Larger region sizes tend to cause the cache hit-ratio for regions to improve. This improves the write performance, but it also prolongs the recovery time. The default value is 1024 sectors. If DRL sequential logging is configured, the value of `voldrl_min_regionsz` must be set to at least half the value of `vol_maxio`.
`voliomem_chunk_size`	The granularity of memory chunks used by VxVM when allocating or releasing system memory. A larger granularity reduces CPU overhead due to memory allocation by allowing VxVM to retain hold of a larger amount of memory. The default value is 64KB.

Table 16-1 Kernel tunable parameters for VxVM *(continued)*

Parameter	Description
`voliomem_maxpool_sz`	The maximum memory requested from the system by VxVM for internal purposes. This tunable has a direct impact on the performance of VxVM as it prevents one I/O operation from using all the memory in the system. VxVM allocates two pools that can grow up to this size, one for RAID-5 and one for mirrored volumes. A write request to a RAID-5 volume that is greater than one tenth of the pool size is broken up and performed in chunks of one tenth of the pool size. A write request to a mirrored volume that is greater than half the pool size is broken up and performed in chunks of one half of the pool size. The default value is 5% of memory up to a maximum of 128MB. The value of `voliomem_maxpool_sz` must be greater than the value of `volraid_minpool_size`. The value of `voliomem_maxpool_sz` must be at least 10 times greater than the value of `vol_maxio`.
`voliot_errbuf_dflt`	The default size of the buffer maintained for error tracing events. This buffer is allocated at driver load time and is not adjustable for size while VxVM is running. The default value is 16384 bytes (16KB). Increasing this buffer can provide storage for more error events at the expense of system memory. Decreasing the size of the buffer can result in an error not being detected via the tracing device. Applications that depend on error tracing to perform some responsive action are dependent on this buffer.
`voliot_iobuf_default`	The default size for the creation of a tracing buffer in the absence of any other specification of desired kernel buffer size as part of the trace `ioctl`. The default value is 8192 bytes (8KB). If trace data is often being lost due to this buffer size being too small, then this value can be tuned to a more generous amount.

Table 16-1 Kernel tunable parameters for VxVM *(continued)*

Parameter	Description
`voliot_iobuf_limit`	The upper limit to the size of memory that can be used for storing tracing buffers in the kernel. Tracing buffers are used by the VxVM kernel to store the tracing event records. As trace buffers are requested to be stored in the kernel, the memory for them is drawn from this pool. Increasing this size can allow additional tracing to be performed at the expense of system memory usage. Setting this value to a size greater than can readily be accommodated on the system is inadvisable. The default value is 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 the channel is not in use.

Table 16-1 Kernel tunable parameters for VxVM *(continued)*

Parameter	Description
`volpagemod_max_memsz`	The amount of memory, measured in kilobytes, that is allocated for caching FastResync and cache object metadata. The default value is 6144KB (6MB). The valid range for this tunable is from 0 to 50% of physical memory. The memory allocated for this cache is exclusively dedicated to it. It is not available for other processes or applications. Setting the value below 512KB fails if cache objects or volumes that have been prepared for instant snapshot operations are present on the system. If you do not use the FastResync or DRL features that are implemented using a version 20 DCO volume, the value can be set to 0. However, if you subsequently decide to enable these features, you can use the `vxtune` command to change the value to a more appropriate one: `# vxtune volpagemod_max_memsz` *`value`* 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 is 16384 sectors (8MB).

Table 16-1 Kernel tunable parameters for VxVM *(continued)*

Parameter	Description
volraid_rsrtransmax	The maximum number of transient reconstruct operations that can be performed in parallel for RAID-5. A transient reconstruct operation is one that occurs on a non-degraded RAID-5 volume that has not been predicted. Limiting the number of these operations that can occur simultaneously removes the possibility of flooding the system with many reconstruct operations, and so reduces the risk of causing memory starvation. The default value is 1. Increasing this size improves the initial performance on the system when a failure first occurs and before a detach of a failing object is performed, but can lead to memory starvation.

DMP tunable parameters

Table 16-2 shows the DMP parameters that can be tuned by using the `vxdmpadm settune` command.

Table 16-2 DMP parameters that are tunable

Parameter	Description
dmp_cache_open	If this parameter is set to `on`, the first open of a device that is performed by an array support library (ASL) is cached. This caching enhances the performance of device discovery by minimizing the overhead that is caused by subsequent opens by ASLs. If this parameter is set to `off`, caching is not performed. The default value is `on`.
dmp_daemon_count	The number of kernel threads that are available for servicing path error handling, path restoration, and other DMP administrative tasks. The default number of threads is 10.
dmp_delayq_interval	How long DMP should wait before retrying I/O after an array fails over to a standby path. Some disk arrays are not capable of accepting I/O requests immediately after failover. The default value is 15 seconds.

Table 16-2 DMP parameters that are tunable *(continued)*

Parameter	Description
dmp_enable_restore	If this parameter is set to on, it enables the path restoration thread to be started. See "Configuring DMP path restoration policies" on page 210. If this parameter is set to off, it disables the path restoration thread. If the path restoration thread is currently running, use the vxdmpadm stop restore command to stop the process. See "Stopping the DMP path restoration thread" on page 212.
dmp_failed_io_threshold	The time limit that DMP waits for a failed I/O request to return before the device is marked as INSANE, I/O is avoided on the path, and any remaining failed I/O requests are returned to the application layer without performing any error analysis. The default value is 57600 seconds (16 hours). See "Configuring the response to I/O failures" on page 206. See "Configuring the I/O throttling mechanism" on page 207.
dmp_fast_recovery	Whether DMP should try to obtain SCSI error information directly from the HBA interface. Setting the value to on can potentially provide faster error recovery, provided that the HBA interface supports the error enquiry feature. If this parameter is set to off, the HBA interface is not used. The default setting is on.
dmp_health_time	DMP detects intermittently failing paths, and prevents I/O requests from being sent on them. The value of dmp_health_time represents the time in seconds for which a path must stay healthy. If a path's state changes back from enabled to disabled within this time period, DMP marks the path as intermittently failing, and does not re-enable the path for I/O until dmp_path_age seconds elapse. The default value is 60 seconds. A value of 0 prevents DMP from detecting intermittently failing paths.

Table 16-2 DMP parameters that are tunable *(continued)*

Parameter	Description
dmp_log_level	The level of detail that is displayed for DMP console messages. The following level values are defined: 1 — Display all DMP log messages that existed in releases before 5.0. 2 — Display level 1 messages plus messages that relate to I/O throttling, suspected paths, repeated path failures and DMP node migration. 3 — Display level 1 and 2 messages plus messages that relate to I/O errors, I/O error analysis and path media errors. 4 — Display level 1, 2 and 3 messages plus messages that relate to setting or changing attributes on a path. The default value is 1.
dmp_monitor_fabric	Whether the Event Source daemon (vxesd) uses the Storage Networking Industry Association (SNIA) HBA API. This API allows DDL to improve the performance of failover by collecting information about the SAN topology and by monitoring fabric events. If this parameter is set to on, DDL uses the SNIA HBA API. (Note that the vendor-provided ASL must also support the use of the SNIA HBA API.) If this parameter is set to off, the SNIA HBA API is not used. The default setting is off for releases before 5.0 that have been patched to support this DDL feature. The default setting is on for 5.0 and later releases.
dmp_path_age	The time for which an intermittently failing path needs to be monitored as healthy before DMP again tries to schedule I/O requests on it. The default value is 300 seconds. A value of 0 prevents DMP from detecting intermittently failing paths.

Table 16-2 DMP parameters that are tunable *(continued)*

Parameter	Description
dmp_pathswitch_blks_shift	The default number of contiguous I/O blocks that are sent along a DMP path to an Active/Active array before switching to the next available path. The value is expressed as the integer exponent of a power of 2; for example 11 represents 2048 blocks. The default value of this parameter is set to 11. In this case, 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. 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 for an array. See "Specifying the I/O policy" on page 197.
dmp_probe_idle_lun	If DMP statistics gathering is enabled, set this tunable to 1 (default) to have the DMP path restoration thread probe idle LUNs. Set this tunable to 0 to turn off this feature. (Idle LUNs are VM disks on which no I/O requests are scheduled.) The value of this tunable is only interpreted when DMP statistics gathering is enabled. Turning off statistics gathering also disables idle LUN probing. The default value is 1.
dmp_queue_depth	The maximum number of queued I/O requests on a path during I/O throttling. The default value is 20. A value can also be set for paths to individual arrays by using the vxdmpadm command. See "Configuring the I/O throttling mechanism" on page 207.

Table 16-2 DMP parameters that are tunable *(continued)*

Parameter	Description
dmp_restore_interval	The interval attribute specifies how often the path restoration thread examines the paths. Specify the time in seconds. The default value is 300. The value of this tunable can also be set using the vxdmpadm start restore command. See "Configuring DMP path restoration policies" on page 210.
dmp_restore_cycles	If the DMP restore policy is check_periodic, the number of cycles after which the check_all policy is called. The default value is 10. The value of this tunable can also be set using the vxdmpadm start restore command. See "Configuring DMP path restoration policies" on page 210.
dmp_restore_policy	The DMP restore policy, which can be set to one of the following values: ■ check_all ■ check_alternate ■ check_disabled ■ check_periodic The default value is check_disabled The value of this tunable can also be set using the vxdmpadm start restore command. See "Configuring DMP path restoration policies" on page 210.
dmp_retry_count	If an inquiry succeeds on a path, but there is an I/O error, the number of retries to attempt on the path. The default value is 5. A value can also be set for paths to individual arrays by using the vxdmpadm command. See "Configuring the response to I/O failures" on page 206.

Table 16-2 DMP parameters that are tunable *(continued)*

Parameter	Description
dmp_scsi_timeout	Determines the timeout value to be set for any SCSI command that is sent via DMP. If the HBA does not receive a response for a SCSI command that it has sent to the device within the timeout period, the SCSI command is returned with a failure error code. The default value is 30 seconds.
dmp_stat_interval	The time interval between gathering DMP statistics. The default and minimum value is 1 second.

Appendix A

Using Veritas Volume Manager commands

This appendix includes the following topics:

- About Veritas Volume Manager commands
- Online manual pages

About Veritas Volume Manager commands

Most Veritas Volume Manager (VxVM) commands (excepting daemons, library commands and supporting scripts) are linked to the `/usr/sbin` directory from the `/opt/VRTS/bin` directory. It is recommended that you add the following directories to your PATH environment variable:

- If you are using the Bourne or Korn shell (`sh` or `ksh`), use the commands:

```
$ PATH=$PATH:/usr/sbin:/opt/VRTS/bin:/opt/VRTSvxfs/sbin:\
  /opt/VRTSdbed/bin:/opt/VRTSdb2ed/bin:/opt/VRTSsybed/bin:\
  /opt/VRTSob/bin
$ MANPATH=/usr/share/man:/opt/VRTS/man:$MANPATH
$ export PATH MANPATH
```

- If you are using a C shell (`csh` or `tcsh`), use the commands:

```
% set path = ( $path /usr/sbin /opt/VRTSvxfs/sbin \
  /opt/VRTSdbed/bin /opt/VRTSdb2ed/bin /opt/VRTSsybed/bin \
  /opt/VRTSob/bin /opt/VRTS/bin )
% setenv MANPATH /usr/share/man:/opt/VRTS/man:$MANPATH
```

Note: If you have not installed database software, you can omit `/opt/VRTSdbed/bin`, `/opt/VRTSdb2ed/bin` and `/opt/VRTSsybed/bin`. Similarly, `/opt/VRTSvxfs/bin` is only required to access some 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.

See "Online manual pages" on page 593.

Commands and scripts that are provided to support other commands and scripts, and which are not intended for general use, are not located in `/opt/VRTS/bin` and do not have manual pages.

Commonly-used commands are summarized in the following tables:

- Table A-1 lists commands for obtaining information about objects in VxVM.
- Table A-2 lists commands for administering disks.
- Table A-3 lists commands for creating and administering disk groups.
- Table A-4 lists commands for creating and administering subdisks.
- Table A-5 lists commands for creating and administering plexes.
- Table A-6 lists commands for creating volumes.
- Table A-7 lists commands for administering volumes.
- Table A-8 lists commands for monitoring and controling tasks in VxVM.

Table A-1 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 155. Example: `# vxdisk -g mydg list`

Table A-1 Obtaining information about objects in VxVM *(continued)*

Command	Description
`vxdg list [diskgroup]`	Lists information about disk groups. See "Displaying disk group information" on page 220. Example: `# vxdg list mydg`
`vxdg -s list`	Lists information about shared disk groups. See "Listing shared disk groups" on page 502. Example: `# vxdg -s list`

`vxinfo [-g diskgroup] [volume ...]`	Displays information about the accessibility and usability of volumes.
	See the *Veritas Volume Manager Troubleshooting Guide*.
	Example:
	`# vxinfo -g mydg myvol1 \`
	`  myvol2`

`vxprint -hrt [-g diskgroup] [object ...]`	Prints single-line information about objects in VxVM.
	See "Displaying volume information" on page 328.
	Example:
	`# vxprint -g mydg myvol1 \`
	`  myvol2`

Table A-1 Obtaining information about objects in VxVM *(continued)*

Command	Description
`vxprint -st [-g diskgroup] [subdisk ...]`	Displays information about subdisks. See "Displaying subdisk information" on page 272. Example: `# vxprint -st -g mydg`
`vxprint -pt [-g diskgroup] [plex ...]`	Displays information about plexes. See "Displaying plex information" on page 283. Example: `# vxprint -pt -g mydg`

Table A-2 Administering disks

Command	Description
`vxdiskadm`	Administers disks in VxVM using a menu-based interface.
`vxdiskadd [devicename ...]`	Adds a disk specified by device name. See "Using vxdiskadd to put a disk under VxVM control" on page 122. Example: `# vxdiskadd c0t1d0`
`vxedit [-g diskgroup] rename \ olddisk newdisk`	Renames a disk under control of VxVM. See "Renaming a disk" on page 153. Example: `# vxedit -g mydg rename \ mydg03 mydg02`

Table A-2 Administering disks *(continued)*

Command	Description

`vxedit [-g diskgroup] set \ reserve=on|off diskname`	Sets aside/does not set aside a disk from use in a disk group.
	See "Reserving disks" on page 154.
	Examples:
	`# vxedit -g mydg set \ reserve=on mydg02`
	`# vxedit -g mydg set \ reserve=off mydg02`

`vxedit [-g diskgroup] set \ nohotuse=on|off diskname`	Does not/does allow free space on a disk to be used for hot-relocation.
	See "Excluding a disk from hot-relocation use" on page 465.
	See "Making a disk available for hot-relocation use" on page 466.
	Examples:
	`# vxedit -g mydg set \ nohotuse=on mydg03`
	`# vxedit -g mydg set \ nohotuse=off mydg03`

`vxedit [-g diskgroup] set \ spare=on|off diskname`	Adds/removes a disk from the pool of hot-relocation spares.
	See "Marking a disk as a hot-relocation spare" on page 463.
	See "Removing a disk from use as a hot-relocation spare" on page 464.
	Examples:
	`# vxedit -g mydg set \ spare=on mydg04`
	`# vxedit -g mydg set \ spare=off mydg04`

Table A-2 Administering disks *(continued)*

Command	Description
`vxdisk offline devicename`	Takes a disk offline. See "Taking a disk offline" on page 153. Example: `# vxdisk offline c0t1d0`
`vxdg -g diskgroup rmdisk diskname`	Removes a disk from its disk group. See "Removing a disk from a disk group" on page 223. 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 223. Example: `# vxdiskunsetup c0t3d0`

Table A-3 Creating and administering disk groups

Command	Description

`vxdg [-s] init diskgroup \`
`[diskname=]devicename`
	Creates a disk group using a pre-initialized disk.
	See "Creating a disk group" on page 221.
	See "Creating a shared disk group" on page 503.
	Example:
	`# vxdg init mydg \`
	`mydg01=c0t1d0`

Table A-3 Creating and administering disk groups *(continued)*

Command	Description
`vxsplitlines -g diskgroup`	Reports conflicting configuration information. See "Handling conflicting configuration copies" on page 244. Example: `# vxsplitlines -g mydg`
`vxdg [-n newname] deport diskgroup`	Deports a disk group and optionally renames it. See "Deporting a disk group" on page 224. 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 226. Example: `# vxdg -n newdg import mydg`

`vxdg [-n newname] -s import diskgroup`	Imports a disk group as shared by a cluster, and optionally renames it.
	See "Importing disk groups as shared" on page 504.
	Example:
	`# vxdg -n newsdg -s import \`
	`  mysdg`

Table A-3 Creating and administering disk groups *(continued)*

Command	Description
`vxdg [-o expand] listmove sourcedg \ targetdg object ...`	Lists the objects potentially affected by moving a disk group. See "Listing objects potentially affected by a move" on page 255. Example: `# vxdg -o expand listmove \ mydg newdg myvol1`
`vxdg [-o expand] move sourcedg \ targetdg object ...`	Moves objects between disk groups. See "Moving objects between disk groups" on page 257. Example: `# vxdg -o expand move mydg \ newdg myvol1`
`vxdg [-o expand] split sourcedg \ targetdg object ...`	Splits a disk group and moves the specified objects into the target disk group. See "Splitting disk groups" on page 260. Example: `# vxdg -o expand split mydg \ newdg myvol2 myvol3`
`vxdg join sourcedg targetdg`	Joins two disk groups. See "Joining disk groups" on page 261. Example: `# vxdg join newdg mydg`

Table A-3 Creating and administering disk groups *(continued)*

Command	Description
`vxdg -g diskgroup set \ activation=ew|ro|sr|sw|off`	Sets the activation mode of a shared disk group in a cluster. See "Changing the activation mode on a shared disk group" on page 506. Example: `# vxdg -g mysdg set \ activation=sw`
`vxrecover -g diskgroup -sb`	Starts all volumes in an imported disk group. See "Moving disk groups between systems" on page 238. Example: `# vxrecover -g mydg -sb`
`vxdg destroy diskgroup`	Destroys a disk group and releases its disks. See "Destroying a disk group" on page 263. Example: `# vxdg destroy mydg`

Table A-4 Creating and administering subdisks

Command	Description
`vxmake [-g diskgroup] sd subdisk \ diskname,offset,length`	Creates a subdisk. See Creating subdisks. Example: `# vxmake -g mydg sd \ mydg02-01 mydg02,0,8000`

Table A-4 Creating and administering subdisks *(continued)*

Command	Description
`vxsd [-g diskgroup] assoc plex \ subdisk...`	Associates subdisks with an existing plex. See "Associating subdisks with plexes" on page 275. 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 275. 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 273. 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 274. Example: `# vxsd -g mydg -s 1000m \ split mydg03-02 mydg03-02 \ mydg03-03`

Table A-4 Creating and administering subdisks *(continued)*

Command	Description
`vxsd [-g diskgroup] join \ sd1 sd2 ... subdisk`	Joins two or more subdisks. See "Joining subdisks" on page 274. Example: `# vxsd -g mydg join \ mydg03-02 mydg03-03 \ mydg03-02`
`vxassist [-g diskgroup] move \ volume !olddisk newdisk`	Relocates subdisks in a volume between disks. See "Moving relocated subdisks using vxassist" on page 469. Example: `# vxassist -g mydg move \ myvol !mydg02 mydg05`
`vxunreloc [-g diskgroup] original_disk`	Relocates subdisks to their original disks. See "Moving relocated subdisks using vxunreloc" on page 470. Example: `# vxunreloc -g mydg mydg01`
`vxsd [-g diskgroup] dis subdisk`	Dissociates a subdisk from a plex. See "Dissociating subdisks from plexes" on page 277. Example: `# vxsd -g mydg dis mydg02-01`
`vxedit [-g diskgroup] rm subdisk`	Removes a subdisk. See "Removing subdisks" on page 278. Example: `# vxedit -g mydg rm mydg02-01`

Table A-4 Creating and administering subdisks *(continued)*

Command	Description

`vxsd [-g diskgroup] -o rm dis subdisk`	Dissociates and removes a subdisk from a plex.
	See "Dissociating subdisks from plexes" on page 277.
	Example:
	`# vxsd -g mydg -o rm dis \`
	`  mydg02-01`

Table A-5 Creating and administering plexes

Command	Description

`vxmake [-g diskgroup] plex plex \`
`sd=subdisk1[,subdisk2,...]`
	Creates a concatenated plex.
	See "Creating plexes" on page 282.
	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 282.
	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 287.
	See "Reattaching plexes" on page 289.
	Example:
	`# vxplex -g mydg att vol01 \`
	`  vol01-02`

