



XVM Volume Manager Administrator Guide

007-4003-028

COPYRIGHT

© 1999-2011, 2012, SGI. All rights reserved; provided portions may be copyright in third parties, as indicated elsewhere herein. No permission is granted to copy, distribute, or create derivative works from the contents of this electronic documentation in any manner, in whole or in part, without the prior written permission of SGI.

LIMITED RIGHTS LEGEND

The software described in this document is "commercial computer software" provided with restricted rights (except as to included open/free source) as specified in the FAR 52.227-19 and/or the DFAR 227.7202, or successive sections. Use beyond license provisions is a violation of worldwide intellectual property laws, treaties and conventions. This document is provided with limited rights as defined in 52.227-14.

TRADEMARKS AND ATTRIBUTIONS

CXFS, Performance Co-Pilot, SGI, the SGI logo, SGI UV, Supportfolio, and XFS are trademarks or registered trademarks of Silicon Graphics International Corp. or its subsidiaries in the United States and other countries.

Brocade is a trademark of Brocade Communications Systems, Inc. FLEXlm is a trademark of GLOBEtrotter, Inc. Java is a registered trademark of Sun Microsystems, Inc. Linux is a registered trademark of Linus Torvalds in several countries. Novell and SUSE are registered trademarks of Novell, Inc., in the United States and other countries. Red Hat and all Red Hat-based trademarks are trademarks or registered trademarks of Red Hat, Inc. in the United States and other countries.

All other trademarks mentioned herein are the property of their respective owners.

What's New in This Guide

This revision of the *XVM Volume Manager Administrator Guide* supports the SGI InfiniteStorage Software Platform (ISSP) 2.6 release.

Major Documentation Changes

- Updated information in “Path Manager” on page 32
- Updated information about running the `fsck` command in “Limitations and Considerations” on page 32.
- Added “Using the Appropriate Affinity Values in `/etc/failover2.conf` for XVM Device” on page 96
- Updated affinity information in “Example `failover2.conf` Files” on page 96.

Record of Revision

Version	Description
001	October 1999 Original printing. Supports the IRIX 6.5.6 release
002	October 1999 Supports the IRIX 6.5.6 release
003	January 2000 Supports the IRIX 6.5.7 release
004	April 2000 Supports the IRIX 6.5.8 release
005	September 2000 Supports the IRIX 6.5.10 release
006	April 2001 Supports the IRIX 6.5.12 release
007	July 2001 Supports the IRIX 6.5.13 release
008	January 2002 Supports the IRIX 6.5.15 release
009	April 2002 Supports the IRIX 6.5.16 release

011	July 2002 Supports the IRIX 6.5.17 release (there was no 010 version due to an internal numbering mechanism)
012	October 2002 Supports the IRIX 6.5.18 release
013	January 2003 Supports the IRIX 6.5.19 release and SGI ProPack v2.1 for Linux
014	April 2003 Supports the IRIX 6.5.20 release and SGI ProPack v2.2 for Linux
015	July 2003 Supports the IRIX 6.5.21 release and SGI ProPack v2.2 for Linux
016	October 2003 Supports the IRIX 6.5.22 release and SGI ProPack v2.3 for Linux
017	April 2004 Supports the IRIX 6.5.24 release and SGI ProPack v3.0 for Linux
018	February 2005 Supports the IRIX 6.5.27 release and SGI ProPack v3.0 for Linux
019	November 2006 Supports the IRIX 6.5.30 and SGI ProPack 5 for Linux releases
020	March 2007 Supports the IRIX 6.5.30 and SGI ProPack 5 for Linux Service Pack 1 releases
021	September 2007 Supports the IRIX 6.5.30 and SGI ProPack 5 for Linux Service Pack 3 releases
022	March 2008 Supports the IRIX 6.5.30 and SGI ProPack 5 for Linux Service Pack 5 releases

-
- 023 April 2009
Supports the IRIX 6.5.30 and SGI ProPack 6 for Linux Service Pack 3 releases
- 024 July 2009
Supports the IRIX 6.5.30 and SGI ProPack 6 for Linux Service Pack 4 releases
- 025 February 2011
Supports the SGI Foundation 2.3 release
- 026 May 2011
Supports the XVM product in the SGI InfiniteStorage Software Platform (ISSP) 2.4 release.
- 027 November 2011
Supports the XVM product in the SGI InfiniteStorage Software Platform (ISSP) 2.5 release.
- 028 May 2012
Supports the XVM product in the SGI InfiniteStorage Software Platform (ISSP) 2.6 release.

Contents

What's New in This Guide	iii
Record of Revision	v
Figures	xix
Tables	xxiii
About This Guide.	xxv
Related Documentation	xxvi
Conventions Used in This Guide	xxvi
Reader Comments.	xxvii
1. Introduction to the XVM Volume Manager	1
XVM Volume Manager Features.	2
Partition Layout under XVM.	4
XVM Partition Layout with SGI/DVH Disk Format	4
XVM Partition Layout with GPT Disk Format	8
Composition of XVM Logical Volumes.	16
Volumes	20
Subvolumes	22
Slices	23
Concat.	23
Stripes	25
Mirrors.	27
Writing Data to Logical Volumes	29
XVM Logical Volumes in a CXFS Cluster	31
XVM Logical Volumes and Failover.	31
Installing the XVM Volume Manager	32
Path Manager	32
Limitations and Considerations	32

. 32
2. XVM Administration Concepts 33
XVM Objects 33
XVM Domains 34
Physical Disk Administration 37
Creating Physical Volumes 37
Managing Physical Volumes 38
Displaying Physical Volumes 39
Changing the Domain of a Physical Volume 39
Adding a Physical Volume to Running System 39
Replacing a Physical Volume 39
Renaming a Physical Volume 40
Physical Volume Statistics 40
Destroying Physical Volumes 40
Creating Logical Resources 40
Creating Topologies 41
Automatic Volume and Subvolume Creation 41
Volume Element Naming 41
Attaching Volume Elements 42
Detaching Volume Elements 43
Empty Volume Elements 44
Logical Volume Statistics 44
Creating Slices 44
Creating Concats 45
Creating Stripes 45
Creating Mirrors 45
Read Policies 46
Primary Leg 47
The -clean Mirror Creation Option 48
The -norevive Mirror Creation Option 48
Creating Volumes 48
Creating Subvolumes 48
Reorganizing Logical Volumes 49

Managing Logical Resources	49
Displaying Volume Elements	50
Disabling Volume Elements	52
Bringing a Volume Element Online.	52
Making Online Changes	52
Saving and Regenerating XVM Configurations	52
Destroying Logical Resources	53
Deleting Volume Elements	53
Removing Configuration Information for Inaccessible Disks	54
3. The XVM Command Line Interface	55
Using the XVM CLI	55
Online Help for XVM CLI Commands	57
XVM CLI Syntax	58
Object Names in XVM.	59
XVM Object Specification	59
Piece Syntax	61
XVM Object Name Examples	63
Regular Expressions	63
XVM Device Directories and Pathnames	64
Command Output and Redirection	64
Safe Versus Unsafe Commands	65
4. XVM Administration Commands	67
Physical Volume Commands.	67
Changing the Current Domain with the set Command	68
Assigning Disks to the XVM Volume Manager with the label Command	68
Displaying Physical Volumes with the show Command.	69
Modifying Physical Volumes with the change Command	73
Probing a Physical Volume with the probe Command	73
Regenerating XVM Physical Volumes using the dump command	73
Removing Disks from the XVM Volume Manager with the unlabel Command	73

Logical Volume Commands 74
Creating Volume Elements 74
The slice Command 74
The concat Command 75
The mirror Command 75
The stripe Command 77
The subvolume Command 78
The volume Command 79
Modifying Volume Elements. 79
The change Command 79
The attach Command. 80
The detach Command 81
The remake Command 81
Modifying Volume Elements on a Running System 82
The insert command 82
The collapse Command 83
Displaying Volume Elements: Using the show Command 85
Reconstructing Volume Elements: Using the dump Command 86
Deleting Volume Elements: Using the delete Command 86
Removing Configuration Information: Using the reprobe Command 87
5. XVM Failover 91
Selecting a Failover Version 92
Failover V1 93

Failover V2	94
The failover2.conf File	95
Using the Appropriate Affinity Values in /etc/failover2.conf for XVM Device	96
Example failover2.conf Files.	96
Parsing the failover2.conf File	98
Switching physvol Path Interactively	99
Returning to a preferred path	99
Switching to a new device	99
Setting a new affinity	100
Switching paths for all nodes in a cluster	100
Automatic Probe after Labeling a Device	100
How to Create a failover2.conf File	101
Create an initial /etc/failover2.conf file	101
Set affinity for each path in /etc/failover2.conf	101
Set the preferred path for each physvol	102
Initialize the XVM configuration in the kernel	102
Set all LUNs to their preferred path	102
Sample /etc/failover2.conf file	103
6. XVM Administration Procedures	105
Before You Begin	105
Preparing to Configure XVM Volumes under Linux	107
Creating a Logical Volume with a Three-Way Stripe	108
Striping a Portion of a Disk	112
Creating a Logical Volume with a Data and Log Subvolume	116
Creating a Logical Volume with a Data and Log Subvolume	118
Creating a Volume from the Top Down	121
Creating an XVM Logical Volume with Striped Mirrors	123
Giving Away a System Disk from the Miniroot	126
Online Reconfiguration Using Mirroring	127

Online Modification of a Logical Volume	133
Creating the Logical Volume	134
Growing the Logical Volume.	135
Mirroring Data on the Logical Volume	137
Converting a Concat to a Stripe Using Mirroring	139
Removing a Mirror	141
Mirroring Individual Stripe Members	142
Making an XVM Volume Using a GPT Label	143
Making a GPT Label	144
Example Using the parted Command	145
Making the Partitions	146
Making the XVM Label and Slices	147
Converting an SGI DVH XVM Label to a GPT Label Suitable for XVM	149
Tuning XVM	155
xvm_mr_daemon_max Variable	155
7. Statistics.	157
Physical Volume Statistics.	158
Subvolume Statistics	158
Stripe Statistics	158
Concat Statistics	160
Mirror Statistic	161
Slice Statistics	162
8. XVM Volume Manager Operation	163
Cluster System Startup.	163
Mirror Revives	164
Mirror Revives on Recovery in a Cluster	165
XVM Mirror Revive Resources	165
Modifying Mirror Revive Resources	166
XVM Subsystem Parameters	167
9. The XVM Manager GUI	169
Installing the XVM Manager GUI	170

Starting the XVM Manager GUI171
Starting the GUI on Linux171
Starting the GUI on a PC171
Logging In171
The XVM Manager GUI Window172
Configuring the System178
Selecting Items to View or Modify181
Configuring the System Quickly.182
Analyzing I/O Performance183
Using Drag-and-Drop for XVM Configuration.184
Structuring Volume Topologies.184
Configuring Disks184
Drag-and-Drop Restrictions185
Viewing Log Messages185
Important GUI and CLI Differences.185
Disks Tasks.186
Label Disks186
Slice Disks188
Rename a Disk189
Remove XVM Labels from Disks189
Modify Statistics Collection on Disks190
Give Away Ownership of Disks.190
Steal a Foreign Disk191
Probe Disks for Labels193
Dump Volume Element or Physical Volume Structure to File193

Volume Element Tasks.	194
Create a Concat	194
Create a Mirror.	195
Create a Stripe	196
Delete Volume Elements	198
Rename a Volume Element	199
Insert Mirrors or Concats above Volume Elements	199
Remove Unneeded Mirrors and Concats	200
Enable Volume Elements	200
Disable Volume Elements	200
Bring Volume Elements Online	201
Remake a Volume Element	201
Detach Volume Elements	202
Create a Volume	202
Create a Subvolume	203
Modify Subvolumes	204
Modify Statistics Collection on Volume Elements	205
Filesystem Tasks	205
Make Filesystems	205
Grow a Filesystem.	207
Mount a Filesystem Locally	208
Unmount a Locally Mounted Filesystem	208
Remove Filesystem Mount Information	209
Privileges Tasks.	209
Grant Task Access to a User or Users	209
Granting Access to a Few Tasks.	210
Granting Access to Most Tasks	211
Revoke Task Access from a User or Users	211
A. XVM and XLV Logical Volumes.	213
XVM and XLV Logical Volume Creation Comparison.	213
Upgrading from XLV to XVM Logical Volumes	215
Converting XLV Mirrored Stripes to XVM Striped Mirrors	216