Table A-5 Creating and administering plexes *(continued)*

Command	Description
`vxplex [-g diskgroup] det plex`	Detaches a plex. See "Detaching plexes" on page 289. Example: `# vxplex -g mydg det vol01-02`
`vxmend [-g diskgroup] off plex`	Takes a plex offline for maintenance. See "Taking plexes offline" on page 288. Example: `# vxmend -g mydg off vol02-02`
`vxmend [-g diskgroup] on plex`	Re-enables a plex for use. See "Reattaching plexes" on page 289. Example: `# vxmend -g mydg on vol02-02`
`vxplex [-g diskgroup] mv oldplex \ newplex`	Replaces a plex. See "Moving plexes" on page 290. Example: `# vxplex -g mydg mv \ vol02-02 vol02-03`
`vxplex [-g diskgroup] cp volume \ newplex`	Copies a volume onto a plex. See "Copying plexes" on page 291. Example: `# vxplex -g mydg cp vol02 \ vol03-01`

Table A-5 Creating and administering plexes *(continued)*

Command	Description
`vxmend [-g diskgroup] fix clean plex`	Sets the state of a plex in an unstartable volume to CLEAN. See "Reattaching plexes" on page 289. 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 291. Example: `# vxplex -g mydg -o rm dis \ vol03-01`

Table A-6 Creating volumes

Command	Description
`vxassist [-g diskgroup] maxsize \ layout=layout [attributes]`	Displays the maximum size of volume that can be created. See "Discovering the maximum size of a volume" on page 304. Example: `# vxassist -g mydg maxsize \ layout=raid5 nlog=2`

Table A-6 Creating volumes *(continued)*

Command	Description
`vxassist -b [-g diskgroup] make \ volume length [layout=layout] \ [attributes]`	Creates a volume. See "Creating a volume on any disk" on page 305. See "Creating a volume on specific disks" on page 305. Example: `# vxassist -b -g mydg make \ myvol 20g layout=concat \ mydg01 mydg02`
`vxassist -b [-g diskgroup] make \ volume length layout=mirror \ [nmirror=N][attributes]`	Creates a mirrored volume. See "Creating a mirrored volume" on page 310. Example: `# vxassist -b -g mydg make \ mymvol 20g layout=mirror \ nmirror=2`
`vxassist -b [-g diskgroup] make \ volume length layout=layout \ exclusive=on [attributes]`	Creates a volume that may be opened exclusively by a single node in a cluster. See "Creating volumes with exclusive open access by a node" on page 507. Example: `# vxassist -b -g mysdg make \ mysmvol 20g layout=mirror \ exclusive=on`

Table A-6 Creating volumes *(continued)*

Command	Description
`vxassist -b [-g diskgroup] make \ volume length layout={stripe|raid5} \ [stripeunit=W] [ncol=N] \ [attributes]`	Creates a striped or RAID-5 volume. See "Creating a striped volume" on page 316. See "Creating a RAID-5 volume" on page 319. Example: `# vxassist -b -g mydg make \ mysvol 20g layout=stripe \ stripeunit=32 ncol=4`
`vxassist -b [-g diskgroup] make \ volume length layout=mirror \ mirror=ctlr [attributes]`	Creates a volume with mirrored data plexes on separate controllers. See "Mirroring across targets, controllers or enclosures" on page 318. Example: `# vxassist -b -g mydg make \ mymcvol 20g layout=mirror \ mirror=ctlr`
`vxmake -b [-g diskgroup] \ -Uusage_type vol volume \ [len=length] plex=plex,...`	Creates a volume from existing plexes. See "Creating a volume using vxmake" on page 322. Example: `# vxmake -g mydg -Uraid5 \ vol r5vol \ plex=raidplex,raidlog1,\ raidlog2`

Table A-6 Creating volumes *(continued)*

Command	Description
`vxvol [-g diskgroup] start volume`	Initializes and starts a volume for use. See "Initializing and starting a volume" on page 324. See "Starting a volume" on page 335. 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 324. Example: `# vxvol -g mydg init zero \ myvol`

Table A-7 Administering volumes

Command	Description
`vxassist [-g diskgroup] mirror \ volume [attributes]`	Adds a mirror to a volume. See "Adding a mirror to a volume " on page 336. 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 339. Example: `# vxassist -g mydg remove \ mirror myvol !mydg11`

Table A-7 Administering volumes *(continued)*

Command	Description
`vxassist [-g diskgroup] \ {growto|growby} volume length`	Grows a volume to a specified size or by a specified amount. See "Resizing volumes using vxassist" on page 355. 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 355. Example: `# vxassist -g mydg shrinkto \ myvol 20g`
`vxresize -b -F vxfs [-g diskgroup] \ volume length diskname ...`	Resizes a volume and the underlying Veritas File System. See "Resizing volumes using vxresize" on page 353. Example: `# vxassist -b -F vxfs \ -g mydg myvol 20g mydg10 \ mydg11`
`vxsnap [-g diskgroup] prepare volume [drl=on|sequential|off]`	Prepares a volume for instant snapshots and for DRL logging. See "Preparing a volume for DRL and instant snapshots" on page 341. Example: `# vxsnap -g mydg prepare \ myvol drl=on`

Table A-7 Administering volumes *(continued)*

Command	Description

`vxsnap [-g diskgroup] make \`
`source=volume\`
`/newvol=snapvol\`
`[/nmirror=number]`
	Takes a full-sized instant snapshot of a volume by breaking off plexes of the original volume.
	See "Creating instant snapshots" on page 387.
	Example:
	`# vxsnap -g mydg make \`
	`source=myvol/\`
	`newvol=mysnpvol/\`
	`nmiror=2`

`vxsnap [-g diskgroup] make \`
`source=volume/snapvol=snapvol`
	Takes a full-sized instant snapshot of a volume using a prepared empty volume. See Creating a volume for use as a full-sized instant or linked break-off snapshot.
	See "Creating instant snapshots" on page 387.
	Example:
	`# vxsnap -g mydg make \`
	`source=myvol/snapvol=snpvol`

`vxmake [-g diskgroup] cache \`
`cache_object cachevolname=volume \`
`[regionsize=size]`
	Creates a cache object for use by space-optimized instant snapshots.
	See "Creating a shared cache object" on page 391.
	A cache volume must have already been created, as shown in this example:
	`# vxassist -g mydg make \`
	`cvol 1g layout=mirror \`
	`init=active mydg16 mydg17`
	`# vxmake -g mydg cache cobj \`
	`cachevolname=cvol`

Table A-7 Administering volumes *(continued)*

Command	Description
`vxsnap [-g diskgroup] make \ source=volume/newvol=snapvol\ /cache=cache_object`	Takes a space-optimized instant snapshot of a volume. See "Creating instant snapshots" on page 387. Example: `# vxsnap -g mydg make \ source=myvol/\ newvol=mysosvol/\ cache=cobj`
`vxsnap [-g diskgroup] refresh snapshot`	Refreshes a snapshot from its original volume. See "Refreshing an instant snapshot" on page 409. Example: `# vxsnap -g mydg refresh \ mysnpvol`
`vxsnap [-g diskgroup] dis snapshot`	Turns a snapshot into an independent volume. See "Dissociating an instant snapshot" on page 412. Example: `# vxsnap -g mydg dis mysnpvol`
`vxsnap [-g diskgroup] unprepare \ volume`	Removes support for instant snapshots and DRL logging from a volume. See "Removing support for DRL and instant snapshots from a volume" on page 346. Example: `# vxsnap -g mydg unprepare \ myvol`

Table A-7 Administering volumes *(continued)*

Command	Description
`vxassist [-g diskgroup] relayout \ volume [layout=layout] \ [relayout_options]`	Performs online relayout of a volume. See "Performing online relayout" on page 363. Example: `# vxassist -g mydg relayout \ vol2 layout=stripe`
`vxassist [-g diskgroup] relayout \ volume layout=raid5 \ stripeunit=W \ ncol=N`	Relays out a volume as a RAID-5 volume with stripe width *W* and *N* columns. See "Performing online relayout" on page 363. Example: `# vxassist -g mydg relayout \ vol3 layout=raid5 \ stripeunit=16 ncol=4`
`vxrelayout [-g diskgroup] -o bg \ reverse volume`	Reverses the direction of a paused volume relayout. See "Volume sets" on page 77. 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 370. Example: `# vxassist -g mydg convert \ vol3 layout=stripe-mirror`

Table A-7 Administering volumes *(continued)*

Command	Description
`vxassist [-g diskgroup] remove \ volume volume`	Removes a volume. See "Removing a volume" on page 359. Example: `# vxassist -g mydg remove \ myvol`

Table A-8 Monitoring and controlling tasks

Command	Description
`command [-g diskgroup] -t tasktag \ [options] [arguments]`	Specifies a task tag to a VxVM command. See "Specifying task tags" on page 332. Example: `# vxrecover -g mydg \ -t mytask -b mydg05`
`vxtask [-h] [-g diskgroup] list`	Lists tasks running on a system. See "Using the vxtask command" on page 333. Example: `# vxtask -h -g mydg list`
`vxtask monitor task`	Monitors the progress of a task. See "Using the vxtask command" on page 333. Example: `# vxtask monitor mytask`

Table A-8 Monitoring and controlling tasks *(continued)*

Command	Description
`vxtask pause task`	Suspends operation of a task. See "Using the vxtask command" on page 333. Example: `# vxtask pause mytask`
`vxtask -p [-g diskgroup] list`	Lists all paused tasks. See "Using the vxtask command" on page 333. Example: `# vxtask -p -g mydg list`
`vxtask resume task`	Resumes a paused task. See "Using the vxtask command" on page 333. Example: `# vxtask resume mytask`
`vxtask abort task`	Cancels a task and attempts to reverse its effects. See "Using the vxtask command" on page 333. Example: `# vxtask abort mytask`

Online manual pages

Manual pages are organized into the following sections:

1M	Administrative commands.
4	File formats.

7 Device driver interfaces.

Section 1M — administrative commands

Table A-9 lists the manual pages in section 1M for commands that are used to administer Veritas Volume Manager.

Table A-9 Section 1M manual pages

Name	Description
`vxapslice`	Manage an area of disk for use by an Alternate Pathing database.
`vxassist`	Create, relayout, convert, mirror, backup, grow, shrink, delete, and move volumes.
`vxbootsetup`	Set up system boot information on a Veritas Volume Manager disk.
`vxcache`	Administer the cache object for space-optimized snapshots.
`vxcached`	Resize cache volumes when required.
`vxcdsconvert`	Make disks and disk groups portable between systems.
`vxclustadm`	Start, stop, and reconfigure a cluster.
`vxcmdlog`	Administer command logging.
`vxconfigbackup`	Back up disk group configuration.
`vxconfigbackupd`	Disk group configuration backup daemon.
`vxconfigd`	Veritas Volume Manager configuration daemon
`vxconfigrestore`	Restore disk group configuration.
`vxdarestore`	Restore simple or nopriv disk access records.
`vxdco`	Perform operations on version 0 DCO objects and DCO volumes.
`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.

Table A-9 Section 1M manual pages *(continued)*

Name	Description
vxdiskadd	Add one or more disks for use with Veritas Volume Manager.
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 I/O threads.
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.
vxpool	Create and administer ISP storage pools.

Table A-9 Section 1M manual pages *(continued)*

Name	Description
vxprint	Display records from the Veritas Volume Manager configuration.
vxr5check	Verify RAID-5 volume parity.
vxreattach	Reattach disk drives that have become accessible again.
vxrecover	Perform volume recovery operations.
vxrelayout	Convert online storage from one layout to another.
vxrelocd	Monitor Veritas Volume Manager for failure events and relocate failed subdisks.
vxresize	Change the length of a volume containing a file system.
vxrootmir	Mirror areas necessary for booting onto a new disk.
vxsd	Perform Veritas Volume Manager operations on subdisks.
vxse	Storage Expert rules.
vxsnap	Enable DRL on a volume, and create and administer instant snapshots.
vxsparecheck	Monitor for disk failure, and replace failed disks.
vxspcshow	Add SAN Access Layer (SAL) user details or display device properties obtained from SAL.
vxsplitlines	Show disks with conflicting configuration copies in a cluster.
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.

Table A-9 Section 1M manual pages *(continued)*

Name	Description
vxunroot	Remove Veritas Volume Manager hooks from encapsulated root volumes.
vxusertemplate	Create and administer ISP user templates.
vxvol	Perform Veritas Volume Manager operations on volumes.
vxvoladm	Create and administer ISP application volumes on allocated storage.
vxvoladmtask	Administer ISP tasks.
vxvset	Create and administer volume sets.

Section 4 — file formats

Table A-10 lists the manual pages in section 4 that describe the format of files that are used by Veritas Volume Manager.

Table A-10 Section 4 manual pages

Name	Description
vol_pattern	Disk group search specifications.
vxmake	vxmake description file.

Section 7 — device driver interfaces

Table A-11 lists the manual pages in section 7 that describe the interfaces to Veritas Volume Manager devices.

Table A-11 Section 7 manual pages

Name	Description
vxconfig	Configuration device.
vxdmp	Dynamic multipathing device.
vxinfo	General information device.
vxio	Virtual disk device.

Table A-11 Section 7 manual pages *(continued)*

Name	Description
`vxiod`	I/O daemon process control device.
`vxtrace`	I/O tracing device.

Appendix B

Veritas Storage Foundation support for Solaris Zones

This appendix includes the following topics:

- About Solaris Zones
- Adding VxFS file systems to a non-global zone
- Delegating VxFS file systems to a non-global zone
- Veritas Storage Foundation Cluster File System mounts
- Concurrent I/O access in non-global zones
- Veritas extension for Oracle Disk Manager
- Exporting VxVM volumes to a non-global zone
- Software limitations of Storage Foundation support of non-global zones

About Solaris Zones

Solaris Zones is a software partitioning technology, which provides a means of virtualizing operating system services to create an isolated environment for running applications. This isolation prevents processes that are running in one zone from monitoring or affecting processes running in other zones.

See the *System Administration Guide: Solaris Containers--Resource Management and Solaris Zones* Solaris operating environment document.

Sun Microsystems, Inc. provides regular updates and patches for the Solaris Zones feature. Contact Sun Microsystems, Inc. for more information.

Adding VxFS file systems to a non-global zone

VxFS file systems that were previously created in the global zone can be made available in the non-global zone using a loopback file system mount. This functionality is especially useful when the sole purpose of making the file system available in the non-global zone is to share access of this file system with one or more non-global zones. For example, if a configuration file is available in a particular file system and this configuration file is required by the non-global zone, then the file system can be shared with the non-global zone using a loopback file system mount.

The following commands share access of file system `/mnt1` as a loopback file system mount with the non-global zone `myzone`:

```
global# zonecfg -z myzone
zonecfg:myzone> add fs
zonecfg:myzone:fs> set dir=/mnt1
zonecfg:myzone:fs> set special=/mnt1
zonecfg:myzone:fs> set type=lofs
zonecfg:myzone:fs> end
zonecfg:myzone> commit
```

The value of `dir` is a directory in the non-global zone. The value of `special` is a directory in the global zone to be mounted in the non-global zone.

Caution: Sharing file systems with non-global zones through a loopback file system mount makes the file system available for simultaneous access from all the non-global zones. This method should be used only when you want shared access to the file system.

The loopback file system mount mode of sharing file systems in the non-global zones is supported in Veritas File System 4.1 and later.

Delegating VxFS file systems to a non-global zone

Exclusive access of a VxFS file system can be delegated to a non-global zone by direct mounting the file system in the non-global zone. Using direct mounts limits the visibility of and access to the file system to only the non-global zone that has direct mounted this file system.

Delegating file systems to non-global zone using direct mounts is supported in Veritas File System 4.1 Maintenance Pack 1 and later.

To direct mount a VxFS file system in a non-global zone, the directory to mount must be in the non-global zone and the mount must take place from the global zone. The following procedure mounts the directory `dirmnt` in the non-global zone `myzone` with a mount path of `/zonedir/myzone/root/dirmnt`.

Note: VxFS entries in the global zone `/etc/vfstab` file for non-global zone direct mounts are not supported, as the non-global zone may not yet be booted at the time of `/etc/vfstab` execution.

Once a file system has been delegated to a non-global zone through a direct mount, the mount point will be visible in the global zone through the `mount` command, but not through the `df` command.

To direct mount a VxFS file system in a non-global zone

1 Log in to the zone and make the mount point:

```
global# zlogin myzone
myzone# mkdir dirmnt
myzone# exit
```

2 Mount the VxFS file system:

```
global# mount -F vxfs /dev/vx/dsk/dg/vol1 /zonedir/zone1/root/dirmnt
```

3 Log in to the non-global zone and ensure that the file system is mounted:

```
global# zlogin myzone
myzone# df | grep dirmnt
/dirmnt (/dirmnt):142911566 blocks 17863944 files
```

Veritas Storage Foundation Cluster File System mounts

Veritas Storage Foundation Cluster File System (SFCFS) provides support for the same file system to be made available from multiple nodes that have been grouped together as a cluster. VxFS supports the sharing or delegation of cluster-mounted file systems in the non-global zone.

See "Adding VxFS file systems to a non-global zone" on page 600.

See "Delegating VxFS file systems to a non-global zone" on page 600.

The requirements to support SFCFS in non-global zones parallels that of SFCFS support in global zones. Some key points are as follows:

- Both lofs and direct mount are supported; Symantec recommends direct mount
- The device must be visible and shared on all nodes
- The mount point path must be the same on all nodes
- The zone name and configuration should be the same on all nodes

Support for SFCFS in a non-global zone is available in Veritas File System 5.0 Maintenance Pack 1 and later.

Concurrent I/O access in non-global zones

Concurrent I/O allows multiple processes to read from or write to the same file without blocking other `read`(2) or `write`(2) calls. POSIX semantics requires `read` and `write` calls to be serialized on a file with other `read` and `write` calls. Concurrent I/O is generally used by applications that require high performance for accessing data and do not perform overlapping writes to the same file.

Veritas Storage Foundation supports concurrent I/O for applications running in the non-global zones as well. This implies that a process in a non-global zone can access the file concurrently with other processes in the global or non-global zone. The application or running threads are responsible for coordinating the write activities to the same file when using Concurrent I/O.

An application must perform the following activities to enable the concurrent I/O advisory on a file:

```
fd=open(filename, oflag)
ioctl(fd, VX_SETCACHE, VX_CONCURRENT)
write(fd, buff, numofbytes)
```

Veritas extension for Oracle Disk Manager

The Veritas extension for Oracle Disk Manager (ODM) is specifically designed for Oracle9i or later to enhance file management and disk I/O throughput. The features of Oracle Disk Manager are best suited for databases that reside in a file system contained in Veritas File System. Oracle Disk Manager allows Oracle9i or later users to improve database throughput for I/O intensive workloads with special I/O optimization.

The Veritas extension for Oracle Disk Manager is supported in non-global zones. To run Oracle 10g on a non-global zone and use Oracle Disk Manager, the Oracle software version must be 10.1.0.3 or higher.

The following procedure enables Oracle Disk Manager file access from non-global zones with Veritas File System.

To enable Oracle Disk Manager file access from non-global zones with Veritas File System

1 Make global zone licenses visible to the non-global zone by exporting the `/etc/vx/licenses/lic` directory to the non-global zone as a lofs:

```
global# zonecfg -z myzone
zonecfg:myzone> add fs
zonecfg:myzone:fs> set dir=/etc/vx/licenses/lic
zonecfg:myzone:fs> set special=/etc/vx/licenses/lic
zonecfg:myzone:fs> set type=lofs
zonecfg:myzone:fs> end
zonecfg:myzone> commit
```

2 Create the `/dev/odm` directory in the non-global zone from the global zone:

```
global# mkdir -p myzone/dev/odm
```

3 Log in to the non-global zone and mount `/dev/odm`, or run `S92odm start`:

```
global# zlogin myzone
myzone# mount -F odm /dev/odm /dev/odm
```

Or:

```
global# /etc/rc2.d/S92odm start
```

`/dev/odm` is not automatically mounted after a zone is booted. Use the `S92odm start` command to mount `/dev/odm` after a zone is booted.

Exporting VxVM volumes to a non-global zone

A volume device node can be exported for use in non-global zone using the `zonecfg` command. The following procedure makes a volume `vol1` available in the non-global zone `myzone`.

Caution: Exporting raw volumes to non-global zones has implicit security risks. It is possible for the zone administrator to create malformed file systems that could later panic the system when a mount is attempted. Directly writing to raw volumes, exported to non-global zones, and using utilities such as `dd` can lead to data corruption in certain scenarios.

To export VxVM volumes to a non-global zone

1. Create a volume `vol1` in the global zone:

```
global# ls -l /dev/vx/rdsk/rootdg/vol1
crw-------   1 root     root     301, 102000 Jun  3
12:54 /dev/vx/rdsk/rootdg/vol1crw-------   1 root     sys      301, 10200
0 Jun  3 12:54 /devices/pseudo/vxio@0:rootdg,vol1,102000,raw
```

2. Add the volume device `vol1` to the non-global zone `myzone`:

```
global# zonecfg -z myzone
zonecfg:myzone> add device
zonecfg:myzone:device> set match=/dev/vx/rdsk/rootdg/vol1
zonecfg:myzone:device> end
zonecfg:myzone> commit
```

3. Ensure that the devices will be seen in the non-global zone:

```
global# zoneadm -z myzone halt
global# zoneadm -z myzone boot
```

4. Verify that `/myzone/dev/vx` contains the raw volume node and that the non-global zone can perform I/O to the raw volume node.

   The exported device can now be using for performing I/O or for creating file systems. Symantec recommends using VxFS file systems, due to the increased fault tolerance provided by VxFS.

Volume physical nodes in Solaris 10

On the Solaris 10 operating environment, there are two physical nodes corresponding to each volume node entry, `/devices` and `/dev`, respectively, with the same major and minor number. The physical nodes appear as follows:

```
/devices raw volume node : /devices/pseudo/vxio@0:
  dgname,volname,minor_number,raw
/devices block volume node : /devices/pseudo/vxio@0:
  dgname,volname,minor_number,blk
/dev raw volume node : /dev/vx/rdsk/dgname/volumename
/dev block volume node : /dev/vx/dsk/dgname/volumename
```

The following example provides sample values in `/devices`:

```
vm240v1:/-> ls -l /devices/pseudo/vxio*vol1*
brw-------   1 root     sys      302, 66000 Mar 25
17:21 /devices/pseudo/vxio@0:mydg,vol1,66000,blk
crw-------   1 root     sys      302, 66000 Mar 25
17:21 /devices/pseudo/vxio@0:mydg,vol1,66000,raw
```

The following example provides sample values in `/dev`:

```
vm240v1:/-> ls -l /dev/vx/*dsk/mydg/vol1
brw-------   1 root     root     302, 66000 Mar 25 17:21 /dev/vx/dsk/mydg/vol1
crw-------   1 root     root     302, 66000 Mar 25 17:21 /dev/vx/rdsk/mydg/vol1
```

Removing a VxVM volume from a non-global zone

The following procedure removes a VxVM volume from a non-global zone.

To remove a VxVM volume from a non-global zone

- Remove the volume device `vol3` from the non-global zone `myzone`:

```
global# zonecfg -z myzone
zonecfg:myzone> remove device match=/dev/vx/rdsk/pddg/vol3
zonecfg:myzone> end
zonecfg:myzone> commit
```

Software limitations of Storage Foundation support of non-global zones

This section describes the software limitations of Storage Foundation support of non-global zones in this release.

Solaris patches required for Veritas Volume Manager support in non-global zone

The *Veritas Storage Foundation Release Notes* contains a list of patches required for the proper operation of VxVM in non-global zones.

See the *Veritas Storage Foundation Release Notes*.

Cannot remove a volume configured in a non-global zone

After a volume node is configured in a Solaris non-global zone, the node cannot be removed in certain Solaris releases. Solaris patch `122660-10` resolves this issue.

Administration commands are not supported in non-global zone

All administrative tasks, such as resizing a volume, adding a volume to a volume set, and file system reorganization, are supported only in the global zone. Consequently, administrative commands, such as `fsadm`, `fsvoladm`, and `vxassist`, and administrative ioctls are not supported in the non-global zone by both VxFS and VxVM.

A cluster-mounted VxFS file system is not supported as the root of a non-global zone

The root of a non-global zone cannot currently be on a cluster-mounted VxFS file system.

QIO and CQIO are not supported

Quick I/O and Cached Quick I/O are not supported by VxFS in non-global zones.

Package and patch installation in non-global zones

To ensure that the package and patch updates applied to Veritas products in the global zone are also propagated to the non-global zone, ensure that the non-global zones are in a bootable state. For example, to update the software installed in a system, the file system housing the root directory of the non-global zone needs to be mounted and the disk group housing the non-global zone needs to be online at the time that you run `patchadd`, `pkgadd`, or the CPI installation scripts.

Root volume cannot be added to non-global zones

The root volume cannot be added to non-global zones.

Some Veritas Volume Manager operations can cause volume device names to go out of sync

If a volume is exported to a non-global zone, some Veritas Volume Manager operations can cause the global and non-global volume device names to go out of sync, which can create data corruption. This is because the Solaris operating environment zone support is not aware of the `devfsadm`(1M) command, and thus the zone configuration is not updated with any changes to the `/dev` or `/devices` namespaces.

The following operations can cause device names to go out of sync:

- Removing a volume

- Importing a disk group
- Deporting a disk group
- Renaming a disk group or volume
- Reminoring a disk group
- Restarting `vxconfigd` or resetting the kernel

To prevent device names from to going out of sync, if a volume is exported to a non-global zone and an operation that can cause device names to go out of sync occurs on that volume, remove the volume from the zone configuration using the `zonecfg` command and reboot the zone using the `zoneadm` command.

See the `zonecfg`(1M) and `zoneadm`(1M) manual pages.

Note: This issue applies to any Solaris device for which the `/dev` or `/devices` device node is changed and has been configured in the non-global zone before the change, including Solaris Volume Manager volumes.

Appendix C

Migrating arrays

This appendix includes the following topics:

- Migrating to thin provisioning

Migrating to thin provisioning

The SmartMove™ feature enables migration from a traditional LUN to a thinly provisioned LUN, removing unused space in the process.

To migrate to thin provisioning

1. Turn on the SmartMove feature. Edit the file `/etc/default/vxsf` so that the variable `usefssmartmove=all`.
2. Add the new, thin LUN to the existing disk group:

```
# vxdisksetup -i THINARRAY0_02

# vxdg -g datadg adddisk THINARRAY0_02
```

3. Add the new, thin LUN as a new plex to the volume.

   NOTE: The file system must be mounted to get the benefits of the SmartMove feature.

   The following methods are available to add the LUN:

   - Use the default settings:

     ```
     # vxassist -g datadg mirror datavol THINARRAY0_02
     ```

   - Use the options for fast completion. The following command has more I/O impact.

```
# vxassist -b -oiosize=1m -t thinmig -g datadg mirror \
  datavol THINARRAY0_02

# vxtask monitor thinmig
```

- Use the options for minimal impact. The following command takes longer to complete:

```
# vxassist -oslow -g datadg mirror datavol THINARRAY0_02
```

4 Optionally, test the performance of the new LUN before removing the old LUN.

To test the performance, use the following steps:

- Determine which plex corresponds to the thin LUN:

```
# vxprint -g datadg
```

```
TY NAME                ASSOC           KSTATE   LENGTH     PLOFFS  STATE     TUTIL0  PUTIL0
dg datadg              datadg          -        -          -       -         -       -
dm THINARRAY0_02       THINARRAY0_02   -        83886080   -       -         -       -
dm STDARRAY1_01        STDARRAY1_01    -        41943040   -       -OHOTUSE  -       -

v  datavol             fsgen           ENABLED  41943040   -       ACTIVE    -       -
pl datavol-01          datavol         ENABLED  41943040   -       ACTIVE    -       -
sd STDARRAY1_01-01     datavol-01      ENABLED  41943040   0       -         -       -
pl datavol-02          datavol         ENABLED  41943040   -       ACTIVE    -       -
sd THINARRAY0_02-01    datavol-02      ENABLED  41943040   0       -         -       -
```

The above output indicates that the thin LUN corresponds to plex datavol-02.

- Direct all reads to come from that LUN:

```
# vxvol -g datadg rdpol prefer datavol datavol-02
```

5 Remove the original LUN. Note that ! is an escape on some shells; the example given avoids bash escape.

```
# vxassist -g datadg remove mirror datavol \!STDARRAY1_01

# vxdg -g datadg rmdisk STDARRAY1_01

# vxdisk rm STDARRAY1_01
```

6 Grow the file system and volume to use all of the larger thin LUN:

```
# vxresize -g datadg -x datavol 40g THINARRAY0_02
```

Appendix D

Configuring Veritas Volume Manager

This appendix includes the following topics:

- Setup tasks after installation
- Unsupported disk arrays
- Foreign devices
- Initialization of disks and creation of disk groups
- Guidelines for configuring storage
- VxVM's view of multipathed devices
- Cluster support

Setup tasks after installation

A number of setup tasks can be performed after installing the Veritas Volume Manager (VxVM) software.

The following tasks are to perform initial setup:

- Create disk groups by placing disks under Veritas Volume Manager control.
- If you intend to use the Intelligent Storage Provisioning (ISP) feature, create storage pools within the disk groups.
- Create volumes in the disk groups.
- Configure file systems on the volumes.

The following setup tasks are optional:

- Encapsulate the `root` disk, and mirror it to create an alternate boot disk.
- Designate hot-relocation spare disks in each disk group.
- Add mirrors to volumes.
- Configure DRL and FastResync on volumes.

The following tasks are to perform ongoing maintenance:

- Resize volumes and file systems.
- Add more disks, create new disk groups, and create new volumes.
- Create and maintain snapshots.

Unsupported disk arrays

After installation, add any disk arrays that are unsupported by Symantec to the DISKS (JBOD) category.

See "How to administer the Device Discovery Layer" on page 93.

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.

See "How to administer the Device Discovery Layer" on page 93.

Initialization of disks and creation of disk groups

To place disks in disk groups, use VEA or the `vxdiskadm` program after completing the installation.

See "Adding a disk to VxVM" on page 113.

See the VEA online help for information on how to add your disks to new disk groups.

Storage pools are only required if you intend using the ISP feature of VxVM.

See the *Veritas Storage Foundation Intelligent Storage Provisioning Administrator's Guide*.

Guidelines for configuring storage

A disk failure can cause loss of data on the failed disk and loss of access to your system. Loss of access is due to the failure of a key disk used for system operations. Veritas Volume Manager can protect your system from these problems.

To maintain system availability, data important to running and booting your system must be mirrored. The data must be preserved so it can be used in case of failure.

The following are suggestions for protecting your system and data:

- Perform regular backups to protect your data. Backups are necessary if all copies of a volume are lost or corrupted. Power surges can damage several (or all) disks on your system. Also, typing a command in error can remove critical files or damage a file system directly. Performing regular backups ensures that lost or corrupted data is available to be retrieved.
- Place the disk containing the `root` file system (the root or boot disk) under Veritas Volume Manager control through encapsulation. Encapsulation converts the `root` and `swap` devices to volumes (`rootvol` and `swapvol`). Mirror the root disk so that an alternate root disk exists for booting purposes. By mirroring disks critical to booting, you ensure that no single disk failure leaves your system unbootable and unusable.
  See "Rootability" on page 128.
- Use mirroring to protect data against loss from a disk failure.
  See "Mirroring guidelines" on page 616.
- Use the DRL feature to speed up recovery of mirrored volumes after a system crash.
  See "Dirty region logging guidelines" on page 616.
- Use striping to improve the I/O performance of volumes.
  See "Striping guidelines" on page 617.
- 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 618.

- Leave the Veritas Volume Manager hot-relocation feature enabled.
  See "Hot-relocation guidelines" on page 618.

Mirroring guidelines

Refer to the following guidelines when using mirroring.

- Do not place subdisks from different plexes of a mirrored volume on the same physical disk. This action compromises the availability benefits of mirroring and degrades performance. Using the `vxassist` or `vxdiskadm` commands precludes this from happening.
- To provide optimum performance improvements through the use of mirroring, at least 70 percent of physical I/O operations should be read operations. A higher percentage of read operations results in even better performance. Mirroring may not provide a performance increase or may even result in a performance decrease in a write-intensive workload environment.
- The operating system implements a file system cache. Read requests can frequently be satisfied from the cache. This can cause the read/write ratio for physical I/O operations through the file system to be biased toward writing (when compared to the read/write ratio at the application level).
- Where possible, use disks attached to different controllers when mirroring or striping. Most disk controllers support overlapped seeks. This allows seeks to begin on two disks at once. Do not configure two plexes of the same volume on disks that are attached to a controller that does not support overlapped seeks. This is important for older controllers or SCSI disks that do not cache on the drive. It is less important for modern SCSI disks and controllers. Mirroring across controllers allows the system to survive a failure of one of the controllers. Another controller can continue to provide data from a mirror.
- A plex exhibits superior performance when striped or concatenated across multiple disks, or when located on a much faster device. Set the read policy to prefer the faster plex. By default, a volume with one striped plex is configured to prefer reading from the striped plex.

See "Mirroring (RAID-1)" on page 46.

Dirty region logging guidelines

Dirty region logging (DRL) can speed up recovery of mirrored volumes following a system crash. When DRL is enabled, Veritas Volume Manager keeps track of the regions within a volume that have changed as a result of writes to a plex.

Warning: Using Dirty Region Logging can adversely impact system performance in a write-intensive environment.

See "Dirty region logging" on page 62.

Striping guidelines

Refer to the following guidelines when using striping.

- Do not place more than one column of a striped plex on the same physical disk.
- Calculate stripe-unit sizes carefully. In general, a moderate stripe-unit size (for example, 64 kilobytes, which is also the default used by `vxassist`) is recommended.
- If it is not feasible to set the stripe-unit size to the track size, and you do not know the application I/O pattern, use the default stripe-unit size.
- Many modern disk drives have variable geometry. This means that the track size differs between cylinders, so that outer disk tracks have more sectors than inner tracks. It is therefore not always appropriate to use the track size as the stripe-unit size. For these drives, use a moderate stripe-unit size (such as 64 kilobytes), unless you know the I/O pattern of the application.
- Volumes with small stripe-unit sizes can exhibit poor sequential I/O latency if the disks do not have synchronized spindles. Generally, striping over disks without synchronized spindles yields better performance when used with larger stripe-unit sizes and multi-threaded, or largely asynchronous, random I/O streams.
- Typically, the greater the number of physical disks in the stripe, the greater the improvement in I/O performance; however, this reduces the effective mean time between failures of the volume. If this is an issue, combine striping with mirroring to combine high-performance with improved reliability.
- If only one plex of a mirrored volume is striped, set the policy of the volume to `prefer` for the striped plex. (The default read policy, `select`, does this automatically.)
- If more than one plex of a mirrored volume is striped, configure the same stripe-unit size for each striped plex.
- Where possible, distribute the subdisks of a striped volume across drives connected to different controllers and buses.
- Avoid the use of controllers that do not support overlapped seeks. (Such controllers are rare.)

The `vxassist` command automatically applies and enforces many of these rules when it allocates space for striped plexes in a volume.

See "Striping (RAID-0)" on page 42.

RAID-5 guidelines

Refer to the following guidelines when using RAID-5.

In general, the guidelines for mirroring and striping together also apply to RAID-5. The following guidelines should also be observed with RAID-5:

- Only one RAID-5 plex can exist per RAID-5 volume (but there can be multiple log plexes).
- The RAID-5 plex must be derived from at least three subdisks on three or more physical disks. If any log plexes exist, they must belong to disks other than those used for the RAID-5 plex.
- RAID-5 logs can be mirrored and striped.
- If the volume length is not explicitly specified, it is set to the length of any RAID-5 plex associated with the volume; otherwise, it is set to zero. If you specify the volume length, it must be a multiple of the stripe-unit size of the associated RAID-5 plex, if any.
- If the log length is not explicitly specified, it is set to the length of the smallest RAID-5 log plex that is associated, if any. If no RAID-5 log plexes are associated, it is set to zero.
- Sparse RAID-5 log plexes are not valid.
- RAID-5 volumes are not supported for sharing in a cluster.

See "RAID-5 (striping with parity)" on page 49.

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 "How hot-relocation works" on page 456.

Accessing volume devices

As soon as a volume has been created and initialized, it is available for use as a virtual disk partition by the operating system for the creation of a file system, or by application programs such as relational databases and other data management software.

Creating a volume in a disk group sets up block and character (raw) device files that can be used to access the volume:

`/dev/vx/dsk/`*`dg`*`/`*`vol`*	block device file for volume *vol* in disk group *dg*
`/dev/vx/rdsk/`*`dg`*`/`*`vol`*	character device file for volume *vol* in disk group *dg*

The pathnames include a directory named for the disk group. Use the appropriate device node to create, mount and repair file systems, and to lay out databases that require raw partitions.

VxVM's view of multipathed devices

You can use the `vxdiskadm` command to control how a device is treated by the Dynamic Multipathing (DMP) feature of VxVM.

See "Disabling multipathing and making devices invisible to VxVM" on page 170.

Cluster support

The Veritas Volume Manager software includes a licensable feature that enables it to be used in a cluster environment. The cluster functionality in Veritas Volume Manager allows multiple hosts to simultaneously access and manage a set of disks under Veritas Volume Manager control. A cluster is a set of hosts sharing a set of disks; each host is referred to as a node in the cluster.

See the *Veritas Storage Foundation Getting Started Guide*.

Configuring shared disk groups

If you are installing Veritas Volume Manager for the first time or adding disks to an existing cluster, you need to configure new shared disks.

Note: RAID-5 volumes are not supported for sharing in a cluster.

If you are setting up Veritas Volume Manager for the first time, configure the shared disks using the following steps in the specified order:

- Start the cluster on one node only to prevent access by other nodes.
- On one node, run the `vxdiskadm` program and choose option 1 to initialize new disks. When asked to add these disks to a disk group, choose `none` to leave the disks for future use.
- On other nodes in the cluster, run `vxdctl enable` to see the newly initialized disks.
- From the master node, create disk groups on the shared disks. To determine if a node is a master or slave, run the command `vxdctl -c mode`.
- Use the `vxdg` command or VEA to create disk groups. If you use the `vxdg` command, specify the `-s` option to create shared disk groups.
- From the master node only, use `vxassist` or VEA to create volumes in the disk groups.
- If the cluster is only running with one node, bring up the other cluster nodes. Enter the `vxdg list` command on each node to display the shared disk groups.

Converting existing VxVM disk groups to shared disk groups

To convert existing disk groups to shared disk groups

1. Start the cluster on one node only to prevent access by other nodes.
2. Configure the disk groups using the following procedure.

   To list all disk groups, use the following command:

   ```
   # vxdg list
   ```

   To deport the disk groups that are to be shared, use the following command:

   ```
   # vxdg deport diskgroup
   ```

   To import disk groups to be shared, use the following command:

   ```
   # vxdg -s import diskgroup
   ```

   This procedure marks the disks in the shared disk groups as shared and stamps them with the ID of the cluster, enabling other nodes to recognize the shared disks.

   If dirty region logs exist, ensure they are active. If not, replace them with larger ones.

   To display the shared flag for all the shared disk groups, use the following command:

   ```
   # vxdg list
   ```

   The disk groups are now ready to be shared.
3. Bring up the other cluster nodes. Enter the `vxdg list` command on each node to display the shared disk groups. This command displays the same list of shared disk groups displayed earlier.

   See the *Veritas Storage Foundation Cluster File System Installation Guide*.

Appendix E

Migrating from Solaris Volume Manager to VxVM

This appendix includes the following topics:

- About migration from Solaris Volume Manager
- How Solaris Volume Manager objects are mapped to VxVM objects
- Overview of the conversion process
- Planning the conversion
- Preparing a Solaris Volume Manager configuration for conversion
- Setting up a Solaris Volume Manager configuration for conversion
- Converting from the Solaris Volume Manager software to VxVM
- Post conversion tasks
- Converting a root disk

About migration from Solaris Volume Manager

Veritas Volume Manager (VxVM) provides utilities that you can use to convert objects, file systems, and device partitions controlled by the Sun Microsystems Solaris Volume Manager software (also known as Solstice Disk Suite™) software to VxVM control. The tools convert all Solaris Volume Manager objects into VxVM objects and encapsulate file systems and swap device partitions used by Solaris Volume Manager software. The conversion happens in place. You do not need to copy data into temporary storage. After the conversion, the Solaris Volume Manager software is disabled and can be removed.

The conversion utilities have the following characteristics:

- The utilities are provided in the `VRTSvxvm` package.
- All objects under Solaris Volume Manager control are converted. The conversion commands have no partial conversion option.
- You must use the recommended root disk conversion procedure if you want to move only a root disk from under Solaris Volume Manager control to VxVM. See "Converting a root disk" on page 639.

Warning: You cannot use the conversion utilities in a mixed environment. Even if you move only the root disk from Solaris Volume Manager software to VxVM, you cannot later use the conversion utilities to move other Solaris Volume Manager objects to VxVM.