Index.	221
-------------------------	-----

Figures

Figure 1-1	XVM Option Disk Partition Layout on SGI/DVH Disk	5
Figure 1-2	XVM DiskPartition Layout with Combined root and usr Filesystems	6
Figure 1-3	XVM Disk Partition Layout with Separate root and usr Filesystems .	7
Figure 1-4	Partition Layout of System Disk with Multiple Root Filesystems. .	8
Figure 1-5	GPT Disk Layout for XVM	10
Figure 1-6	GPT Disk Layout for XVM Using Two Partitions	11
Figure 1-7	Basic XVM Striped Logical Volume	17
Figure 1-8	XVM Logical Volume with Mirrored Stripe and Three Subvolumes .	18
Figure 1-9	XVM Logical Volume after Insertion of Concat.	19
Figure 1-10	XVM Volume with System-Defined Subvolume Types	21
Figure 1-11	XVM Volume with User-Defined Subvolume Types	22
Figure 1-12	XVM Subvolume Examples	23
Figure 1-13	Concat Composed of Two Slices	24
Figure 1-14	Concat Composed of Two Mirrors	25
Figure 1-15	Three-Way Stripe	26
Figure 1-16	Stripe on Stripe Volume Element	27
Figure 1-17	Mirror Composed of Two Slices	28
Figure 1-18	Mirror Composed of Two Stripes.	28
Figure 1-19	Mirror Composed of a Stripe and a Concat	29
Figure 1-20	Writing Data to a Non-Striped Logical Volume.	30
Figure 1-21	Writing Data to a Striped Logical Volume	30
Figure 2-1	XVM Physical Volume in Local Domain.	35
Figure 2-2	XVM Physical Volume in Cluster Domain	36
Figure 2-3	Reading Data from a Mirror with a Round-Robin Read Policy . . .	47
Figure 2-4	Reading Data from a Mirror with a Sequential Read Policy . . .	47
Figure 3-1	XVM Logical Volume with System-Generated Names.	62
Figure 4-1	XVM Logical Volumes for Root and Swap	87
Figure 4-2	XVM Mirrored Root Physvol	88

Figure 4-3	Root and Swap Mirrored Logical Volumes	89
Figure 6-1	XVM Logical Volume with Three-Way Stripe	109
Figure 6-2	Striping a Portion of a Disk	112
Figure 6-3	XVM Logical Volume with a Log Subvolume	116
Figure 6-4	Logical Volume with Data and Log Subvolumes	118
Figure 6-5	XVM Logical Volume with Striped Mirrors	124
Figure 6-6	Original Online Filesystem.	127
Figure 6-7	Filesystem after Insertion of Mirror	129
Figure 6-8	Filesystem after Attaching Stripe to Mirror	130
Figure 6-9	Filesystem after Detaching Original Slice.	131
Figure 6-10	Reconfigured Filesystem	132
Figure 6-11	Original XVM Logical Volume	134
Figure 6-12	XVM Logical Volume after Insert	135
Figure 6-13	XVM Logical Volume After Mirroring	137
Figure 6-14	XVM Logical Volume after Conversion from Concat to Mirror	139
Figure 6-15	XVM Logical Volume after Mirror Removal	141
Figure 6-16	XVM Logical Volume after Mirroring Slices	142
Figure 6-17	Creating XVM Volumes and a GPT Label on a LUN	144
Figure 9-1	XVM Manager GUI Window	173
Figure 9-2	XVM Manager GUI Window with Item Selected	175
Figure 9-3	Create a Concat Task Window.	179
Figure 9-4	XVM Manager GUI Label Disks Task.	180
Figure 9-5	XVM Manager GUI Browse Window	181

Tables

Table 3-1	Prefixes Specifying Object Type	60
Table 3-2	Specifying Logical Volume Elements Using Piece Syntax	62
Table 6-1	XVM DVH disk layout.149
Table 6-2	XVM GPT disk layout149
Table 6-3	XVM GPT disk layout example 1150
Table 6-4	XVM DVH disk layout example 1151
Table 6-5	XVM GPT disk layout example 2151
Table 9-1	XVM GUI rpms under Linux170
Table 9-2	Key to XVM Manager GUI Icons176
Table 9-3	Key to XVM Manager GUI States177
Table 9-4	XVM Manager GUI Command Buttons182
Table A-1	XVM and XLV Logical Volume Creation214

About This Guide

XVM Volume Manager Administrator Guide describes the configuration and administration of XVM logical volumes using the XVM Volume Manager.

This guide contains the following information:

- Chapter 1, “Introduction to the XVM Volume Manager,” describes the features of the XVM Volume Manager and provides an introduction to the components of an XVM logical volume. It also provides instructions for installing XVM as a standalone volume manager.
- Chapter 2, “XVM Administration Concepts,” describes the concepts that underlie the administration commands.
- Chapter 3, “The XVM Command Line Interface,” describes the operation of the XVM command line interface and the features it provides.
- Chapter 4, “XVM Administration Commands,” summarizes the XVM commands and provides examples of each command. It also provides information about configuring XVM system disks.
- Chapter 5, “XVM Failover,” provides an overview of failover configuration under XVM.
- Chapter 6, “XVM Administration Procedures,” provides examples of many common XVM administration procedures.
- Chapter 7, “Statistics,” provides examples of the statistics that XVM maintains for the components of XVM logical volumes.
- Chapter 8, “XVM Volume Manager Operation,” describes various aspects of the way the XVM Volume Manager operates.
- Chapter 9, “The XVM Manager GUI,” describes the installation and operation of the XVM Manager graphical user interface (GUI).
- Appendix A, “XVM and XLV Logical Volumes,” provides a side-by-side comparison of XVM and XLV logical volume configuration and provides

procedures for converting an existing XLV logical volume configuration to an XVM configuration.

Related Documentation

The following documents may contain additional information required to use this product:

- *CXFS 6 Administration Guide for SGI InfiniteStorage*
- *CXFS 6 Client-Only Guide for SGI InfiniteStorage*
- *SGI Performance Suite 1.4 Start Here*
- *SGI Foundation Software 2.6 Start Here*

Conventions Used in This Guide

These type conventions and symbols are used in this guide:

The following conventions are used throughout this publication:

Convention	Meaning
<code>command</code>	This fixed-space font denotes literal items such as commands, files, routines, path names, signals, messages, and programming language structures.
<i>variable</i>	Italic typeface denotes variable entries and words or concepts being defined.
user input	This bold, fixed-space font denotes literal items that the user enters in interactive sessions. (Output is shown in nonbold, fixed-space font.)
<code>manpage(x)</code>	Man page section identifiers appear in parentheses after man page names.
GUI element	This font denotes the names of graphical user interface (GUI) elements such as windows, screens, dialog boxes, menus, toolbars, icons, buttons, boxes, fields, and lists.

Reader Comments

If you have comments about the technical accuracy, content, or organization of this document, please tell us. Be sure to include the title and part number of the document with your comments. (Online, the document number is located in the front matter of the manual. In printed manuals, the document number can be found on the back cover.)

You can contact us in any of the following ways:

- Send e-mail to the following address:
techpubs@sgi.com
- Contact your customer service representative and ask that an incident be filed in the SGI incident tracking system.

- Send mail to the following address:

Technical Publications
SGI
46600 Landing Parkway
Fremont, CA 94538

We value your comments and will respond to them promptly.

Introduction to the XVM Volume Manager

The XVM Volume Manager provides a logical organization to disk storage that enables an administrator to combine underlying physical disk storage into a single logical unit, known as a *logical volume*. Logical volumes behave like standard disk partitions and can be used as arguments anywhere a partition can be specified.

A logical volume allows a filesystem or raw device to be larger than the size of a physical disk. Using logical volumes can also increase disk I/O performance because a volume can be striped across more than one disk. Logical volumes can also be used to mirror data on different disks.

Note: The XVM Volume Manager can be used in a clustered environment with CXFS filesystems. For information on CXFS filesystems, see *CXFS 6 Administration Guide for SGI InfiniteStorage*.

This chapter provides an overview of the XVM Volume Manager and includes sections on the following topics:

- “XVM Volume Manager Features” on page 2
- “Partition Layout under XVM” on page 4
- “Composition of XVM Logical Volumes” on page 16
- “Writing Data to Logical Volumes” on page 29
- “XVM Logical Volumes in a CXFS Cluster” on page 31
- “XVM Logical Volumes and Failover” on page 31
- “Installing the XVM Volume Manager” on page 32
- “Path Manager” on page 32

- “Limitations and Considerations” on page 32

XVM Volume Manager Features

The XVM Volume Manager provides all of the basic features of logical volumes that were provided with XLV logical volumes, an older logical volume design developed at SGI. These features include the following:

- Self-identifying volumes

Persistent configuration and attribute information for a logical volume is distributed among all disks that are part of the logical volume. The information is stored in a label file on a disk, removing any dependence on the filesystem. Whole sets of disks can be moved within and between systems.

- Multiple storage types

Logical volumes support aggregate storage through concatenation and striping. Logical volumes also support redundant storage through mirroring.

- Multiple address spaces

A logical volume can support multiple mutually exclusive address spaces in the form of *subvolumes*. Each subvolume within a logical volume has a different usage defined by the application accessing the data. The XVM Volume Manager supports a *log subvolume* for separating filesystem meta-data from the data itself, and a *data subvolume* where most data, including user files, resides.

- Path failover

When a host performs an I/O operation and it fails due to connection problems, the XVM volume manager will automatically try all paths that the host has to the disk by switching (failing over) from a failing path that is being used to one of the alternate paths.

- Online configuration changes

The XVM Volume Manager allows an administrator to perform certain volume reconfigurations without taking the volume offline. Volume reconfigurations that can be performed online include increasing the size of a concatenated volume and adding or removing a piece of a mirror.

In addition to the features that XLV logical volumes provide, the XVM Volume Manager provides the following significant features:

- Support for a cluster environment

The XVM Volume Manager supports a cluster environment, providing an image of the XVM devices across all nodes in a cluster and allowing for administration of XVM devices from any node in the cluster. Disks within a cluster can be assigned dynamically to the entire cluster or to individual nodes within the cluster, as local volumes.

- Flexible volume layering and configuration

The elements that make up an XVM logical volume can be layered in any configuration. For example, using the XVM Volume Manager, an administrator can mirror disks at any level of the logical volume configuration, or use stripe-on-stripe layering rather than a simple striped volume in situations where this results in better volume throughput.

- Support for a graphical user interface

XVM Manager Graphical User Interface (GUI) provides access to the tasks that help you set up and administer your logical volumes and provides icons representing status and structure.

Note: The XVM GUI is not supported on the Red Hat Enterprise Linux Server.

- Large number of slices

The layout of a disk under XVM is independent of the underlying device driver. The XVM Volume Manager determines how the disk is sliced. Because of this, the XVM Volume Manager can divide a disk into an arbitrary number of slices.

- Large number of volumes

The XVM Volume Manager supports thousands of volumes on a single disk and allows for the expansion of the label file as needed. Under XVM, there are no restrictions on volume width, which is the number of volume elements that make up the widest layer of a volume.

- Improved mirror performance

The XVM Volume Manager allows you to specify the read policy for an XVM mirror element, allowing you to read from the mirror in a sequential or round-robin fashion, depending on the needs of your configuration. You can also specify whether a particular leg of a mirror is to be preferred for reading.

The XVM Volume Manager also allows you to specify when a mirror does not need to be synchronized at creation, and to specify that a particular mirror, such as a mirror of a scratch filesystem, will never need to be synchronized.

- Built-in statistics support

The XVM Volume Manager tracks statistics at every level of the volume tree and provides type-specific statistics. Statistics are tracked per host and interfaces are provided to Performance Co-Pilot to present a global state.

- Device hot plug

A disk containing XVM logical volumes can be added to a running system and the system will be able to read the XVM configuration information without rebooting. This feature allows you to move disks between systems and to configure a new system from existing disks that contain XVM logical volumes.

- Insertion and removal

The XVM administration commands provide the ability to insert and remove components from existing disk configurations, allowing you to grow and modify a disk configuration on a running system with open volumes.

Partition Layout under XVM

Before you create XVM logical volumes on a disk, you must label the disk as an XVM disk. In order to label a disk as an XVM disk, the disk must be formatted.

- For information on formatting a disk as an SGI disk under Linux, see “Preparing to Configure XVM Volumes under Linux” on page 107.
- For information on using GPT disks with XVM, see “XVM Partition Layout with GPT Disk Format” on page 8.