- The conversion cannot be done in a mixed environment, which is a system that has both Solaris Volume Manager metadevices and VxVM objects (disk groups, un-used plexes, etc).
- All file systems that are on a Solaris Volume Manager metadevice and that are listed in `/etc/vfstab` are put under VxVM control.
- A conversion retains the on-disk layouts of Solaris Volume Manager volumes and does not improve layouts as supported by VxVM.
- The conversion utilities do not allow a conversion to be reversed. Once the conversion process starts, the best way to return to the Solaris Volume Manager configuration is to restore it from backups.
- After the conversion process starts on RAID-5 volumes, the data in these volumes is reorganized. Returning RAID-5 volumes to Solaris Volume Manager control can be done only by restoring the Solaris Volume Manager configuration and the volume data from backups.
- If you are very familiar with Solaris Volume Manager software and VxVM, you may be able to reverse a conversion without performing a full Solaris Volume Manager restoration from backups. However, RAID-5 volumes must always be restored from backups because the conversion utilities always reorganize RAID-5 parity. To manually reverse a conversion, preserve all Solaris Volume Manager configuration information that is required for its restoration. This includes output from `metastat -p` and `metadb` for every diskset, and `prtvtoc` output for all disks.
- Any partition that is on a disk with Solaris Volume Manager metadevices and that is not listed as a file system in `/etc/vfstab` is lost during the conversion. The partition is considered free space when the disk is converted to VxVM control. Even if the partitions are in use, such as for Oracle raw tablespace

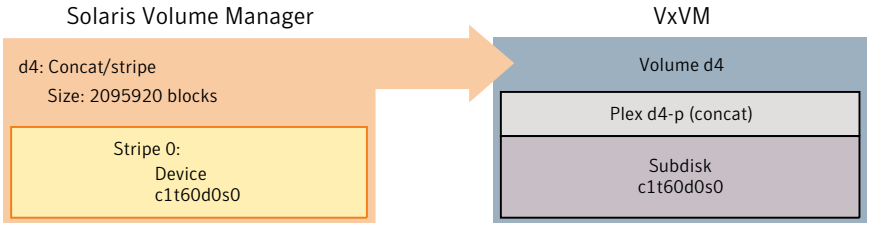
data, they are not encapsulated during the conversion. The raw partitions are lost when the disk layout changes. Either encapsulate the partition under Solaris Volume Manager control and allow the conversion to convert the resulting metadevice, or back up the partition, plan to create a new volume for it, and restore the partition after the conversion.

How Solaris Volume Manager objects are mapped to VxVM objects

The following illustrations give examples of how the conversion process maps Solaris Volume Manager concat/stripe objects to VxVM objects. RAID-5 volumes and mirrors come through the conversion intact.

Figure E-1 shows a single Solaris Volume Manager partition encapsulated in a concat/stripe Solaris Volume Manager object.

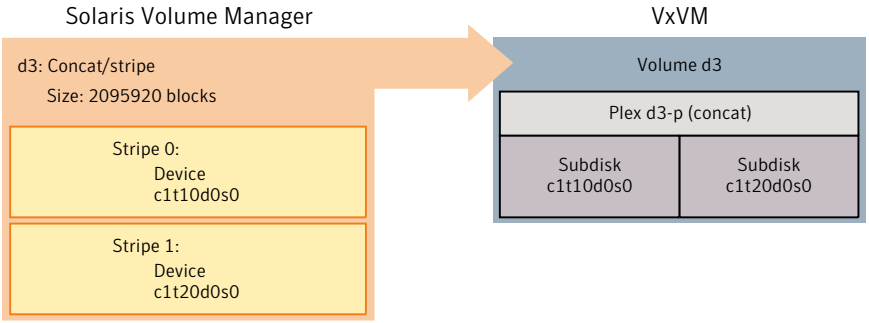
Figure E-1 Single Solaris Volume Manager partition encapsulated in a concat/stripe Solaris Volume Manager object



The partition becomes a simple volume under VxVM.

Figure E-2 shows a number of one-partition Solaris Volume Manager stripes in a concat/stripe Solaris Volume Manager object.

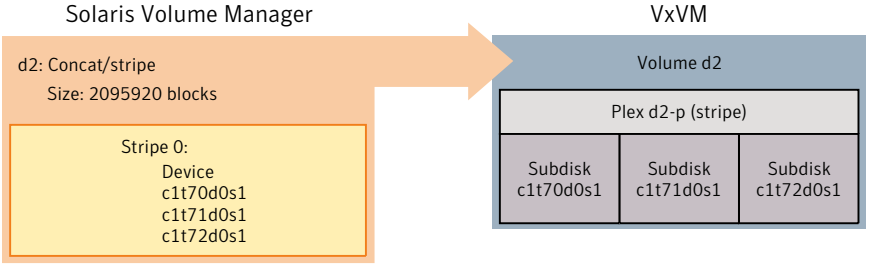
Figure E-2 A number of one-partition Solaris Volume Manager stripes in a concat/stripe Solaris Volume Manager object



The stripes become a concat plex under VxVM.

Figure E-3 shows a single Solaris Volume Manager stripe with an arbitrary number of partitions in a concat/stripe Solaris Volume Manager object.

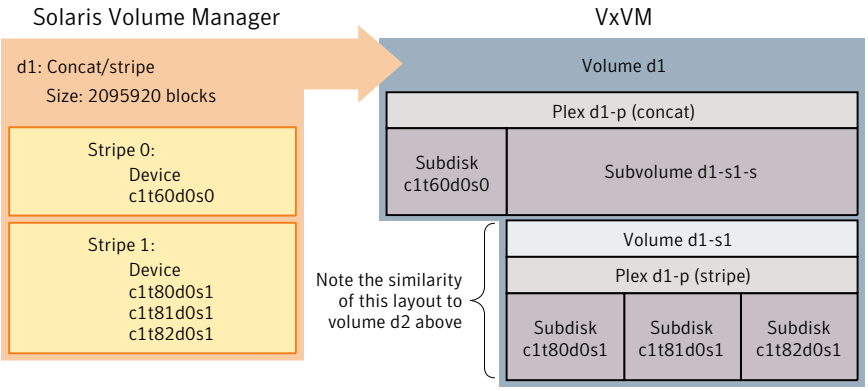
Figure E-3 A single Solaris Volume Manager stripe with any number of partitions in a concat/stripe Solaris Volume Manager object



The partitions become a striped plex under VxVM.

Figure E-4 shows a Solaris Volume Manager concat/stripe object with a mix of stripes.

Figure E-4 A Solaris Volume Manager concat/stripe object with a mix of stripes



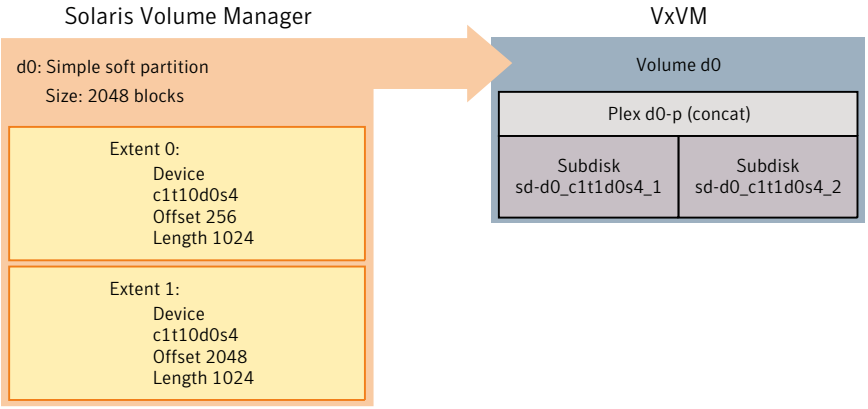
Under VxVM, the stripes are concatenated and may require a layered volume.

Conversion of soft partitions

The provided utilities support the conversion of the following soft partition-based layouts:

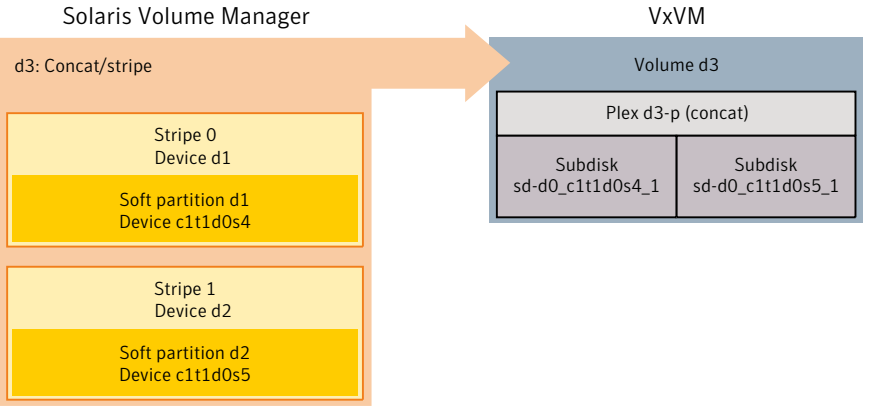
- A simple soft partition on a disk slice with one or more extents.
  Figure E-5 shows the conversion of such a layout.

Figure E-5 Simple soft partition on a disk slice with two extents



- Concat/Stripe on a soft partition on a disk slice with a single extent.
  Figure E-6 shows the conversion of such a layout.

Figure E-6 Concat/stripe Solaris Volume Manager object on a soft partition on a disk slice with a single extent



- RAID-5 on a soft partition on a disk slice with a single extent.
- Concat/Stripe on a soft partition on a disk slice with multiple extents.

The utilities do not support the conversion of the following soft partition-based layouts:

- RAID-5 on soft partitions on a disk with multiple extents.
- Soft partition on a logical volume.

More information about soft partitions may be found in the Solaris documentation.

See "Check metadevices" on page 632.

Overview of the conversion process

The conversion utilities allow you to do most of the preparation before starting the conversion, and before disrupting your system. To take advantage of this feature, read this entire manual and be familiar with the required tasks before starting the conversion from the Solaris Volume Manager configuration to VxVM.

The conversion utilities perform the following tasks:

- Planning and preparing for conversion.
  See "Plan and prepare for the conversion" on page 629.
- Set up the conversion.
  See "Set up the conversion" on page 629.
- Convert the volumes.
  See "Set up the conversion" on page 629.

- Perform post conversion cleanup tasks.
  See "Perform the conversion" on page 630.

Note: For a successful conversion, follow the steps in the given order.

Plan and prepare for the conversion

The tasks in this category can be done well before starting the actual conversion.

Plan the process and collect data for the conversion. The conversion may require changes to some Solaris Volume Manager metadevices.

See "Check metadevices" on page 632.

See "Planning the conversion" on page 630.

See "Preparing a Solaris Volume Manager configuration for conversion" on page 633.

Install the basic Veritas Volume Manager (without creating disk groups or objects and without initializing disks). The conversion utilities complete the VxVM setup with objects converted from Solaris Volume Manager software.

Set up the conversion

The tasks in this category should be done or updated just before starting the conversion. Information from the tasks in this category must be current when the conversion starts.

Use the `preconvert` utility to analyze the Solaris Volume Manager configuration and build a description for the VxVM conversion. You can run `preconvert` multiple times as it makes no changes to the Solaris Volume Manager configuration or file systems.

Run the `showconvert` utility to interpret `preconvert` output into a readable format. The `showconvert` output flags where changes must be made in the Solaris Volume Manager configuration for the conversion process to succeed. After making changes, run `preconvert` and `showconvert` again before starting the conversion.

Make backups of everything needed to restore the Solaris Volume Manager configuration.

See "Setting up a Solaris Volume Manager configuration for conversion" on page 634.

Perform the conversion

The first task in this category starts the conversion from Solaris Volume Manager software to VxVM. You have no easy way to return to a Solaris Volume Manager configuration once the conversion starts.

Run the `doconvert` utility to make the conversion. `Doconvert` relies on information from the `preconvert` utility.

Reboot the system. The reboot activates system configuration changes.

Change volume references in applications that use the volumes. VxVM uses volume names that are different than the names used by Solaris Volume Manager software. Every application that uses the Solaris Volume Manager volumes must be updated to use the new names.

See "Converting from the Solaris Volume Manager software to VxVM" on page 637.

Perform post-conversion tasks

After the `doconvert` utility performs a reboot of the system, the converted volumes and file systems are under the control of Veritas Volume Manger.

Use VxVM tools such as `vxassist` or the graphical user interface to improve the layout of volumes.

Remove or re-initialize the Solaris Volume Manager software for use on non-VxVM disks. After the conversion, the Solaris Volume Manager software is disabled and does not control any data. You can either remove the Solaris Volume Manager software or initialize the Solaris Volume Manager software with disks that are not used by VxVM. Both VxVM and the Solaris Volume Manager software can run on the same system.

See "Post conversion tasks" on page 638.

Planning the conversion

Before using any of the conversion utilities, you should consider how to schedule the steps in the process.

Scheduling considerations

The following preparation tasks can be done at any time before the conversion:

- Schedule downtime.
- Check and reconfigure Solaris Volume Manager metadevices.

- Identify references from applications to Solaris Volume Manager volumes.
- Install VxVM.

The setup tasks should be done just before starting the conversion. If you make changes to the Solaris Volume Manager configuration after doing the setup tasks, perform the following setup tasks again to update information used during the conversion:

- Run the `preconvert` utility.
- Run the `showconvert` utility.
- Make Solaris Volume Manager configuration changes if `showconvert` shows error messages.
- Re-run `preconvert` and `showconvert` if needed.
- Make Solaris Volume Manager backups.

The conversion tasks mark the point where you first make changes that cannot be reversed easily. Previous tasks of gathering information and reconfiguring the Solaris Volume Manager software (if needed) all allow you to continue using the Solaris Volume Manager software with no disruption. Do the following conversion tasks as soon as possible one after the other:

- Run the `doconvert` utility.
- Reboot the system.
- Change volume references in applications.

After the conversion completes successfully, the following post-conversion tasks are optional and can be done at any time:

- Improve volume layouts.
- Remove or reinitialize the Solaris Volume Manager software.

Schedule downtime

A conversion includes downtime for applications using Solaris Volume Manager volumes during the `doconvert`, reboot, and change volume references tasks and downtime for the system during the reboot. A second reboot happens automatically if a root device hosts Solaris Volume Manager objects or is a Solaris Volume Manager metadevice.

The doconvert utility takes about a minute unless the conversion includes RAID-5 volumes. RAID-5 conversions take about a minute for each megabyte of RAID-5 data. The total downtime to expect for a conversion is the sum of the following times:

- Approximately one minute to run `doconvert`.
- The time needed for one or two system reboots.
- Approximately one minute for each megabyte of RAID-5 data if RAID-5 volumes are included.
- The time needed for changing volume references in applications.

Check metadevices

Solaris Volume Manager metadevices may need changes before starting the conversion.

- Extended RAID-5 metadevices can not be converted. A RAID-5 metadevice becomes extended when you add space after the device is first initialized. To find extended RAID-5 metadevices, use the `metastat -p` command of the Solaris Volume Manager software for each diskset. Look for RAID-5 devices in the output that have a `-o` option showing more original partitions than current partitions. The following example of output from `metastat -p` shows an extended metadevice:

  ```
  d100 -r clt65d0s3 clt66d0s3 clt67d0s3 clt99d0s3 -k -i 32b -0 3
  ```

  The example lists four partitions: 65, 66, 67, and 99. The `-o` entry shows the number of original partitions, three in this case.

  Use either of the following options to deal with extended RAID-5 devices:

  - Use the Solaris Volume Manager commands or the Solaris Volume Manager user interface to create a new metadevice large enough to contain all the data. Copy the data to the new metadevice.
  - Dump the data from the metadevice to tape and restore it to a VxVM volume after the conversion process finishes.

- Metadevices using a hot spare can not be converted. Use the `metastat` command of the Solaris Volume Manager software to identify metadevices that currently use hot spares. The following example of output from `metastat` shows a metadevice using a hot spare:

  ```
  d3:Submirror of d51
     State: Okay
     Hot spare pool: hsp000
     Size 5922252 blocks
     Stripe 0:
     Device       Start Block     Dbase     State     Hot Spare
     clt75d0s1          0          No        Okay      clt66d0s6
  ```

With no hot spare in use, the “Hot Spare” column at the far right is empty.

Use either of the following options to deal with hot spares:

- Use the `metareplace` command of the Solaris Volume Manager software to move data from the hot spare onto the original partition (if there is room).
- Use the `metareplace` and `metahs` commands of the Solaris Volume Manager software to move the hot spare partition from the hot spare pool into the metadevice.

Identify references by applications

Find and list all references to Solaris Volume Manager metadevices from applications. The conversion utilities update `/etc/vfstab` with a new path for each metadevice that is converted to a VxVM volume. However, you must manually change references to Solaris Volume Manager metadevices in every application that uses a Solaris Volume Manager metadevice.

See “Run convertname” on page 635.

Preparing a Solaris Volume Manager configuration for conversion

The following preparation tasks can be done at any time before starting the conversion.

Installing VxVM

You must install Veritas Volume Manager before running the conversion utilities. Obtain the VxVM software and a full VxVM license key.

During the VxVM installation, do not run `vxinstall`. If required, you can use the `vxlicinst` command to install any license keys.

After installation, do not initialize any disks, and do not create any disk groups or VxVM objects.

The conversion process completes the VxVM installation and configuration.

Setting up a Solaris Volume Manager configuration for conversion

Do the setup tasks just before converting from the Solaris Volume Manager software to VxVM. If you make any changes between the setup tasks and the actual conversion, update the setup tasks.

Run preconvert

The `preconvert` utility analyzes the current Solaris Volume Manager configuration and builds a description for the new VxVM configuration. `Preconvert` does not make any changes to the Solaris Volume Manager configuration or to the host system. The description created by `preconvert` must be current with the state of the Solaris Volume Manager software when the conversion process starts or the process will fail. Always run `preconvert` again if you make any changes to the Solaris Volume Manager configuration.

Preconvert has no options and uses no arguments:

```
# preconvert
```

Messages from `preconvert` appear on the screen and in a file that is used by the `showconvert` utility.

`WARNING` errors are for your information only and require no changes to the Solaris Volume Manager configuration.

`FATAL` errors mean that the conversion process will fail unless you make changes to the Solaris Volume Manager configuration. A `FATAL` error does not stop `preconvert` from examining the rest of the Solaris Volume Manager configuration. `FATAL` errors that do not make sense may be related to earlier `FATAL` errors. Correct the earlier `FATAL` errors and run `preconvert` again to see if the later errors still exist.

The following are possible causes of FATAL errors:

- A metadevice is being initialized or resynchronized.
- A disk is offline.
- `The /etc/vfstab` file contains incorrect entries.
- An extended RAID-5 metadevice exists.
- A metadevice is using a hot spare.

Run showconvert

The `showconvert` utility displays the `preconvert` conversion plan in a readable format. The command `showconvert help` gives a list of command options.

```
# showconvert keyword [arg ...]
```

The following keywords and arguments are defined for the `showconvert` command:

`log`	Displays error messages generated by the `preconvert` utility. All `FATAL` messages must be resolved and `preconvert` run again before the conversion process can run successfully.
`list [diskgroup ...]`	Lists all the VxVM disk groups that will be created from Solaris Volume Manager disk sets and gives information about each disk group. These VxVM disk groups are not created to be compatible with CDS. Additionally, the disks in these disk groups are set up in `slice` and not `cds` format. To make the disk groups compatible with CDS, use the `vxcdsconvert` command. See the `vxcdsconvert`(1M) manual page.
`print diskgroup [vxprint_args]`	Uses standard arguments from the VxVM `vxprint` utility and displays the VxVM objects that the conversion will create.

The conversion plan displayed by `showconvert` describes the layouts that the conversion utilities create. The layouts will have the same data on the same disks as the Solaris Volume Manager configuration. If you want different layouts, make changes to the Solaris Volume Manager configuration and run `preconvert` and `showconvert` again.

Run convertname

The `convertname` utility takes Solaris Volume Manager device paths as arguments (metadevice paths or raw disk paths) and returns the VxVM volume path for the device as it will show after the conversion. Use the command to make a list of path names for replacing references in applications. One or more device paths as arguments returns one line for each argument with the proposed VxVM path. Use absolute path names. For example:

```
# convertname /dev/md/dsk/d2 /dev/md/rdsk/d3
```

Table E-1 shows examples of how names are translated by the conversion process:

Table E-1 Examples of the translation of device path names

Metadevice path	VxVM volume path
`/dev/md/dsk/d2`	`/dev/vx/dsk/rootdg/d2`
`/dev/md/rdsk/d3`	`/dev/vx/rdsk/rootdg/d3`
`/dev/md/setname/dsk/d10`	`/dev/vx/dsk/setname/d10`
`/dev/md/setname/rdsk/d12`	`/dev/vx/rdsk/setname/d12`

The following metadevices do not retain their dN names or are converted from raw partitions that did not have dN names:

- The volume for the root file system is always named `rootvol` as a VxVM convention.
- A trans metadevice has its data device converted using the master device name. The log device is lost.
- A file system encapsulated from a raw partition is named for the last component of its mount point.
- A swap device on a disk that will include VxVM volumes after the conversion receives a name starting with `swapvol`.
- The `/usr`, `/opt`, and `/var` file systems when converted from being Solaris Volume Manager metadevices or when encapsulated retain the names `/usr`, `/opt`, and `/var`.

Warning: Any partition that is on a disk with Solaris Volume Manager metadevices and that is not listed as a file system in `/etc/vfstab` is lost during the conversion. The partition is considered free space when the disk is converted to VxVM control. Even if the partitions are in use, such as for Oracle raw tablespace data, they are not encapsulated during the conversion. The raw partitions are lost when the disk layout changes. Either encapsulate the partition under the Solaris Volume Manager software and allow the conversion to convert the resulting metadevice, or back up the partition, plan to create a new volume for it, and restore the partition after the conversion.