The partition layout of a disk under XVM is described in the following sections.

XVM Partition Layout with SGI/DVH Disk Format

When using SGI/DVH disks, the XVM Volume Manager controls the partitioning of an XVM disk. Partitions are not used to define the storage available for XVM slices, as they are for XLV logical volumes.

When you label an SGI/DVH disk as an XVM disk, you can specify whether the disk will be an XVM option disk, an XVM system disk with combined root and `usr` filesystems, or an XVM system disk with separate root and `usr` filesystems. An XVM disk is labeled as an option disk by default.

Figure 1-1 shows the partition layout of an XVM option disk with SGI/DVH disk format. Partition 10 contains the entire disk and partition 8 contains the volume header. The remainder of the disk that is not part of partition 8 is divided into slices that you specify using the XVM Volume Manager.

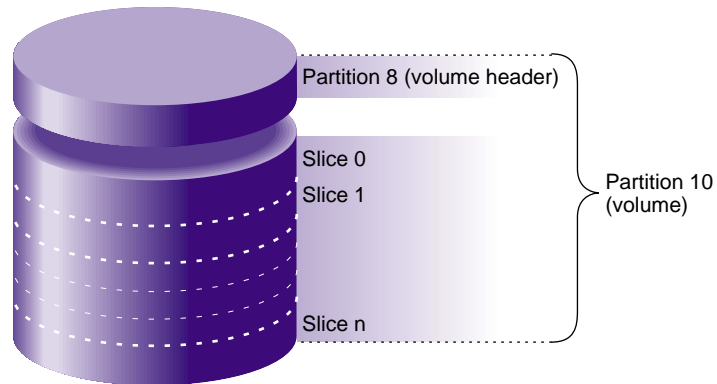


Figure 1-1 XVM Option Disk Partition Layout on SGI/DVH Disk

Figure 1-2 shows the partition layout of an XVM system disk with combined root and `usr` filesystems. Partition 8 contains the volume header, partition 9 contains the XVM label area where the information about the XVM volume elements on a disk is stored, partition 0 contains the root partition, and partition 1 contains the swap partition. Partition 10 contains the entire disk.

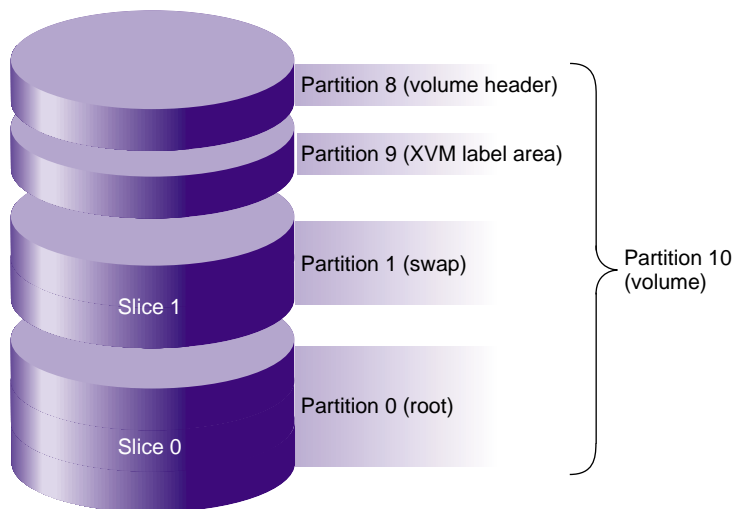


Figure 1-2 XVM DiskPartition Layout with Combined root and `usr` Filesystems

Figure 1-3 shows the partition layout of an XVM system disk with separate root and `usr` filesystems. Partition 8 contains the volume header, partition 9 contains the XVM label area where the information about the XVM volume elements on a disk is stored, partition 0 contains the root partition, partition 1 contains the swap partition, and partition 6 contains the `usr` partition. Partition 10 contains the entire disk. In this illustration the XVM system disk includes space on the disk that can be used for other filesystems besides root, swap and `usr`.

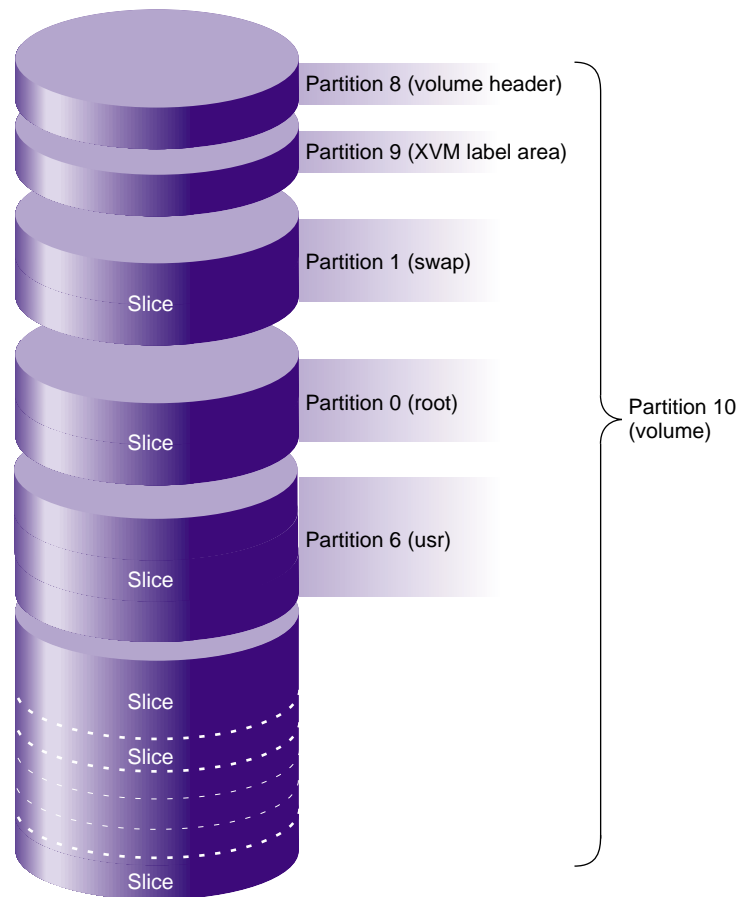


Figure 1-3 XVM Disk Partition Layout with Separate root and `usr` Filesystems

Figure 1-4 shows the partition layout of an XVM system disk with multiple root filesystems as well as a separate `usr` filesystem. Partition 8 contains the volume header,

partition 9 contains the XVM label area where the information about the XVM volume elements on a disk is stored, partition 0 contains the first root partition, partition 1 contains the swap partition, partitions 2 and 3 contain additional root filesystems, and partition 6 contains the `usr` partition. Partition 10 contains the entire disk.

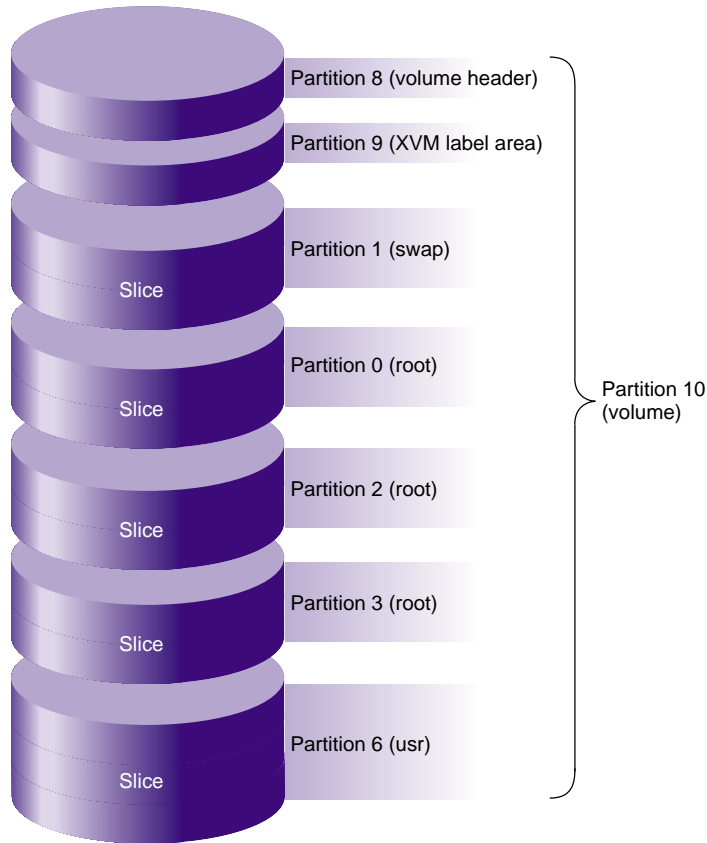


Figure 1-4 Partition Layout of System Disk with Multiple Root Filesystems

XVM Partition Layout with GPT Disk Format

The XVM logical volume manager can be used with GUID partition table (GPT) disks. As with SGI/DVH disks, you must label a GPT disk as an XVM volume before you can create XVM logical volumes on the disk.

For information on labeling a disk as an XVM disk, see “Creating Physical Volumes” on page 37, and “Assigning Disks to the XVM Volume Manager with the label Command” on page 68.

Note: You cannot label a GPT disk as an XVM system disk.

You can create these labels on SGI Performance Suite server-capable nodes and Linux third-party clients. The GPT label puts header data in sector 1 of a LUN, leaving sector 0 for a master boot record. Partition information is stored in a variable number of sectors, starting at sector 2.

In order to use a GPT disk for XVM logical volumes, the GPT disk must be formatted with one or two partitions:

If the disk is formatted with one partition, that partition must start before block 64. It is used for both the XVM metadata and the user’s data. The partition typically includes all the space on the disk not used by the GPT label.

If two partitions are used, follow these guidelines:

- The first partition must start before block 64 and must be at least 512KB in size. The XVM metadata (the label and slice information) will be placed in this partition.
- The second partition is typically the remainder of the disk. This is the space that XVM will slice up and use for data.

There is no advantage to using two partitions; two partitions are supported for backward compatibility.

Figure 1-5 shows a GPT disk that is formatted for XVM.

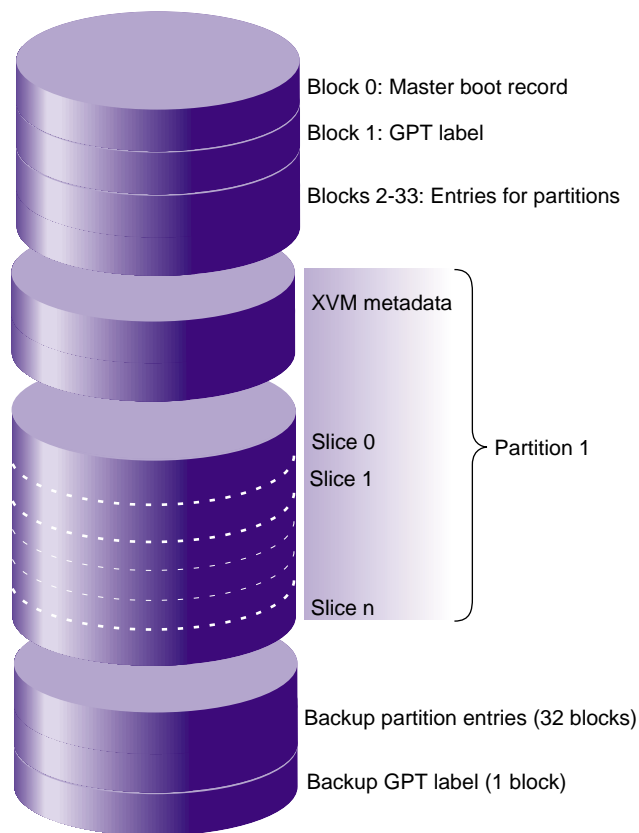


Figure 1-5 GPT Disk Layout for XVM

Figure 1-6 shows GPT disk layout for XVM using two partitions:

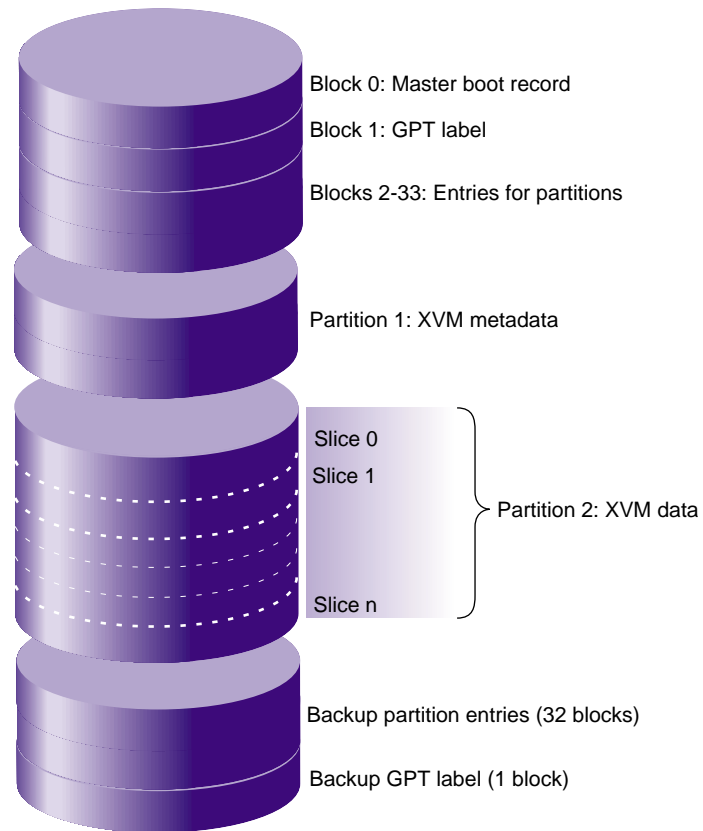


Figure 1-6 GPT Disk Layout for XVM Using Two Partitions

The `parted(8)` command is a Linux disk partitioning and partition resizing program. The following examples show how to create a GPT label on an SGI Altix system using the `parted` command, followed by an example using the `xvm label` command.

To use the parted command to create a GPT label, perform the following:

```
> parted /dev/xscsi/pci0002:00:02.1/node20000011c61dd850/port1/lun0/disc
```

```
GNU Parted 1.6.25.1
```

```
Copyright (C) 1998 - 2005 Free Software Foundation, Inc.
```

```
This program is free software, covered by the GNU General Public License.
```

```
This program is distributed in the hope that it will be useful, but WITHOUT ANY  
WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A  
PARTICULAR PURPOSE. See the GNU General Public License for more details.
```

```
Using /dev/sdae
```

```
(parted) mklabel gpt
```

```
(parted) print
```

```
Disk geometry for /dev/sdae: 0kB - 37GB
```

```
Disk label type: gpt
```

Number	Start	End	Size	File system	Name	Flags
--------	-------	-----	------	-------------	------	-------

```
(parted) mkpart PRIMARY XFS 0 37GB
```

```
(parted) unit s
```

```
(parted) print
```

```
Disk geometry for /dev/sdae: 0s - 71687371s
```

```
Disk label type: gpt
```

Number	Start	End	Size	File system	Name	Flags
--------	-------	-----	------	-------------	------	-------

```
1      34s      71687338s 71687305s
```

```
(parted) quit
```

```
Information: Don't forget to update /etc/fstab, if necessary.
```


To write an XVM label on the GPT-labeled lun, perform the following:

```
> xvm label -name test1 /dev/xscsi/pci0002:00:02.1/node20000011c61dd850/port1/lun0/disc
test1
Performing automatic probe for alternate paths.
Could not probe /dev/xscsi/pci0001:00:03.0-1/target2/lun0/disc: device has
been claimed by another user
Performing automatic path switch to preferred path for phys/test1.
phys/test1: failover: group has no preferred member

> xvm show -v phys/test1
XVM physvol phys/test1
=====
size: 71670955 blocks sectorsize: 512 bytes state: online,local,accessible
uuid: a8853e23-5af5-4524-bbc9-fdd0b04aelec
system physvol: no
physical drive: /dev/xscsi/pci0002:00:02.1/node20000011c61dd850/port1/lun0/disc on host
puffin
preferred path: unspecified
available paths:
                /dev/xscsi/pci0002:00:02.1/node20000011c61dd850/port2/lun0/disc <dev 2176>
affinity=0
                /dev/xscsi/pci0002:00:02.0/node20000011c61dd850/port2/lun0/disc <dev 16976>
affinity=0
                /dev/xscsi/pci0002:00:02.0/node20000011c61dd850/port1/lun0/disc <dev 17328>
affinity=0
                /dev/xscsi/pci0002:00:02.1/node20000011c61dd850/port1/lun0/disc <dev 16864>
affinity=0 <current path>
Disk has the following XVM label:
  Clusterid: 0
  Host Name: puffin
  Disk Name: test1
  Magic: 0x786c6162 (balx)      Version 2
  Uuid: a8853e23-5af5-4524-bbc9-fdd0b04aelec
  last update: Mon May  4 15:41:20 2009
  state: 0x91<online,local,accessible> flags: 0x0<idle>
  sechbytes: 512
  label area: 8157 blocks starting at disk block 35 (0 used)
  user area: 71670955 blocks starting at disk block 8192

Physvol Usage:
Start          Length          Name
-----
0              71670955      (unused)
```

Local stats for phys/test1 since being enabled or reset:

stats collection is not enabled for this physvol

To use the parted command to create a GPT label with two partitions, perform the following:

```
> parted /dev/disk/by-path/pci-0002:00:02.0-fc-0x21000011c61dd850:0x0000000000000000
```

```
GNU Parted 1.8.8
```

```
Using /dev/sdad
```

```
Welcome to GNU Parted! Type 'help' to view a list of commands.
```

```
(parted) mklabel gpt
```

```
Warning: The existing disk label on /dev/sdad will be destroyed and all data on this disk will be lost. Do you want to continue?
```

```
Yes/No? y
```

```
(parted) print
```

```
Model: SGI ST336753FC (scsi)
```

```
Disk /dev/sdad: 36.7GB
```

```
Sector size (logical/physical): 512B/512B
```

```
Partition Table: gpt
```

```
Number  Start  End  Size  File system  Name  Flags
```

```
(parted) mkpart PRIMARY XFS 0 2MB
```

```
(parted) mkpart PRIMARY XFS 2MB 37GB
```

```
(parted) unit s
```

```
(parted) print
```

```
Model: SGI ST336753FC (scsi)
```

```
Disk /dev/sdad: 71687372s
```

```
Sector size (logical/physical): 512B/512B
```

```
Partition Table: gpt
```

```
Number  Start  End  Size  File system  Name  Flags
  1      34s   3906s  3873s          PRIMARY  , , , , , , , , , ,
  2     3907s 71687338s 71683432s          PRIMARY  , , , , , , , , , ,
```

```
(parted) quit
```

```
Information: You may need to update /etc/fstab.
```

To write an XVM label on the GPT-labeled lun, perform the following:

```
> xvm label -name test2
```

```
/dev/disk/by-path/pci-0002:00:02.0-fc-0x21000011c61dd850:0x0000000000000000
```

test2

Performing automatic probe for alternate paths.