Make backups

Make backups of all Solaris Volume Manager software-related files and information before running the conversion utility. If you need to return to the Solaris Volume Manager configuration, using backups is the recommended way.

Converting from the Solaris Volume Manager software to VxVM

Warning: This is the first step that makes changes, and that is not easily reversed. Be sure that you are ready for the conversion before performing this step. This step also includes downtime for volume management and leads to a reboot for the system.

Run the `doconvert` utility to start the conversion from the Solaris Volume Manager software to VxVM. A manual reboot of the system completes the conversion by updating configuration changes that were made by `doconvert`.

```
# doconvert [force]
```

The `doconvert` utility checks that the VxVM licenses are correct, makes configuration changes, and then prompts for a system reboot. You must reboot the system immediately to complete the conversion process. If you run doconvert with the `force` option, the system is automatically rebooted without prompting you.

Warning: Do not make system configuration changes after running `doconvert` and before rebooting. Such changes can lead to loss of data.

If you choose not to reboot, the location of a script is given that allows you to reverse the changes to the system files made by `doconvert`. If you run this script, the system files are reinstated and the conversion is prevented from completing at the next reboot.

If doconvert fails, the system remains unchanged. You can then return to the preparation steps to correct problems. The following situations cause doconvert to fail:

- The output of `preconvert` does not exist or is stale.
- `Preconvert` terminated with an error.
- VxVM licenses are not sufficient for the planned VxVM configuration.

- A VxVM configuration already exists.

During the conversion, `doconvert` displays locations for some system file backups.

Reboot the system

Reboot the system immediately after doconvert finishes. During the reboot:

- If the system root device hosts Solaris Volume Manager objects or is a Solaris Volume Manager metadevice, the root device becomes a VxVM volume during the reboot. The process of making the root device a VxVM volume automatically starts a second reboot.
- If the system goes down, from a power loss or other problem, the conversion process continues when the system boots again. Messages such as “Already exists...” appear for processes that completed before the system went down. The messages are harmless.

Change volume references

After the reboot, you must manually change references in applications that use Solaris Volume Manager metadevices. The references need to point to the new VxVM volumes.

See “Identify references by applications” on page 633.

See “Run convertname” on page 635.

Post conversion tasks

Do the following tasks at any time after a successful conversion.

Improve volume layouts

Start VxVM and use tools such as `vxassist` or the VxVM user interface to relayout volumes online. New layouts may require additional disk space.

You may wish to create logs for converted RAID-5 volumes. RAID-5 logging from the Solaris Volume Manager configuration is not suitable for conversion to use by VxVM. Logs are not necessary for normal operations, but are useful for recovery. Use the following command to create a log for a RAID-5 volume:

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

Remove the Solaris Volume Manager software

After the conversion, the Solaris Volume Manager software controls no data and is disabled. You can safely remove the Solaris Volume Manager software or you can re-initialize the Solaris Volume Manager software for use on disks not controlled by VxVM. The Solaris Volume Manager software and VxVM can run at the same time on the same machine.

Converting a root disk

The following steps allow you to move only a root disk from Solaris Volume Manager control to VxVM control. Use this procedure if you want to move only the root disk from the Solaris Volume Manager software to VxVM.

Warning: You cannot use the conversion utilities in a mixed environment. Even if you move only the root disk from the Solaris Volume Manager software to VxVM, you cannot later use the conversion utilities to move other Solaris Volume Manager objects to Veritas Volume Manager.

To convert a root disk

1. Use the Solaris Volume Manager unencapsulation procedure to remove the root disk from Solaris Volume Manager control. See the Solaris Volume Manager documentation for details.
2. Create VxVM volumes from the partitions on the root disk by encapsulating the root disk.

   See "Rootability" on page 128.

Glossary

Active/Active disk arrays	This type of multipathed disk array allows you to access a disk in the disk array through all the paths to the disk simultaneously, without any performance degradation.
Active/Passive disk arrays	This type of multipathed disk array allows one path to a disk to be designated as primary and used to access the disk at any time. Using a path other than the designated active path results in severe performance degradation in some disk arrays.
associate	The process of establishing a relationship between VxVM objects; for example, a subdisk that has been created and defined as having a starting point within a plex is referred to as being associated with that plex.
associated plex	A plex associated with a volume.
associated subdisk	A subdisk associated with a plex.
atomic operation	An operation that either succeeds completely or fails and leaves everything as it was before the operation was started. If the operation succeeds, all aspects of the operation take effect at once and the intermediate states of change are invisible. If any aspect of the operation fails, then the operation aborts without leaving partial changes. In a cluster, an atomic operation takes place either on all nodes or not at all.
attached	A state in which a VxVM object is both associated with another object and enabled for use.
block	The minimum unit of data transfer to or from a disk or array.
boot disk	A disk that is used for the purpose of booting a system.
boot disk group	A private disk group that contains the disks from which the system may be booted.
bootdg	A reserved disk group name that is an alias for the name of the boot disk group.
clean node shutdown	The ability of a node to leave a cluster gracefully when all access to shared volumes has ceased.
cluster	A set of hosts (each termed a node) that share a set of disks.
cluster manager	An externally-provided daemon that runs on each node in a cluster. The cluster managers on each node communicate with each other and inform VxVM of changes in cluster membership.

cluster-shareable disk group A disk group in which access to the disks is shared by multiple hosts (also referred to as a shared disk group).

column A set of one or more subdisks within a striped plex. Striping is achieved by allocating data alternately and evenly across the columns within a plex.

concatenation A layout style characterized by subdisks that are arranged sequentially and contiguously.

configuration copy A single copy of a configuration database.

configuration database A set of records containing detailed information on existing VxVM objects (such as disk and volume attributes).

DCO (data change object) A VxVM object that is used to manage information about the FastResync maps in the DCO volume. Both a DCO object and a DCO volume must be associated with a volume to implement Persistent FastResync on that volume.

data stripe This represents the usable data portion of a stripe and is equal to the stripe minus the parity region.

DCO volume A special volume that is used to hold Persistent FastResync change maps and dirty region logs. See also see dirty region logging.

detached A state in which a VxVM object is associated with another object, but not enabled for use.

device name The device name or address used to access a physical disk, such as `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.

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.

Fibre Channel A collective name for the fiber optic technology that is commonly used to set up a Storage Area Network (SAN).

file system A collection of files organized together into a structure. The UNIX file system is a hierarchical structure consisting of directories and files.

free space An area of a disk under VxVM control that is not allocated to any subdisk or reserved for use by any other VxVM object.

free subdisk A subdisk that is not associated with any plex and has an empty `putil[0]` field.

hostid A string that identifies a host to VxVM. The host ID for a host is stored in its `volboot` file, and is used in defining ownership of disks and disk groups.

hot-relocation A technique of automatically restoring redundancy and access to mirrored and RAID-5 volumes when a disk fails. This is done by relocating the affected subdisks to disks designated as spares and/or free space in the same disk group.

hot-swap Refers to devices that can be removed from, or inserted into, a system without first turning off the power supply to the system.

initiating node The node on which the system administrator is running a utility that requests a change to VxVM objects. This node initiates a volume reconfiguration.

JBOD (just a bunch of disks) The common name for an unintelligent disk array which may, or may not, support the hot-swapping of disks.

log plex A plex used to store a RAID-5 log. The term log plex may also be used to refer to a Dirty Region Logging plex.

log subdisk A subdisk that is used to store a dirty region log.

master node A node that is designated by the software to coordinate certain VxVM operations in a cluster. Any node is capable of being the master node.

mastering node The node to which a disk is attached. This is also known as a disk owner.

mirror A duplicate copy of a volume and the data therein (in the form of an ordered collection of subdisks). Each mirror consists of one plex of the volume with which the mirror is associated.

mirroring A layout technique that mirrors the contents of a volume onto multiple plexes. Each plex duplicates the data stored on the volume, but the plexes themselves may have different layouts.

multipathing Where there are multiple physical access paths to a disk connected to a system, the disk is called multipathed. Any software residing on the host, (for example, the DMP driver) that hides this fact from the user is said to provide multipathing functionality.

node One of the hosts in a cluster.

node abort A situation where a node leaves a cluster (on an emergency basis) without attempting to stop ongoing operations.

node join The process through which a node joins a cluster and gains access to shared disks.

Non-Persistent FastResync A form of FastResync that cannot preserve its maps across reboots of the system because it stores its change map in memory.

object An entity that is defined to and recognized internally by VxVM. The VxVM objects are: volume, plex, subdisk, disk, and disk group. There are actually two types of disk objects—one for the physical aspect of the disk and the other for the logical aspect.

parity A calculated value that can be used to reconstruct data after a failure. While data is being written to a RAID-5 volume, parity is also calculated by performing an exclusive OR (XOR) procedure on data. The resulting parity is then written to the volume. If a portion of a RAID-5 volume fails, the data that was on that portion of the failed volume can be recreated from the remaining data and the parity.

parity stripe unit A RAID-5 volume storage region that contains parity information. The data contained in the parity stripe unit can be used to help reconstruct regions of a RAID-5 volume that are missing because of I/O or disk failures.

partition The standard division of a physical disk device, as supported directly by the operating system and disk drives.

path When a disk is connected to a host, the path to the disk consists of the HBA (Host Bus Adapter) on the host, the SCSI or fibre cable connector and the controller on the disk or disk array. These components constitute a path to a disk. A failure on

	any of these results in DMP trying to shift all I/O for that disk onto the remaining (alternate) paths.
pathgroup	In the case of disks which are not multipathed by `vxdmp`, VxVM will see each path as a disk. In such cases, all paths to the disk can be grouped. This way only one of the paths from the group is made visible to VxVM.
Persistent FastResync	A form of FastResync that can preserve its maps across reboots of the system by storing its change map in a DCO volume on disk).
persistent state logging	A logging type that ensures that only active mirrors are used for recovery purposes and prevents failed mirrors from being selected for recovery. This is also known as kernel logging.
physical disk	The underlying storage device, which may or may not be under VxVM control.
plex	A plex is a logical grouping of subdisks that creates an area of disk space independent of physical disk size or other restrictions. Mirroring is set up by creating multiple data plexes for a single volume. Each data plex in a mirrored volume contains an identical copy of the volume data. Plexes may also be created to represent concatenated, striped and RAID-5 volume layouts, and to store volume logs.
primary path	In Active/Passive disk arrays, a disk can be bound to one particular controller on the disk array or owned by a controller. The disk can then be accessed using the path through this particular controller.
private disk group	A disk group in which the disks are accessed by only one specific host in a cluster.
private region	A region of a physical disk used to store private, structured VxVM information. The private region contains a disk header, a table of contents, and a configuration database. The table of contents maps the contents of the disk. The disk header contains a disk ID. All data in the private region is duplicated for extra reliability.
public region	A region of a physical disk managed by VxVM that contains available space and is used for allocating subdisks.
RAID (redundant array of independent disks)	A disk array set up with part of the combined storage capacity used for storing duplicate information about the data stored in that array. This makes it possible to regenerate the data if a disk failure occurs.
read-writeback mode	A recovery mode in which each read operation recovers plex consistency for the region covered by the read. Plex consistency is recovered by reading data from blocks of one plex and writing the data to all other writable plexes.
root configuration	The configuration database for the root disk group. This is special in that it always contains records for other disk groups, which are used for backup purposes only. It also contains disk records that define all disk devices on the system.
root disk	The disk containing the root file system. This disk may be under VxVM control.

root file system The initial file system mounted as part of the UNIX kernel startup sequence.

root partition The disk region on which the root file system resides.

root volume The VxVM volume that contains the root file system, if such a volume is designated by the system configuration.

rootability The ability to place the `root` file system and the `swap` device under VxVM control. The resulting volumes can then be mirrored to provide redundancy and allow recovery in the event of disk failure.

secondary path In Active/Passive disk arrays, the paths to a disk other than the primary path are called secondary paths. A disk is supposed to be accessed only through the primary path until it fails, after which ownership of the disk is transferred to one of the secondary paths.

sector A unit of size, which can vary between systems. Sector size is set per device (hard drive, CD-ROM, and so on). Although all devices within a system are usually configured to the same sector size for interoperability, this is not always the case.

A sector is commonly 512 bytes.

shared disk group A disk group in which access to the disks is shared by multiple hosts (also referred to as a cluster-shareable disk group).

shared volume A volume that belongs to a shared disk group and is open on more than one node of a cluster at the same time.

shared VM disk A VM disk that belongs to a shared disk group in a cluster.

slave node A node that is not designated as the master node of a cluster.

slice The standard division of a logical disk device. The terms partition and slice are sometimes used synonymously.

snapshot A point-in-time copy of a volume (volume snapshot) or a file system (file system snapshot).

spanning A layout technique that permits a volume (and its file system or database) that is too large to fit on a single disk to be configured across multiple physical disks.

sparse plex A plex that is not as long as the volume or that has holes (regions of the plex that do not have a backing subdisk).

SAN (storage area network) A networking paradigm that provides easily reconfigurable connectivity between any subset of computers, disk storage and interconnecting hardware such as switches, hubs and bridges.

stripe A set of stripe units that occupy the same positions across a series of columns.

stripe size The sum of the stripe unit sizes comprising a single stripe across all columns being striped.

stripe unit Equally-sized areas that are allocated alternately on the subdisks (within columns) of each striped plex. In an array, this is a set of logically contiguous blocks that exist on each disk before allocations are made from the next disk in the array. A stripe unit may also be referred to as a stripe element.

stripe unit size The size of each stripe unit. The default stripe unit size is 64KB. The stripe unit size is sometimes also referred to as the stripe width.

striping A layout technique that spreads data across several physical disks using stripes. The data is allocated alternately to the stripes within the subdisks of each plex.

subdisk A consecutive set of contiguous disk blocks that form a logical disk segment. Subdisks can be associated with plexes to form volumes.

swap area A disk region used to hold copies of memory pages swapped out by the system pager process.

swap volume A VxVM volume that is configured for use as a swap area.

transaction A set of configuration changes that succeed or fail as a group, rather than individually. Transactions are used internally to maintain consistent configurations.

VM disk A disk that is both under VxVM control and assigned to a disk group. VM disks are sometimes referred to as VxVM disks.

volboot file A small file that is used to locate copies of the boot disk group configuration. The file may list disks that contain configuration copies in standard locations, and can also contain direct pointers to configuration copy locations. The `volboot` file is stored in a system-dependent location.

volume A virtual disk, representing an addressable range of disk blocks used by applications such as file systems or databases. A volume is a collection of from one to 32 plexes.

volume configuration device The volume configuration device (`/dev/vx/config`) is the interface through which all configuration changes to the volume device driver are performed.

volume device driver The driver that forms the virtual disk drive between the application and the physical device driver level. The volume device driver is accessed through a virtual disk device node whose character device nodes appear in `/dev/vx/rdsk`, and whose block device nodes appear in `/dev/vx/dsk`.

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

Symbols

/dev/vx/dmp directory 165
/dev/vx/rdmp directory 165
/etc/default/vxassist file 302, 467
/etc/default/vxdg file 222, 481
/etc/default/vxdisk file 87, 113
/etc/default/vxencap file 113
/etc/default/vxse file 533
/etc/init.d/vxvm-recover file 473
/etc/vfstab file 359
/etc/volboot file 269
/etc/vx/darecs file 269
/etc/vx/dmppolicy.info file 197
/etc/vx/volboot file 239
/kernel/drv/vxio.conf file 554–555
/kernel/drv/vxspec.conf file 556
/lib/svc/method/vxvm-recover file 473

A

A/A disk arrays 164
A/A-A disk arrays 164
A/P disk arrays 164
A/P-C disk arrays 165
A/PF disk arrays 164
A/PF-C disk arrays 165
A/PG disk arrays 164
A/PG-C disk arrays 165
A5x00 arrays
    removing and replacing disks 151
access port 164
activation modes for shared disk groups 480–481
ACTIVE
    plex state 284
    volume state 330
active path attribute 193
active paths
    devices 195–196
ACTIVE state 381
Active/Active disk arrays 164
Active/Passive disk arrays 164
adaptive load-balancing 197
adding disks 122
aliased disks 136
alignment constraints 304
allocation
    site-based 512
allsites attribute 521
APM
    configuring 212
application volumes 37
array policy module (APM)
    configuring 212
array ports
    disabling for DMP 203
    displaying information about 183
    enabling for DMP 204
array support library (ASL) 90
Array Volume ID
    device naming 107
arrays
    DMP support 89
ASL
    array support library 89–90
Asymmetric Active/Active disk arrays 164
ATTACHING state 381
attributes
    active 193
    autogrow 392, 395
    autogrowby 392
    cache 395
    cachesize 395
    comment 279, 292
    dcolen 71, 313, 430
    default for disk initialization 113
    default for encapsulation 113
    dgalign_checking 305
    displaying for rules 531
    drl 315, 349
    fastresync 313, 315, 362
    for specifying storage 305
    for Storage Expert 531
    hasdcolog 363
    highwatermark 392

attributes *(continued)*
init 324
len 279
listing for rules 531
loglen 316
logtype 315
maxautogrow 392
maxdev 243
mirdg 403
mirvol 403
name 278, 292
ncachemirror 395
ndcomirror 313, 315, 430
ndcomirs 342, 390
newvol 401
nmirror 401
nomanual 193
nopreferred 194
plex 292
preferred priority 194
primary 194
putil 278, 292
secondary 194
sequential DRL 315
setting for paths 193, 196
setting for rules 532
snapvol 397, 403
source 397, 403
standby 194
subdisk 278
syncing 388, 416
tutil 278, 292
auto disk type 86
autogrow
tuning 418
autogrow attribute 392, 395
autogrowby attribute 392
autotrespass mode 164

B

backups
created using snapshots 387
creating for volumes 374
creating using instant snapshots 387
creating using third-mirror snapshots 420
for multiple volumes 404, 425
implementing online 445
of disk group configuration 269
balanced path policy 198
base minor number 241
blocks on disks 35
boot command
-a flag 134
syntax 134
boot disk
encapsulating 131
listing volumes on 136
mirroring 131
unencapsulating 137
boot disk group 218
boot disks
alternate 132
listing aliases 136
using aliases 135
boot failure
cannot open altboot_disk 135
boot-time restrictions 130
bootdg 218
booting root volumes 129–130
booting system
aliased disks 136
break-off snapshots
emulation of 379
BROKEN state 381

C

c# 26, 83
c#t#d#s# 83
c#t#d#s# based naming 83
c0d0t0 84
cache attribute 395
cache objects
creating 391
enabling 392
listing snapshots in 418
caches
creating 391
deleting 420
finding out snapshots configured on 420
growing 419
listing snapshots in 418
removing 420
resizing 419
shrinking 419
stopping 420
used by space-optimized instant snapshots 378
cachesize attribute 395

Campus Cluster feature
administering 512
campus clusters
administering 512
serial split brain condition in 244
cascade instant snapshots 381
cascaded snapshot hierarchies
creating 408
categories
disks 90
CDS
alignment constraints 304
compatible disk groups 222
disk format 86
cds attribute 222
cdsdisk format 86
check_all policy 210
check_alternate policy 210
check_disabled policy 211
check_periodic policy 211
checkpoint interval 557
CLEAN
plex state 284
volume state 330
clone_disk flag 228
cloned disks 227–228
cluster functionality
enabling 620
shared disks 620
cluster protocol version
checking 509
upgrading 509
cluster-shareable disk groups in clusters 479
clusters
activating disk groups 481
activating shared disk groups 506
activation modes for shared disk groups 480
benefits 475
checking cluster protocol version 508
cluster-shareable disk groups 479
configuration 488
configuring exclusive open of volume by node 507–508
connectivity policies 482
converting shared disk groups to private 505
creating shared disk groups 503
designating shareable disk groups 479
detach policies 482
determining if disks are shared 501
clusters *(continued)*
forcibly adding disks to disk groups 504
forcibly importing disk groups 504
importing disk groups as shared 504
initialization 488
introduced 477
limitations of shared disk groups 487
listing shared disk groups 502
maximum number of nodes in 476
moving objects between disk groups 505
node abort 496
node shutdown 495
nodes 477
operation of DRL in 496–497
operation of vxconfigd in 493
operation of VxVM in 477
private disk groups 479
private networks 478
protection against simultaneous writes 480
reconfiguration of 489
resolving disk status in 482
setting disk connectivity policies in 507
setting failure policies in 507
shared disk groups 479
shared objects 480
splitting disk groups in 506
upgrading cluster protocol version 509
use of DMP in 169
vol_fmr_logsz tunable 558
volume reconfiguration 492
vxclust 489
vxclustadm 490
vxdctl 500
vxrecover 509
vxstat 510
columns
changing number of 367
checking number in volume 538
in striping 43
mirroring in striped-mirror volumes 318
CommandCentral Storage 77
comment
plex attribute 292
subdisk attribute 279
concatenated volumes 40, 296
concatenated-mirror volumes
converting to mirrored-concatenated 370
creating 311
defined 49

concatenated-mirror volumes *(continued)*
recovery 298
concatenation 40
condition flags for plexes 286
configuration backup and restoration 269
configuration changes
monitoring using vxnotify 270
configuration copies for disk group 555
configuration database
checking number of copies 536
checking size of 535
copy size 217
in private region 85
listing disks with 229
metadata 229
reducing size of 250
configuring
shared disks 620
connectivity policies 482
setting for disk groups 507
Controller ID
displaying 185
controllers
checking for disabled 539
disabling for DMP 203
disabling in DMP 172
displaying information about 182
enabling for DMP 204
mirroring across 309, 318
mirroring guidelines 616
number 26
specifying to vxassist 305
upgrading firmware 204
convertname
displaying device name conversion 635
copy-on-write
used by instant snapshots 377
copymaps 71
Cross-platform Data Sharing (CDS)
alignment constraints 304
disk format 86
customized naming
DMP nodes 175
CVM
cluster functionality of VxVM 475

D

d# 26, 83
data change object
DCO 70
data redundancy 47–48, 50
data volume configuration 64
database replay logs and sequential DRL 63
databases
resilvering 64
resynchronizing 64
DCO
adding to RAID-5 volumes 343
adding version 0 DCOs to volumes 429
adding version 20 DCOs to volumes 341
calculating plex size for version 20 72
considerations for disk layout 256
creating volumes with version 0 DCOs attached 311
creating volumes with version 20 DCOs attached 315
data change object 70
determining version of 344
dissociating version 0 DCOs from volumes 432
effect on disk group split and join 256
log plexes 73
log volume 70
moving log plexes 343, 431
reattaching version 0 DCOs to volumes 432
removing version 0 DCOs from volumes 432
specifying storage for version 0 plexes 431
specifying storage for version 20 plexes 342
used with DRL 62
version 0 70
version 20 71
versioning 70
dcolen attribute 71, 313, 430
DCOSNP
plex state 284
DDI_NT_FABRIC property 88
DDL 29
Device Discovery Layer 93
decision support
implementing 449
default disk group 218
defaultdg 218–219
defaults
for vxdisk 87, 113
for vxencap 113
description file with vxmake 323
detach policy
global 484

detach policy *(continued)*
local 484
DETACHED
plex kernel state 287
volume kernel state 331
devalias command 136
device attributes
extended 185
device discovery
introduced 29
partial 88
Device Discovery Layer 93
Device Discovery Layer (DDL) 29, 93
device files to access volumes 326, 620
device names 26, 82
configuring persistent 108
user-specified 175
device nodes
controlling access for volume sets 441
displaying access for volume sets 441
enabling access for volume sets 440
for volume sets 439
devices
adding foreign 104
fabric 88
JBOD 89
listing all 94
metadevices 83
nopriv 127
path redundancy 195–196
pathname 83
volatile 122
dgalign_checking attribute 305
dgfailpolicy attribute 487
dirty bits in DRL 63
dirty flags set on volumes 61
dirty region logging.. *See* DRL
dirty regions 561
disable failure policy 485
DISABLED
plex kernel state 287
volume kernel state 331
disabled paths 175
disk access records
stored in /etc/vx/darecs 269
disk arrays
A/A 164
A/A-A 164
A/P 164
disk arrays *(continued)*
A/P-C 165
A/PF 164
A/PF-C 165
A/PG 164
A/PG-C 165
Active/Active 164
Active/Passive 164
adding disks to DISKS category 101
adding vendor-supplied support package 90
Asymmetric Active/Active 164
defined 27
excluding support for 100
JBOD devices 89
listing excluded 100
listing supported 99
listing supported disks in DISKS category 100
multipathed 28
re-including support for 100
removing disks from DISKS category 103
removing vendor-supplied support package 91
supported with DMP 99
disk drives
variable geometry 617
disk duplexing 47, 318
disk groups
activating shared 506
activation in clusters 481
adding disks to 223
avoiding conflicting minor numbers on import 241
boot disk group 218
bootdg 218
checking for non-imported 536
checking initialized disks 537
checking number of configuration copies in 536
checking on disk config size 536
checking size of configuration database 535
checking version number 536
clearing locks on disks 239
cluster-shareable 479
compatible with CDS 222
configuration backup and restoration 269
configuring site consistency on 519
configuring site-based allocation on 518
converting to private 505
creating 221
creating shared 503
creating with old version number 268

disk groups *(continued)*
default disk group 218
defaultdg 218
defaults file for shared 481
defined 34
deporting 224
designating as shareable 479
destroying 263
determining the default disk group 218
disabling 263
displaying boot disk group 219
displaying default disk group 219
displaying free space in 221
displaying information about 220
displaying version of 268
effect of size on private region 217
elimination of rootdg 216
failure policy 485
features supported by version 265
forcing import of 240
free space in 461
impact of number of configuration copies on performance 555
importing 226
importing as shared 504
importing forcibly 504
importing with cloned disks 228
joining 252, 261
layout of DCO plexes 256
limitations of move
split. *See* and join
listing objects affected by a move 255
listing shared 502
making site consistent 523
moving between systems 238
moving disks between 237, 258
moving licensed EMC disks between 258
moving objects between 251, 257
moving objects in clusters 505
names reserved by system 217
nodg 218
number of spare disks 539
private in clusters 479
recovering destroyed 263
recovery from failed reconfiguration 254
removing disks from 223
renaming 235
reorganizing 250
reserving minor numbers 241
disk groups *(continued)*
restarting moved volumes 259–260, 262
root 34
rootdg 34, 216
serial split brain condition 244
setting connectivity policies in clusters 507
setting default disk group 219
setting failure policies in clusters 507
setting number of configuration copies 555
shared in clusters 479
specifying to commands 217
splitting 251, 260
splitting in clusters 506
Storage Expert rules 535
upgrading version of 264, 268
version 264, 267
disk media names 34, 82
disk names 82
configuring persistent 108
disk sparing
Storage Expert rules 538
disk## 35
disk##-## 35
diskdetpolicy attribute 487
diskgroup## 82
disks 90
adding 122
adding to disk groups 223
adding to disk groups forcibly 504
adding to DISKS category 101
aliased 136
array support library 90
auto-configured 86
c0t0d0 84
categories 90
CDS format 86
changing default layout attributes 113
changing naming scheme 107
checking for failed 539
checking initialized disks not in disk group 537
checking number of configuration copies in disk group 536
checking proportion spare in disk group 539
clearing locks on 239
cloned 228
complete failure messages 460
configuring persistent names 108
default encapsulation values 113
default initialization values 113

disks *(continued)*
determining failed 460
determining if shared 501
Device Discovery Layer 93
disabled path 175
discovery of by VxVM 87, 89
disk access records file 269
disk arrays 27
displaying information 155–156
displaying information about 155, 220
displaying naming scheme 107
displaying spare 463
dynamic LUN expansion 138
enabled path 175
enabling 152
enabling after hot swap 152
enabling Extended Copy Service 140
encapsulating 105
encapsulation 123, 128
enclosures 29
excluding free space from hot-relocation use 465
failure handled by hot-relocation 457
formatting 112
handling clones 227
handling duplicated identifiers 227
hot-relocation 455
initializing 105, 113
installing 112
invoking discovery of 91
layout of DCO plexes 256
listing tags on 228
listing those supported in JBODs 100
making available for hot-relocation 463
making free space available for hot-relocation use 466
marking as spare 463
media name 82
metadevices 83
mirroring boot disk 131
mirroring root disk 131
mirroring volumes on 337
moving between disk groups 237, 258
moving disk groups between systems 238
moving volumes from 360
names 82
naming schemes 83
nopriv 86
nopriv devices 127
number 26
disks *(continued)*
obtaining performance statistics 551
OTHER_DISKS category 90
partial failure messages 459
postponing replacement 145
primary path 175
putting under control of VxVM 105
reinitializing 121
releasing from disk groups 263
removing 141, 145
removing from A5x00 arrays 151
removing from disk groups 223
removing from DISKS category 103
removing from pool of hot-relocation spares 464
removing from VxVM control 144, 224
removing tags from 230
removing with subdisks 143–144
renaming 153
replacing 145
replacing in A5x00 arrays 151
replacing removed 149
reserving for special purposes 154
resolving status in clusters 482
root disk 128
scanning for 87
secondary path 175
setting connectivity policies in clusters 507
setting failure policies in clusters 507
setting tags on 228
simple 86
simple format 86
sliced 86
sliced format 86
spare 461
specifying to vxassist 305
stripe unit size 617
tagging with site name 517
taking offline 153
UDID flag 227
unique identifier 227
unreserving 155
upgrading contoller firmware 204
VM 34
writing a new identifier to 227
DISKS category 90
adding disks 101
listing supported disks 100
removing disks 103

displaying
DMP nodes 178
HBA information 185
redundancy levels 195
supported disk arrays 99
displaying statistics
erroneous I/Os 190
queued I/Os 190
DMP
booting from DMP devices 169
check_all restore policy 210
check_alternate restore policy 210
check_disabled restore policy 211
check_periodic restore policy 211
configuring DMP path restoration policies 210
configuring I/O throttling 207
configuring response to I/O errors 206, 209
disabling array ports 203
disabling controllers 203
disabling multipathing 170
disabling paths 203
displaying DMP database information 173
displaying DMP node for a path 177
displaying DMP node for an enclosure 177–178
displaying DMP nodes 178
displaying information about array ports 183
displaying information about controllers 182
displaying information about enclosures 183
displaying information about paths 174
displaying LUN group for a node 179
displaying paths controlled by DMP node 180
displaying paths for a controller 180
displaying paths for an array port 181
displaying recoveryoption values 209
displaying status of DMP error handling thread 212
displaying status of DMP path restoration thread 212
displaying TPD information 184
dynamic multipathing 163
enabling array ports 204
enabling controllers 204
enabling multipathing 172
enabling paths 204
enclosure-based naming 165
gathering I/O statistics 188
in a clustered environment 169
load balancing 168
logging levels 567
DMP *(continued)*
metanodes 165
nodes 165
path aging 566
path failover mechanism 167
path-switch tunable 568
renaming an enclosure 205
restore policy 210
scheduling I/O on secondary paths 200
setting the DMP restore polling interval 210
stopping the DMP restore daemon 212
vxdmpadm 176
DMP nodes
displaying consolidated information 178
setting names 175
DMP support
JBOD devices 89
dmp_cache_open tunable 565
dmp_daemon_count tunable 565
dmp_delayq_interval tunable 565
dmp_failed_io_threshold tunable 566
dmp_fast_recovery tunable 566
dmp_health_time tunable 566
dmp_log_level tunable 567
dmp_path_age tunable 567
dmp_pathswitch_blks_shift tunable 568
dmp_probe_idle_lun tunable 568
dmp_queue_depth tunable 568
dmp_retry_count tunable 569
dmp_scsi_timeout tunable 570
dmp_stat_interval tunable 570
doconvert
beginning SDS to VxVM conversion 637
DR
dynamic reconfiguration 168
DRL
adding log subdisks 276
adding logs to mirrored volumes 349
checking existence of 534
checking existence of mirror 534
creating volumes with DRL enabled 315
determining if active 346
determining if enabled 345
dirty bits 63
dirty region logging 62
disabling 346
enabling on volumes 342
handling recovery in clusters 497
hot-relocation limitations 457

DRL *(continued)*
log subdisks 63
maximum number of dirty regions 561
minimum number of sectors 561
operation in clusters 496
re-enabling 346
recovery map in version 20 DCO 71
removing logs from mirrored volumes 351
removing support for 346
sequential 63
use of DCO with 62
drl attribute 315, 349
DRL guidelines 616
duplexing 47, 318
dynamic LUN expansion 138
dynamic reconfiguration 168

E

ecopy 139
eeprom
using boot disk aliases 135
EEPROM variables
use-nvramrc? 135
EMC arrays
moving disks between disk groups 258
EMC PowerPath
coexistence with DMP 92
EMC Symmetrix
autodiscovery 92
EMPTY
plex state 284
volume state 330
ENABLED
plex kernel state 287
volume kernel state 331
enabled paths
displaying 175
encapsulating disks 123, 128
encapsulation
default attributes 113
failure of 127
of disks 105
enclosure-based naming 29, 84, 107
displayed by vxprint 112
DMP 165
enclosures 29
discovering disk access names in 112
displaying information about 183
issues with nopriv disks 110
enclosures *(continued)*
issues with simple disks 110
mirroring across 318
path redundancy 195–196
setting attributes of paths 193, 196
tagging with site name 516
erroneous I/Os
displaying statistics 190
error messages
Association count is incorrect 500
Association not resolved 500
Cannot auto-import group 499
cannot open altboot_disk 135
Configuration records are inconsistent 500
Disk for disk group not found 240
Disk group has no valid configuration copies 240, 500
Disk group version doesn't support feature 264
Disk is in use by another host 239
Disk is used by one or more subdisks 223
Disk not moving but subdisks on it are 255
Duplicate record in configuration 500
import failed 239
It is not possible to encapsulate 127
No valid disk found containing disk group 239
The encapsulation operation failed 127
tmpsize too small to perform this relayout 57
unsupported layout 127
Volume has different organization in each mirror 354
vxdg listmove failed 255
errord daemon 166
exclusive-write mode 481
exclusivewrite mode 480
explicit failover mode 164
extended attributes
devices 185
Extended Copy Service
enabling for a disk 140
introduced 139

F

fabric devices 88
FAILFAST flag 167
failover 476–477
failover mode 164
failure handled by hot-relocation 457
failure in RAID-5 handled by hot-relocation 457

failure policies 485
    setting for disk groups 507
FastResync
    checking if enabled on volumes 362
    disabling on volumes 363
    effect of growing volume on 75
    enabling on new volumes 313
    enabling on volumes 362
    limitations 75
    Non-Persistent 69
    Persistent 70, 72
    size of bitmap 558
    snapshot enhancements 375
    use with snapshots 68
fastresync attribute 313, 315, 362
file systems
    growing using vxresize 354
    mirroring on root disk 136
    permitted resizing operations 354
    shrinking using vxresize 354
    unmounting 359
fire drill
    defined 512
    testing 524
firmware
    upgrading 204
FMR.. *See* FastResync
foreign devices
    adding 104
formatting disks 112
free space in disk groups 461
full-sized instant snapshots 376
    creating 397
    creating volumes for use as 393
fullinst snapshot type 415

G

GAB 488
global detach policy 484
Group Membership and Atomic Broadcast (GAB) 488
guidelines
    DRL 616
    mirroring 616
    RAID-5 618

H

hasdcolog attribute 363
HBA information
    displaying 185
HBAs
    listing ports 95
    listing supported 95
    listing targets 95
highwatermark attribute 392
host failures 526
hostnames
    checking 539
hot-relocation
    complete failure messages 460
    configuration summary 462
    daemon 456
    defined 77
    detecting disk failure 457
    detecting plex failure 457
    detecting RAID-5 subdisk failure 457
    excluding free space on disks from use by 465
    limitations 457
    making free space on disks available for use by 466
    marking disks as spare 463
    modifying behavior of 473
    notifying users other than root 474
    operation of 455
    partial failure messages 459
    preventing from running 474
    reducing performance impact of recovery 474
    removing disks from spare pool 464
    Storage Expert rules 538
    subdisk relocation 462
    subdisk relocation messages 467
    unrelocating subdisks 467
    unrelocating subdisks using vxassist 469
    unrelocating subdisks using vxdiskadm 468
    unrelocating subdisks using vxunreloc 470
    use of free space in disk groups 461
    use of spare disks 461
    use of spare disks and free space 462
    using only spare disks for 467
    vxrelocd 456

I

I/O
    gathering statistics for DMP 188
    kernel threads 25
    scheduling on secondary paths 200
    throttling 167

I/O *(continued)*
use of statistics in performance tuning 550
using traces for performance tuning 553
I/O operations
maximum size of 559
I/O policy
displaying 196
example 200
specifying 197
I/O throttling 207
I/O throttling options
configuring 209
identifiers for tasks 332
idle LUNs 568
implicit failover mode 164
init attribute 324
initialization
default attributes 113
of disks 105, 113
initialization of disks 105
instant snapshots
backing up multiple volumes 404
cascaded 381
creating backups 387
creating for volume sets 405
creating full-sized 397
creating space-optimized 394
creating volumes for use as full-sized 393
displaying information about 414
dissociating 412
full-sized 376
improving performance of synchronization 417
reattaching 410
refreshing 409
removing 413
removing support for 346
restoring volumes using 411
space-optimized 378
splitting hierarchies 413
synchronizing 416
Intelligent Storage Provisioning (ISP) 37
intent logging 374
INVALID volume state 330
ioctl calls 560
IOFAIL plex condition 286
IOFAIL plex state 284
iSCSI parameters
administering with DDL 97
setting with vxddladm 97
ISP volumes 37

J

JBOD
DMP support 89
JBODs
adding disks to DISKS category 101
listing supported disks 100
removing disks from DISKS category 103

K

kernel states
for plexes 287
volumes 331

L

layered volumes
converting to non-layered 370
defined 54, 298
striped-mirror 48
layout attributes
changing for disks 113
layouts
changing default used by vxassist 305
left-symmetric 52
specifying default 305
types of volume 296
leave failure policy 485
left-symmetric layout 52
len subdisk attribute 279
link objects 380
linked break-off snapshots 380
creating 402
linked third-mirror snapshots
reattaching 411
listing
alternate boot disks 132
DMP nodes 178
supported disk arrays 99
load balancing 164
across nodes in a cluster 477
displaying policy for 196
specifying policy for 197
load-balancing
specifying policy for 197
local detach policy 484
lock clearing on disks 239
LOG plex state 284

log subdisks 616
    associating with plexes 276
    DRL 63
logdisk 314, 320–321
logical units 164
loglen attribute 316
logs
    adding DRL log 349
    adding for RAID-5 351
    adding sequential DRL logs 349
    adding to volumes 340
    checking for disabled 537
    checking for multiple RAID-5 logs on same disk 534
    RAID-5 54, 61
    removing DRL log 351
    removing for RAID-5 352
    removing sequential DRL logs 351
    resizing using vxvol 357
    specifying number for RAID-5 320
    usage with volumes 298
logtype attribute 315
LUN 164
LUN expansion 138
LUN group failover 164
LUN groups
    displaying details of 179
LUNs
    idle 568
    thin provisioning 609

M

maps
    adding to volumes 340
    usage with volumes 298
master node
    defined 478
    discovering 500
maxautogrow attribute 392
maxdev attribute 243
memory
    granularity of allocation by VxVM 561
    maximum size of pool for VxVM 562
    minimum size of pool for VxVM 564
    persistence of FastResync in 69
messages
    complete disk failure 460
    hot-relocation of subdisks 467
    partial disk failure 459
metadata 229
metadevices 83
metanodes
    DMP 165
migrating to thin storage 609
minimum queue load balancing policy 198
minimum redundancy levels
    displaying for a device 195
    specifying for a device 196
minor numbers 241
mirbrk snapshot type 415
mirdg attribute 403
mirrored volumes
    adding DRL logs 349
    adding sequential DRL logs 349
    changing read policies for 358
    checking existence of mirrored DRL 534
    checking existence of without DRL 534
    configuring VxVM to create by default 336
    creating 310
    creating across controllers 309, 318
    creating across enclosures 318
    creating across targets 307
    defined 297
    dirty region logging 62
    DRL 62
    FastResync 62
    FR 62
    logging 62
    performance 546
    removing DRL logs 351
    removing sequential DRL logs 351
    snapshots 68
mirrored-concatenated volumes
    converting to concatenated-mirror 370
    creating 311
    defined 48
mirrored-stripe volumes
    benefits of 47
    checking configuration 538
    converting to striped-mirror 370
    creating 317
    defined 297
    performance 547
mirroring
    boot disk 131
    defined 47
    guidelines 616
    root disk 131

mirroring controllers 616
mirroring plus striping 48
mirrors
    adding to volumes 336
    creating snapshot 422
    defined 38
    removing from volumes 339
    specifying number of 311
mirvol attribute 403
mirvol snapshot type 415
monitor_fabric tunable 567
MPxIO mode 168
mrl
    keyword 195
Multi-Volume Support 435
multipathing
    disabling 170
    displaying information about 174
    enabling 172

N

names
    changing for disk groups 235
    defining for snapshot volumes 425
    device 26, 82
    disk 82
    disk media 34, 82
    plex 37
    plex attribute 292
    renaming disks 153
    subdisk 35
    subdisk attribute 278
    VM disk 35
    volume 37
naming
    DMP nodes 175
naming scheme
    changing for disks 107
    changing for TPD enclosures 109
    displaying for disks 107
naming schemes
    for disks 83
ncachemirror attribute 395
ndcomirror attribute 313, 315, 430
ndcomirs attribute 342, 390
NEEDSYNC volume state 330
newvol attribute 401
nmirror attribute 400–401
NODAREC plex condition 286
nodes
    DMP 165
    in clusters 477
    maximum number in a cluster 476
    node abort in clusters 496
    requesting status of 500
    shutdown in clusters 495
    use of vxclust 489
    use of vxclustadm to control cluster functionality 490
NODEVICE plex condition 286
nodg 218
nomanual path attribute 193
non-autotrespass mode 164
non-layered volume conversion 370
Non-Persistent FastResync 69
nopreferred path attribute 194
nopriv devices 127
nopriv disk type 86
nopriv disks
    issues with enclosures 110

O

objects
    physical 26
    virtual 31
off-host processing 443, 476
OFFLINE plex state 285
online backups
    implementing 445
online invalid status 156
online relayout
    changing number of columns 367
    changing region size 369
    changing speed of 369
    changing stripe unit size 367
    combining with conversion 370
    controlling progress of 369
    defined 56
    destination layouts 363
    failure recovery 60
    how it works 56
    limitations 59
    monitoring tasks for 369
    pausing 369
    performing 363
    resuming 369
    reversing direction of 369
    specifying non-default 367

online relayout *(continued)*
specifying plexes 368
specifying task tags for 368
temporary area 57
transformation characteristics 60
transformations and volume length 60
types of transformation 364
viewing status of 368
online status 156
OpenBoot PROMs (OPB) 134
ordered allocation 307, 314, 321
OS-based naming 83
OTHER_DISKS category 90
overlapped seeks 616

P

parity in RAID-5 50
partial device discovery 88
partition size
displaying the value of 196
specifying 198
partitions
number 26–27
s2 84, 87
s3 87
s4 87
slices 27
path aging 566
path failover in DMP 167
pathgroups
creating 171
paths
disabling for DMP 203
enabling for DMP 204
setting attributes of 193, 196
performance
analyzing data 550
benefits of using VxVM 545
changing values of tunables 555
combining mirroring and striping 547
displaying tunable values 556
effect of read policies 547
examining ratio of reads to writes 553
hot spots identified by I/O traces 553
impact of number of disk group configuration copies 555
improving for instant snapshot synchronization 417
load balancing in DMP 168
performance *(continued)*
mirrored volumes 546
monitoring 548
moving volumes to improve 551
obtaining statistics for disks 551
obtaining statistics for volumes 549
RAID-5 volumes 547
setting priorities 548
striped volumes 546
striping to improve 552
tracing volume operations 549
tuning large systems 554
tuning VxVM 554
using I/O statistics 550
persistence
device naming option 107
persistent device name database 108
persistent device naming 108
Persistent FastResync 70–72
physical disks
adding to disk groups 223
clearing locks on 239
complete failure messages 460
determining failed 460
displaying information 155
displaying information about 155, 220
displaying spare 463
enabling 152
enabling after hot swap 152
excluding free space from hot-relocation use 465
failure handled by hot-relocation 457
initializing 105
installing 112
making available for hot-relocation 463
making free space available for hot-relocation use 466
marking as spare 463
moving between disk groups 237, 258
moving disk groups between systems 238
moving volumes from 360
partial failure messages 459
postponing replacement 145
releasing from disk groups 263
removing 141, 145
removing from A5x00 arrays 151
removing from disk groups 223
removing from pool of hot-relocation spares 464
removing with subdisks 143–144
replacing 145

physical disks *(continued)*
replacing in A5x00 arrays 151
replacing removed 149
reserving for special purposes 154
spare 461
taking offline 153
unreserving 155
physical objects 26
ping-pong effect 169
plex attribute 401
plex conditions
IOFAIL 286
NODAREC 286
NODEVICE 286
RECOVER 287
REMOVED 287
plex kernel states
DETACHED 287
DISABLED 287
ENABLED 287
plex states
ACTIVE 284
CLEAN 284
DCOSNP 284
EMPTY 284
IOFAIL 284
LOG 284
OFFLINE 285
SNAPATT 285
SNAPDIS 285
SNAPDONE 285
SNAPTMP 285
STALE 285
TEMP 285
TEMPRM 286
TEMPRMSD 286
plexes
adding to snapshots 427
associating log subdisks with 276
associating subdisks with 275
associating with volumes 287
attaching to volumes 287
changing attributes 292
changing read policies for 358
checking for detached 537
checking for disabled 537
comment attribute 292
complete failure messages 460
condition flags 286
plexes *(continued)*
converting to snapshot 424
copying 291
creating 282
creating striped 282
defined 36
detaching from volumes temporarily 289
disconnecting from volumes 288
displaying information about 283
dissociating from volumes 291
dissociating subdisks from 277
failure in hot-relocation 457
kernel states 287
limit on number per volume 548
maximum number of subdisks 561
maximum number per volume 37
mirrors 38
moving 290, 343, 431
name attribute 292
names 37
partial failure messages 459
putil attribute 292
putting online 289, 335
reattaching 289
recovering after correctable hardware failure 460
removing 291
removing from volumes 339
sparse 60, 276, 287, 290
specifying for online relayout 368
states 283
striped 43
taking offline 288, 335
tutil attribute 292
types 36
polling interval for DMP restore 210
ports
listing 95
PowerPath
coexistence with DMP 92
preconvert
setting up SDS for conversion 634
prefer read policy 358
preferred plex
read policy 358
preferred priority path attribute 194
primary boot disk failure 132
primary path 164, 175
primary path attribute 194

priority load balancing 199
private disk groups
    converting from shared 505
    in clusters 479
private network
    in clusters 478
private region
    checking size of configuration database 535
    configuration database 85
    defined 85
    effect of large disk groups on 217
PROMs
    boot 134
prtconf
    displaying tunables 556
public region 85
putil
    plex attribute 292
    subdisk attribute 278

Q

queued I/Os
    displaying statistics 190

R

RAID-0 42
RAID-0+1 47
RAID-1 47
RAID-1+0 48
RAID-5
    adding logs 351
    adding subdisks to plexes 276
    checking existence of log 535
    checking log is mirrored 535
    checking size of log 535
    guidelines 618
    hot-relocation limitations 457
    logs 54, 61
    parity 50
    removing logs 352
    specifying number of logs 320
    subdisk failure handled by hot-relocation 457
    volumes 50
RAID-5 volumes
    adding DCOs to 343
    adding logs 351
    changing number of columns 367
    changing stripe unit size 367
RAID-5 volumes *(continued)*
    checking existence of RAID-5 log 535
    checking number of columns 538
    checking RAID-5 log is mirrored 535
    checking size of RAID-5 log 535
    creating 319
    defined 297
    performance 547
    removing logs 352
raw device nodes
    controlling access for volume sets 441
    displaying access for volume sets 441
    enabling access for volume sets 440
    for volume sets 439
read policies
    changing 358
    performance of 547
    prefer 358
    round 358
    select 358
    siteread 358, 513, 518, 520
read-only mode 481
readonly mode 480
RECOVER plex condition 287
recovery
    checkpoint interval 557
    I/O delay 557
    preventing on restarting volumes 336
recovery accelerator 64
recovery option values
    configuring 209
recovery time
    Storage Expert rules 534
redo log configuration 65
redundancy
    checking for volumes 537
    of data on mirrors 297
    of data on RAID-5 297
redundancy levels
    displaying for a device 195
    specifying for a device 196
redundant-loop access 31
region 85
regionsize attribute 342, 390, 392
reinitialization of disks 121
relayout
    changing number of columns 367
    changing region size 369
    changing speed of 369

relayout *(continued)*
changing stripe unit size 367
combining with conversion 370
controlling progress of 369
limitations 59
monitoring tasks for 369
online 56
pausing 369
performing online 363
resuming 369
reversing direction of 369
specifying non-default 367
specifying plexes 368
specifying task tags for 368
storage 56
transformation characteristics 60
types of transformation 364
viewing status of 368
relocation
automatic 455
complete failure messages 460
limitations 457
partial failure messages 459
Remote Mirror feature
administering 512
remote mirrors
administering 512
REMOVED plex condition 287
removing disks 145
removing physical disks 141
replacing disks 145
replay logs and sequential DRL 63
REPLAY volume state 330
resilvering
databases 64
restoration of disk group configuration 269
restore policy
check_all 210
check_alternate 210
check_disabled 211
check_periodic 211
restored daemon 166
resyncfromoriginal snapback 386
resyncfromreplica snapback 386
resynchronization
checkpoint interval 557
I/O delay 557
of volumes 61
resynchronizing
databases 64
retry option values
configuring 209
root disk
defined 128
encapsulating 131
listing volumes on 136
mirroring 131
mirroring other file systems on 136
unencapsulating 137
root disk group 34, 216
root disks
booting alternate 132
root mirrors
checking 539
root volume 129–130
rootability 128
checking 539
removing 137
rootdg 34
round read policy 358
round-robin
load balancing 199
read policy 358
rules
attributes 544
checking attribute values 531
checking disk group configuration copies 536
checking disk group version number 536
checking for full disk group configuration database 535
checking for initialized disks 537
checking for mirrored DRL 534
checking for multiple RAID-5 logs on a disk 534
checking for non-imported disk groups 536
checking for non-mirrored RAID-5 log 535
checking for RAID-5 log 535
checking hardware 539
checking hot-relocation 538
checking mirrored volumes without a DRL 534
checking mirrored-stripe volumes 538
checking number of configuration copies in disk group 536
checking number of RAID-5 columns 538
checking number of spare disks 539
checking number of stripe columns 538
checking on disk config size 536
checking plex and volume states 537

rules *(continued)*
checking RAID-5 log size 535
checking rootability 539
checking stripe unit size 538
checking system name 539
checking volume redundancy 537
definitions 540–541
finding information about 531
for checking hardware 539
for checking rootability 539
for checking system name 539
for disk groups 535
for disk sparing 538
for recovery time 534
for striped mirror volumes 537
listing attributes 531
result types 532
running 532
setting values of attributes 532

S

s# 26–27, 83
s2 partition 84, 87
s3 partition 87
s4 partition 87
SAN storage
configuring volumes on 77
scandisks
vxdisk subcommand 87
SDS
conversion overview 628
converting to VxVM 637
mapping to VxVM objects 625
planning for conversion 630
post-conversion tasks 638
preparing for conversion 633
root disk conversion 639
setting up for conversion 634
secondary path 164
secondary path attribute 194
secondary path display 175
select read policy 358
sequential DRL
defined 63
maximum number of dirty regions 561
sequential DRL attribute 315
serial split brain condition 512
correcting 249
in campus clusters 244
serial split brain condition *(continued)*
in disk groups 244
setting
path redundancy levels 196
shared disk groups
activating 506
activation modes 480–481
converting to private 505
creating 503
importing 504
in clusters 479
limitations of 487
listing 502
shared disks
configuring 620
shared-read mode 481
shared-write mode 481
sharedread mode 480
sharedwrite mode 480
showconvert
displaying SDS conversion plan 635
simple disk format 86
simple disk type 86
simple disks
issues with enclosures 110
single active path policy 199
Site Awareness license 514
site consistency
configuring 519
defined 512
site failure
simulating 525
site failures
host failures 526
loss of connectivity 526
recovery from 525, 527
scenarios and recovery procedures 525
storage failures 527
site storage class 521
site-based allocation
configuring for disk groups 518
configuring for volumes 521
defined 512
site-based consistency
configuring on existing disk groups 523
siteconsistent attribute 519
siteread read policy 358, 513, 518, 520
sites
reattaching 525

size units 272
slave nodes
    defined 478
sliced disk format 86
sliced disk type 86
slices
    number 26
    partitions 27
    s2 84, 87
    s3 87
    s4 87
SmartMove feature 609
    setting up 303
SmartSync 64
    disabling on shared disk groups 561
    enabling on shared disk groups 561
snap objects 74
snap volume naming 386
snapabort 375
SNAPATT plex state 285
snapback
    defined 376
    merging snapshot volumes 426
    resyncfromoriginal 386
    resyncfromreplica 386, 426
snapclear
    creating independent volumes 427
SNAPDIS plex state 285
SNAPDONE plex state 285
snapmir snapshot type 415
snapshot hierarchies
    creating 408
    splitting 413
snapshot mirrors
    adding to volumes 407
    removing from volumes 408
snapshots
    adding mirrors to volumes 407
    adding plexes to 427
    and FastResync 68
    backing up multiple volumes 404, 425
    backing up volumes online using 387
    cascaded 381
    comparison of features 67
    converting plexes to 424
    creating a hierarchy of 408
    creating backups using third-mirror 420
    creating for volume sets 405
    creating full-sized instant 397
snapshots *(continued)*
    creating independent volumes 427
    creating instant 387
    creating linked break-off 402
    creating snapshots of 383
    creating space-optimized instant 394
    creating third-mirror break-off 399
    creating volumes for use as full-sized instant 393
    defining names for 425
    displaying information about 428
    displaying information about instant 414
    dissociating instant 412
    emulation of third-mirror 379
    finding out those configured on a cache 420
    full-sized instant 67, 376
    hierarchy of 381
    improving performance of synchronization 417
    linked break-off 380
    listing for a cache 418
    merging with original volumes 426
    of volumes 65
    on multiple volumes 386
    reattaching instant 410
    reattaching linked third-mirror 411
    refreshing instant 409
    removing 424
    removing instant 413
    removing linked snapshots from volumes 408
    removing mirrors from volumes 408
    restoring from instant 411
    resynchronization on snapback 386
    resynchronizing volumes from 426
    space-optimized instant 68, 378
    synchronizing instant 416
    third-mirror 66
    use of copy-on-write mechanism 377
snapstart 375
SNAPTMP plex state 285
snapvol attribute 397, 403
snapwait 400, 403
Solstice DiskSuite
    migrating from 623
source attribute 397, 403
space-optimized instant snapshots 378
    creating 394
spaceopt snapshot type 415
spanned volumes 40
spanning 40

spare disks
checking proportion in disk group 539
displaying 463
marking disks as 463
used for hot-relocation 461
sparse plexes 60, 276, 287, 290
specifying
redundancy levels 196
SSTM 168
STALE plex state 285
standby path attribute 194
states
for plexes 283
of link objects 380
volume 330
statistics gathering 167
storage
ordered allocation of 307, 314, 321
storage accounts 77
storage attributes and volume layout 305
storage cache
used by space-optimized instant snapshots 378
Storage Expert
check keyword 531
checking default values of rule attributes 531
command-line syntax 530
diagnosing configuration issues 533
info keyword 531
introduced 529
list keyword 531
listing rule attributes 531
obtaining a description of a rule 531
requirements 530
rule attributes 544
rule definitions 540–541
rule result types 532
rules 530
rules engine 530
run keyword 532
running a rule 532
setting values of rule attributes 532
vxse 529
storage failures 527
storage groups 77
storage processor 164
storage relayout 56
stripe columns 43
stripe unit size recommendations 617
stripe units
changing size 367
checking size 538
defined 43
stripe-mirror-col-split-trigger-pt 318
striped plexes
adding subdisks 276
defined 43
striped volumes
changing number of columns 367
changing stripe unit size 367
checking number of columns 538
checking stripe unit size 538
creating 316
defined 297
failure of 42
performance 546
specifying non-default number of columns 317
specifying non-default stripe unit size 317
Storage Expert rules 537
striped-mirror volumes
benefits of 48
converting to mirrored-stripe 370
creating 318
defined 298
mirroring columns 318
mirroring subdisks 318
performance 547
trigger point for mirroring 318
striping 42
striping guidelines 617
striping plus mirroring 47
subdisk names 35
subdisks
associating log subdisks 276
associating with plexes 275
associating with RAID-5 plexes 276
associating with striped plexes 276
blocks 35
changing attributes 278
comment attribute 279
complete failure messages 460
copying contents of 273
creating 272
defined 35
determining failed 460
displaying information about 272
dissociating from plexes 277
dividing 274

subdisks *(continued)*
DRL log 63
hot-relocation 77, 455, 462
hot-relocation messages 467
joining 274
len attribute 279
listing original disks after hot-relocation 471
maximum number per plex 561
mirroring in striped-mirror volumes 318
moving after hot-relocation 467
moving contents of 273
name attribute 278
partial failure messages 459
physical disk placement 615
putil attribute 278
RAID-5 failure of 457
RAID-5 plex, configuring 618
removing from VxVM 278
restrictions on moving 273
specifying different offsets for unrelocation 471
splitting 274
tutil attribute 278
unrelocating after hot-relocation 467
unrelocating to different disks 470
unrelocating using vxassist 469
unrelocating using vxdiskadm 468
unrelocating using vxunreloc 470
Sun StorEdge T3 168
Sun StorEdge Traffic Manager (SSTM) 168
SYNC volume state 331
synchronization
controlling for instant snapshots 416
improving performance of 417
syncing attribute 388, 416
syncpause 416
syncresume 416
syncstart 416
syncstop 416
syncwait 416
system names
checking 539

T

t# 26, 83
tags
for tasks 332
listing for disks 228
removing from disks 230
removing from volumes 357
tags *(continued)*
renaming 357
setting on disks 228
setting on volumes 321, 357
specifying for online relayout tasks 368
specifying for tasks 332
target IDs
number 26
specifying to vxassist 305
target mirroring 307, 318
targets
listing 95
task monitor in VxVM 332
tasks
aborting 333
changing state of 333
identifiers 332
listing 333
managing 332
modifying parameters of 333
monitoring 333
monitoring online relayout 369
pausing 333
resuming 333
specifying tags 332
specifying tags on online relayout operation 368
tags 332
TEMP plex state 285
temporary area used by online relayout 57
TEMPRM plex state 286
TEMPRMSD plex state 286
thin provisioning
using 609
thin storage
using 609
third-mirror
snapshots 66
third-mirror break-off snapshots
creating 399
third-mirror snapshots 379
third-party driver (TPD) 91
throttling 167
TPD
displaying path information 184
support for coexistence 91
tpdmode attribute 109
trigger point in striped-mirror volumes 318
tunables
changing values of 555

tunables *(continued)*
displaying using prtconf 556
dmp_cache_open 565
dmp_daemon_count 565
dmp_delayq_interval 565
dmp_failed_io_threshold 566
dmp_fast_recovery 566
dmp_health_time 566
dmp_log_level 567
dmp_path_age 567
dmp_pathswitch_blks_shift 568
dmp_probe_idle_lun 568
dmp_queue_depth 568
dmp_retry_count 569
dmp_scsi_timeout 570
dmp_stat_interval 570
monitor_fabric 567
vol_checkpt_default 557
vol_default_iodelay 557
vol_fmr_logsz 70, 558
vol_kmsg_resend_period 558
vol_kmsg_resend_period_usecs 559
vol_kmsg_send_period 559
vol_kmsg_send_period_usecs 559
vol_max_vol 559
vol_maxio 559
vol_maxioctl 560
vol_maxparallelio 560
vol_maxspecialio 560
vol_subdisk_num 561
volcvm_smartsync 561
voldrl_max_drtregs 561
voldrl_max_seq_dirty 63, 561
voldrl_min_regionsz 561
voliomem_chunk_size 561
voliomem_maxpool_sz 562
voliot_errbuf_dflt 562
voliot_iobuf_default 562
voliot_iobuf_limit 563
voliot_iobuf_max 563
voliot_max_open 563
volpagemod_max_memsz 564
volraid_minpool_size 564
volraid_rsrtransmax 565
tutil
plex attribute 292
subdisk attribute 278

U

UDID flag 227
udid_mismatch flag 227
UFS file system resizing 354
unencapsulating the root disk 137
units of size 272
use-nvramrc? 135
use_all_paths attribute 200
use_avid
vxddladm option 107
user-specified device names 175
usesfsmartmove parameter 303

V

V-5-1-2536 354
V-5-1-2829 264
V-5-1-552 223
V-5-1-569 499
V-5-1-587 239
V-5-2-3091 255
V-5-2-369 224
V-5-2-4292 255
version 0
of DCOs 70
version 20
of DCOs 71
versioning
of DCOs 70
versions
checking for disk group 536
disk group 264
displaying for disk group 268
upgrading 264
virtual objects 31
VM disks
aliased 136
defined 34
determining if shared 501
displaying spare 463
excluding free space from hot-relocation use 465
initializing 105
making free space available for hot-relocation use 466
marking as spare 463
mirroring volumes on 337
moving volumes from 360
names 35
postponing replacement 145
removing from pool of hot-relocation spares 464

VM disks *(continued)*
renaming 153
vol## 37
vol##-## 37
vol_checkpt_default tunable 557
vol_default_iodelay tunable 557
vol_fmr_logsz tunable 70, 558
vol_kmsg_resend_period tunable 558
vol_kmsg_resend_period_usecs tunable 559
vol_kmsg_send_period tunable 559
vol_kmsg_send_period_usecs tunable 559
vol_max_vol tunable 559
vol_maxio tunable 559
vol_maxioctl tunable 560
vol_maxparallelio tunable 560
vol_maxspecialio tunable 560
vol_subdisk_num tunable 561
volatile devices 122
volbrk snapshot type 415
volcvm_smartsync tunable 561
voldrl_max_drtregs tunable 561
voldrl_max_seq_dirty tunable 63, 561
voldrl_min_regionsz tunable 561
voliomem_chunk_size tunable 561
voliomem_maxpool_sz tunable 562
voliot_errbuf_dflt tunable 562
voliot_iobuf_default tunable 562
voliot_iobuf_limit tunable 563
voliot_iobuf_max tunable 563
voliot_max_open tunable 563
volpagemod_max_memsz tunable 564
volraid_minpool_size tunable 564
volraid_rsrtransmax tunable 565
volume kernel states
DETACHED 331
DISABLED 331
ENABLED 331
volume length, RAID-5 guidelines 618
volume resynchronization 61
volume sets
adding volumes to 437
administering 435
controlling access to raw device nodes 441
creating 436
creating instant snapshots of 405
displaying access to raw device nodes 441
enabling access to raw device nodes 440
listing details of 437
raw device nodes 439
volume sets *(continued)*
removing volumes from 439
starting 438
stopping 438
volume states
ACTIVE 330
CLEAN 330
EMPTY 330
INVALID 330
NEEDSYNC 330
REPLAY 330
SYNC 331
volumes
accessing device files 326, 620
adding DRL logs 349
adding logs and maps to 340
adding mirrors 336
adding RAID-5 logs 351
adding sequential DRL logs 349
adding snapshot mirrors to 407
adding subdisks to plexes of 276
adding to volume sets 437
adding version 0 DCOs to 429
adding version 20 DCOs to 341
advanced approach to creating 299
assisted approach to creating 300
associating plexes with 287
attaching plexes to 287
backing up 374
backing up online using snapshots 387
block device files 326, 620
boot-time restrictions 130
booting root 129–130
changing layout online 363
changing number of columns 367
changing read policies for mirrored 358
changing stripe unit size 367
character device files 326, 620
checking for disabled 537
checking for stopped 537
checking if FastResync is enabled 362
checking redundancy of 537
combining mirroring and striping for performance 547
combining online relayout and conversion 370
concatenated 40, 296
concatenated-mirror 49, 298
configuring exclusive open by cluster node 507–508

volumes *(continued)*
configuring on SAN storage 77
configuring site consistency on 520
configuring site-based allocation on 521
converting between layered and non-layered 370
converting concatenated-mirror to mirrored-concatenated 370
converting mirrored-concatenated to concatenated-mirror 370
converting mirrored-stripe to striped-mirror 370
converting striped-mirror to mirrored-stripe 370
creating 299
creating concatenated-mirror 311
creating for use as full-sized instant snapshots 393
creating from snapshots 427
creating mirrored 310
creating mirrored-concatenated 311
creating mirrored-stripe 317
creating RAID-5 319
creating snapshots 423
creating striped 316
creating striped-mirror 318
creating using vxmake 322
creating using vxmake description file 323
creating with version 0 DCOs attached 311
creating with version 20 DCOs attached 315
defined 37
detaching plexes from temporarily 289
disabling FastResync 363
disconnecting plexes 288
displaying information 328
displaying information about snapshots 428
dissociating plexes from 291
dissociating version 0 DCOs from 432
DRL 616
effect of growing on FastResync maps 75
enabling FastResync on 362
enabling FastResync on new 313
excluding storage from use by vxassist 306
finding maximum size of 304
finding out maximum possible growth of 353
flagged as dirty 61
initializing contents to zero 325
initializing using vxassist 324
initializing using vxvol 325
kernel states 331

volumes *(continued)*
layered 48, 54, 298
limit on number of plexes 37
limitations 37
listing on boot (root) disk 136
making immediately available for use 324
maximum number of 559
maximum number of data plexes 548
merging snapshots 426
mirrored 47, 297
mirrored-concatenated 48
mirrored-stripe 47, 297
mirroring across controllers 309, 318
mirroring across targets 307, 318
mirroring all 336
mirroring on disks 337
moving from VM disks 360
moving to improve performance 551
names 37
naming snap 386
obtaining performance statistics 549
performance of mirrored 546
performance of RAID-5 547
performance of striped 546
performing online relayout 363
placing in maintenance mode 335
preparing for DRL and instant snapshot operations 341
preventing recovery on restarting 336
RAID-0 42
RAID-0+1 47
RAID-1 47
RAID-1+0 48
RAID-10 48
RAID-5 50, 297
raw device files 326, 620
reattaching plexes 289
reattaching version 0 DCOs to 432
reconfiguration in clusters 492
recovering after correctable hardware failure 460
removing 359
removing DRL logs 351
removing from /etc/vfstab 359
removing linked snapshots from 408
removing mirrors from 339
removing plexes from 339
removing RAID-5 logs 352
removing sequential DRL logs 351

volumes *(continued)*
removing snapshot mirrors from 408
removing support for DRL and instant snapshots 346
removing version 0 DCOs from 432
resizing 353
resizing using vxassist 355
resizing using vxresize 354
resizing using vxvol 356
restarting moved 259–260, 262
restoring from instant snapshots 411
resynchronizing from snapshots 426
snapshots 65
spanned 40
specifying default layout 305
specifying non-default number of columns 317
specifying non-default relayout 367
specifying non-default stripe unit size 317
specifying storage for version 0 DCO plexes 431
specifying storage for version 20 DCO plexes 342
specifying use of storage to vxassist 305
starting 335
starting using vxassist 324
starting using vxvol 325
states 330
stopping 334
stopping activity on 359
striped 42, 297
striped-mirror 48, 298
striping to improve performance 552
taking multiple snapshots 386
tracing operations 549
trigger point for mirroring in striped-mirror 318
types of layout 296
upgrading to use new features 347
using logs and maps with 298
zeroing out contents of 325
vxassist
adding a log subdisk 277
adding a RAID-5 log 351
adding DCOs to volumes 430
adding DRL logs 349
adding mirrors to volumes 288, 336
adding sequential DRL logs 350
advantages of using 300
changing number of columns 367
changing stripe unit size 367
command usage 301
vxassist *(continued)*
configuring exclusive access to a volume 507
configuring site consistency on volumes 520
converting between layered and non-layered volumes 370
creating cache volumes 391
creating concatenated-mirror volumes 311
creating mirrored volumes 310
creating mirrored-concatenated volumes 311
creating mirrored-stripe volumes 317
creating RAID-5 volumes 320
creating snapshots 421
creating striped volumes 316
creating striped-mirror volumes 318
creating volumes 300
creating volumes for use as full-sized instant snapshots 394
creating volumes with DRL enabled 315
creating volumes with version 0 DCOs attached 313
creating volumes with version 20 DCOs attached 315
defaults file 302
defining layout on specified storage 305
discovering maximum volume size 304
displaying information about snapshots 428
dissociating snapshots from volumes 427
excluding storage from use 306
finding out how much volumes can grow 353
listing tags set on volumes 321, 357
merging snapshots with volumes 426
mirroring across controllers 309, 318
mirroring across enclosures 318
mirroring across targets 307, 309
mirroring file systems on root disk 136
moving DCO log plexes 343
moving DCO plexes 431
moving subdisks after hot-relocation 469
moving volumes 552
relaying out volumes online 363
removing DCOs from volumes 349
removing DRL logs 351
removing mirrors 340
removing plexes 340
removing RAID-5 logs 352
removing tags from volumes 357
removing version 0 DCOs from volumes 432
removing volumes 360
replacing tags set on volumes 357

vxassist *(continued)*
reserving disks 155
resizing volumes 355
resynchronizing volumes from snapshots 426
setting default values 302
setting tags on volumes 321, 357–358
snapabort 375
snapback 376
snapshot 376
snapstart 375
specifing number of mirrors 311
specifying number of RAID-5 logs 320
specifying ordered allocation of storage 307
specifying plexes for online relayout 368
specifying storage attributes 305
specifying storage for version 0 DCO plexes 431
specifying tags for online relayout tasks 368
taking snapshots of multiple volumes 425
unrelocating subdisks after hot-relocation 469
vxcache
listing snapshots in a cache 418
resizing caches 419
starting cache objects 392
stopping a cache 420
tuning cache autogrow 419
vxcached
tuning 418
vxclust 489
vxclustadm 490
vxconfigd
managing with vxdctl 268
monitoring configuration changes 270
operation in clusters 493
vxdarestore
handling simple/nopriv disk failures 110
vxdco
dissociating version 0 DCOs from volumes 432
reattaching version 0 DCOs to volumes 432
removing version 0 DCOs from volumes 432
vxdctl
checking cluster protocol version 508
enabling disks after hot swap 152
managing vxconfigd 268
setting a site tag 515
setting default disk group 219
upgrading cluster protocol version 509
usage in clusters 500
vxdctl enable
invoking device discovery 91
vxddladm
adding disks to DISKS category 102
adding foreign devices 104
changing naming scheme 107
displaying the disk-naming scheme 107
excluding support for disk arrays 100
listing all devices 94
listing configured devices 97
listing configured targets 96
listing excluded disk arrays 100, 102
listing ports on a Host Bus Adapter 95
listing supported disk arrays 99
listing supported disks in DISKS category 100
listing supported HBAs 95
re-including support for disk arrays 100
removing disks from DISKS category 92, 103–104
setting iSCSI parameters 97
vxdg
adding disks to disk groups forcibly 504
changing activation mode on shared disk groups 506
clearing locks on disks 240
configuring site consistency for a disk group 519
configuring site-based allocation for a disk group 518
controllingl CDS compatibility of new disk groups 222
converting shared disk groups to private 505
correcting serial split brain condition 250
creating disk groups 222
creating disk groups with old version number 268
creating shared disk groups 503
deporting disk groups 225
destroying disk groups 263
disabling a disk group 263
displaying boot disk group 219
displaying default disk group 219
displaying disk group version 268
displaying free space in disk groups 221
displaying information about disk groups 220
forcing import of disk groups 240
importing a disk group containing cloned disks 228
importing cloned disks 229
importing disk groups 226
importing shared disk groups 504
joining disk groups 261

vxdg *(continued)*
listing disks with configuration database copies 229
listing objects affected by move 255
listing shared disk groups 502
listing spare disks 463
moving disk groups between systems 238
moving disks between disk groups 237
moving objects between disk groups 257
obtaining copy size of configuration database 217
placing a configuration database on cloned disks 229
reattaching a site 525
recovering destroyed disk groups 264
removing disks from disk groups 223
renaming disk groups 235
setting a site name 516
setting base minor number 242
setting disk connectivity policy in a cluster 507
setting disk group policies 487
setting failure policy in a cluster 507
setting maximum number of devices 243
simulating site failure 525
splitting disk groups 260
upgrading disk group version 268
vxdisk
clearing locks on disks 240
defaults file 87, 113
determining if disks are shared 501
discovering disk access names 112
displaying information about disks 220
displaying multipathing information 174
listing disks 156
listing spare disks 463
listing tags on disks 228
notifying dynamic LUN expansion 138
placing a configuration database on a cloned disk 229
removing tags from disks 230
scanning disk devices 87
setting a site name 517
setting tags on disks 228
updating the disk identifier 227
vxdisk scandisks
rescanning devices 88
scanning devices 88
vxdiskadd
adding disks to disk groups 223
vxdiskadd *(continued)*
creating disk groups 222
placing disks under VxVM control 122
vxdiskadm
Add or initialize one or more disks 113, 222
adding disks 113
adding disks to disk groups 223
Change/display the default disk layout 113
changing disk-naming scheme 107
changing the disk-naming scheme 107
creating disk groups 222
deporting disk groups 225
Disable (offline) a disk device 153
Enable (online) a disk device 152
Enable access to (import) a disk group 226
Encapsulate one or more disks 124
Exclude a disk from hot-relocation use 466
excluding free space on disks from hot-relocation use 466
importing disk groups 226
initializing disks 113
List disk information 156
listing spare disks 463
Make a disk available for hot-relocation use 467
making free space on disks available for hot-relocation use 467
Mark a disk as a spare for a disk group 464
marking disks as spare 464
Mirror volumes on a disk 337
mirroring disks 129
mirroring root disks 132
mirroring volumes 337
Move volumes from a disk 361
moving disk groups between systems 241
moving disks between disk groups 237
moving subdisks after hot-relocation 468
moving subdisks from disks 224
moving volumes from VM disks 361
Remove a disk 142, 224
Remove a disk for replacement 145
Remove access to (deport) a disk group 225
removing disks from pool of hot-relocation spares 465
Replace a failed or removed disk 149
Turn off the spare flag on a disk 465
Unrelocate subdisks back to a disk 468
unrelocating subdisks after hot-relocation 468
vxdiskconfig
purpose of 87

vxdiskunsetup
removing disks from VxVM control 144, 224
vxdmpadm
changing TPD naming scheme 109
configuring an APM 213
configuring I/O throttling 207
configuring response to I/O errors 206, 209
disabling controllers in DMP 172
disabling I/O in DMP 203
discovering disk access names 112
displaying APM information 213
displaying DMP database information 173
displaying DMP node for a path 177, 179
displaying DMP node for an enclosure 177–178
displaying I/O error recovery settings 209
displaying I/O policy 196
displaying I/O throttling settings 209
displaying information about controllers 182
displaying information about enclosures 183
displaying partition size 196
displaying paths controlled by DMP node 180
displaying status of DMP error handling thread 212
displaying status of DMP restoration thread 212
displaying TPD information 184
enabling I/O in DMP 204
gathering I/O statistics 188
listing information about array ports 183
removing an APM 213
renaming enclosures 205
setting I/O policy 198–199
setting path attributes 194
setting restore polling interval 210
specifying DMP path restoration policy 210
stopping DMP restore daemon 212
vxdmpadm list
displaying DMP nodes 178
vxedit
changing plex attributes 292
changing subdisk attributes 278–279
configuring number of configuration copies for a disk group 555
excluding free space on disks from hot-relocation use 465
making free space on disks available for hot-relocation use 466
marking disks as spare 463
removing a cache 420
vxedit *(continued)*
removing disks from pool of hot-relocation spares 464
removing instant snapshots 413
removing plexes 292
removing snapshots from a cache 420
removing subdisks from VxVM 278
removing volumes 360
renaming disks 154
reserving disks 155
vxencap
defaults file 113
VxFS file system resizing 354
vxiod I/O kernel threads 25
vxmake
associating plexes with volumes 288
associating subdisks with new plexes 275
creating cache objects 392
creating plexes 282, 336
creating striped plexes 282
creating subdisks 272
creating volumes 322
using description file with 323
vxmend
putting plexes online 335
re-enabling plexes 289
taking plexes offline 288, 335
vxmirror
configuring VxVM default behavior 336
mirroring root disks 132
mirroring volumes 336
vxnotify
monitoring configuration changes 270
vxplex
adding RAID-5 logs 351
attaching plexes to volumes 287, 336
converting plexes to snapshots 424
copying plexes 291
detaching plexes temporarily 289
dissociating and removing plexes 291
dissociating plexes from volumes 292
moving plexes 290
reattaching plexes 289
removing mirrors 340
removing mirrors of root disk volumes 138
removing plexes 340
removing RAID-5 logs 352
vxprint
checking if FastResync is enabled 362

vxprint *(continued)*
determining if DRL is enabled 345
displaying DCO information 342, 431
displaying plex information 283
displaying snapshots configured on a cache 420
displaying subdisk information 272
displaying volume information 328
enclosure-based disk names 112
identifying RAID-5 log plexes 352
listing spare disks 463
listing volumes on boot disks 136
used with enclosure-based disk names 112
verifying if volumes are prepared for instant snapshots 390
viewing base minor number 242
vxrecover
preventing recovery 336
recovering plexes 460
restarting moved volumes 259–260, 262
restarting volumes 336
vxrelayout
resuming online relayout 369
reversing direction of online relayout 369
viewing status of online relayout 368
vxrelocd
hot-relocation daemon 456
modifying behavior of 473
notifying users other than root 474
operation of 457
preventing from running 474
reducing performance impact of recovery 474
vxresize
growing volumes and file systems 354
limitations 354
shrinking volumes and file systems 354
vxsd
adding log subdisks 277
adding subdisks to RAID-5 plexes 276
adding subdisks to striped plexes 276
associating subdisks with existing plexes 275
dissociating subdisks 277
filling in sparse plexes 276
joining subdisks 274
moving subdisk contents 273
removing subdisks from VxVM 278
splitting subdisks 274
vxse
Storage Expert 529
vxse_dc_failures
rule to check for hardware failures 539
vxse_dg1
rule to check for full disk group configuration database 535
vxse_dg2
rule to check disk group configuration copies 536
vxse_dg3
rule to check on disk config size 536
vxse_dg4
rule to check disk group version number 536
vxse_dg5
rule to check number of configuration copies in disk group 536
vxse_dg6
rule to check for non-imported disk groups 536
vxse_disk
rule to check for initialized disks 537
vxse_disklog
rule to check for multiple RAID-5 logs on a disk 534
vxse_drl1
rule to check for mirrored volumes without a DRL 534
vxse_drl2
rule to check for mirrored DRL 534
vxse_host
rule to check system name 539
vxse_mirstripe
rule to check mirrored-stripe volumes 538
vxse_raid5
rule to check number of RAID-5 columns 538
vxse_raid5log1
rule to check for RAID-5 log 535
vxse_raid5log2
rule to check RAID-5 log size 535
vxse_raid5log3
rule to check for non-mirrored RAID-5 log 535
vxse_redundancy
rule to check volume redundancy 537
vxse_rootmir
rule to check rootability 539
vxse_spares
rule to check number of spare disks 539
vxse_stripes1
rule to check stripe unit size 538
vxse_stripes2
rule to check number of stripe columns 538

vxse_volplex
rule to check plex and volume states 537
vxsnap
adding snapshot mirrors to volumes 407
administering instant snapshots 377
backing up multiple volumes 404
controlling instant snapshot synchronization 416
creating a cascaded snapshot hierarchy 408
creating full-sized instant snapshots 397, 403
creating linked break-off snapshot volumes 403
creating space-optimized instant snapshots 395
displaying information about instant snapshots 414
dissociating instant snapshots 412
preparing volumes for DRL and instant snapshots operations 341
preparing volumes for instant snapshots 390
reattaching instant snapshots 410
reattaching linked third-mirror snapshots 411
refreshing instant snapshots 409
removing a snapshot mirror from a volume 408
removing support for DRL and instant snapshots 346
restore 378
restoring volumes 411
splitting snapshot hierarchies 413
vxspcshow
discovering device names 79
vxsplitlines
diagnosing serial split brain condition 249
vxstat
determining which disks have failed 460
obtaining disk performance statistics 551
obtaining volume performance statistics 549
usage with clusters 510
zeroing counters 551
vxtask
aborting tasks 334
listing tasks 333
monitoring online relayout 369
monitoring tasks 334
pausing online relayout 369
resuming online relayout 369
resuming tasks 334
vxtrace
tracing volume operations 549
vxtune
setting volpagemod_max_memsz 564
vxunreloc
listing original disks of hot-relocated subdisks 471
moving subdisks after hot-relocation 470
restarting after errors 472
specifying different offsets for unrelocated subdisks 471
unrelocating subdisks after hot-relocation 470
unrelocating subdisks to different disks 470
vxunroot
removing rootability 138
unencapsulating the root disk 138
VxVM
benefits to performance 545
cluster functionality (CVM) 475
configuration daemon 268
configuring to create mirrored volumes 336
dependency on operating system 25
disk discovery 87, 89
granularity of memory allocation by 561
limitations of shared disk groups 487
maximum number of data plexes per volume 548
maximum number of subdisks per plex 561
maximum number of volumes 559
maximum size of memory pool 562
migrating from SDS 623
minimum size of memory pool 564
objects in 31
operation in clusters 477
performance tuning 554
removing disks from 224
removing disks from control of 144
rootability 128
shared objects in cluster 480
size units 272
task monitor 332
types of volume layout 296
upgrading 264
upgrading disk group version 268
VXVM_DEFAULTDG environment variable 218
vxvol
configuring exclusive access to a volume 508
configuring site consistency on volumes 520
disabling DRL 346
disabling FastResync 363
enabling FastResync 362
initializing volumes 325
putting volumes in maintenance mode 335

vxvol *(continued)*
re-enabling DRL 346
resizing logs 357
resizing volumes 356
restarting moved volumes 259–260, 262
setting read policy 358
starting volumes 325, 335
stopping volumes 334, 360
zeroing out volumes 325
vxvset
adding volumes to volume sets 437
controlling access to raw device nodes 441
creating volume sets 436
creating volume sets with raw device access 440
listing details of volume sets 437
removing volumes from volume sets 439
starting volume sets 438
stopping volume sets 438

W

warning messages
Specified region-size is larger than the limit on the system 388
worldwide name identifiers 83, 106
WWN identifiers 83, 106

Z

zero
setting volume contents to 325