Could not probe /dev/disk/by-path/pci-0001:00:03.0-scsi-0:0:1:0: device has been claimed by another user

Performing automatic path switch to preferred path for phys/test2.

phys/test2: failover: group has no preferred member

>xvm show -v phys/test2

XVM physvol phys/test2

=====

size: 71670955 blocks sectorsize: 512 bytes state: online,local,accessible

uuid: e59ele61-6467-4948-9012-0eb5461c141a

system physvol: no

physical drive: /dev/disk/by-path/pci-0002:00:02.0-fc-0x21000011c61dd850:0x0000000000000000

on host puffin

preferred path: unspecified

available paths:

/dev/disk/by-path/pci-0002:00:02.1-fc-0x22000011c61dd850:0x0000000000000000

<dev 16976> affinity=0

/dev/disk/by-path/pci-0002:00:02.1-fc-0x21000011c61dd850:0x0000000000000000

<dev 17328> affinity=0

/dev/disk/by-path/pci-0002:00:02.0-fc-0x22000011c61dd850:0x0000000000000000

<dev 2160> affinity=0

/dev/disk/by-path/pci-0002:00:02.0-fc-0x21000011c61dd850:0x0000000000000000

<dev 16848> affinity=0 <current path>

Disk has the following XVM label:

Clusterid: 0

Host Name: puffin

Disk Name: test2

Magic: 0x786c6162 (balx) Version 2

Uuid: e59ele61-6467-4948-9012-0eb5461c141a

last update: Mon May 4 16:32:06 2009

state: 0x91<online,local,accessible> flags: 0x0<idle>

secbytes: 512

label area: 8157 blocks starting at disk block 35 (0 used)

user area: 71670955 blocks starting at disk block 8192

Physvol Usage:

Start	Length	Name
0	71670955	(unused)

Local stats for phys/test2 since being enabled or reset:

stats collection is not enabled for this physvol

Note that the user area does not start at the beginning of partition 2, but at the 4MB boundary. This is done to avoid writing user data into the first 4MB of the disk, which is not recommended for some raid products. The length of the user area is also shortened to avoid writing user data into the last 4MB of the disk..

Composition of XVM Logical Volumes

XVM logical volumes are composed of a hierarchy of logical storage objects: volumes are composed of subvolumes; subvolumes are composed of stripes, mirrors, concats (concatenated volume elements), and slices combined in whatever hierarchy suits your system needs; and, at the bottom of the hierarchy, each logical storage object is ultimately made up of slices, which define an area of physical storage. Each of these logical storage objects is known as a *volume element* or a *ve*.

The concat, stripe, and mirror logical volume elements can be arranged and stacked arbitrarily. There is a limit of ten levels from the volume through the slice, inclusive.

A logical volume element beneath another volume element in the hierarchy is known as a *child* or *piece* of the higher-level volume element. Volumes are limited to 255 children, subvolumes are limited to 1 child, and mirrors are limited to 8 children. Other volume elements are limited to 65,536 children.

Figure 1-7 shows an example of a simple XVM logical volume. In this example, there is one data subvolume that consists of a single two-way stripe.

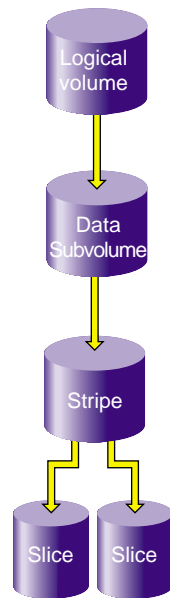


Figure 1-7 Basic XVM Striped Logical Volume

Figure 1-8 shows an XVM logical volume with three subvolumes and a mirrored stripe in the data subvolume.

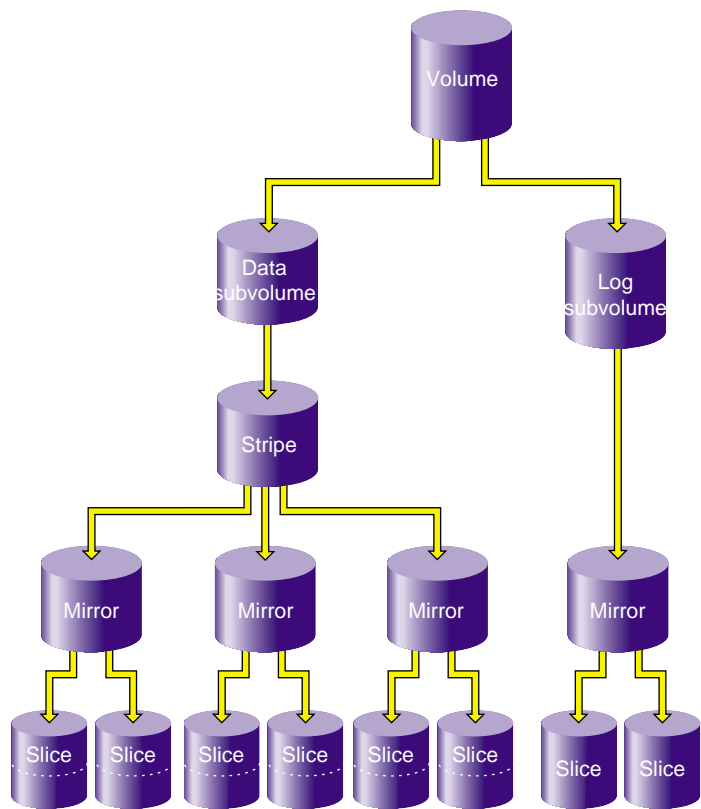


Figure 1-8 XVM Logical Volume with Mirrored Stripe and Three Subvolumes

Figure 1-9 shows the example illustrated in Figure 1-8 after the insertion of a concat. In this example, additional slices were created on the unused disk space on the disks that made up the data subvolume. These slices were used to create a parallel mirrored stripe, which was combined with the existing mirrored stripe to make a concat.

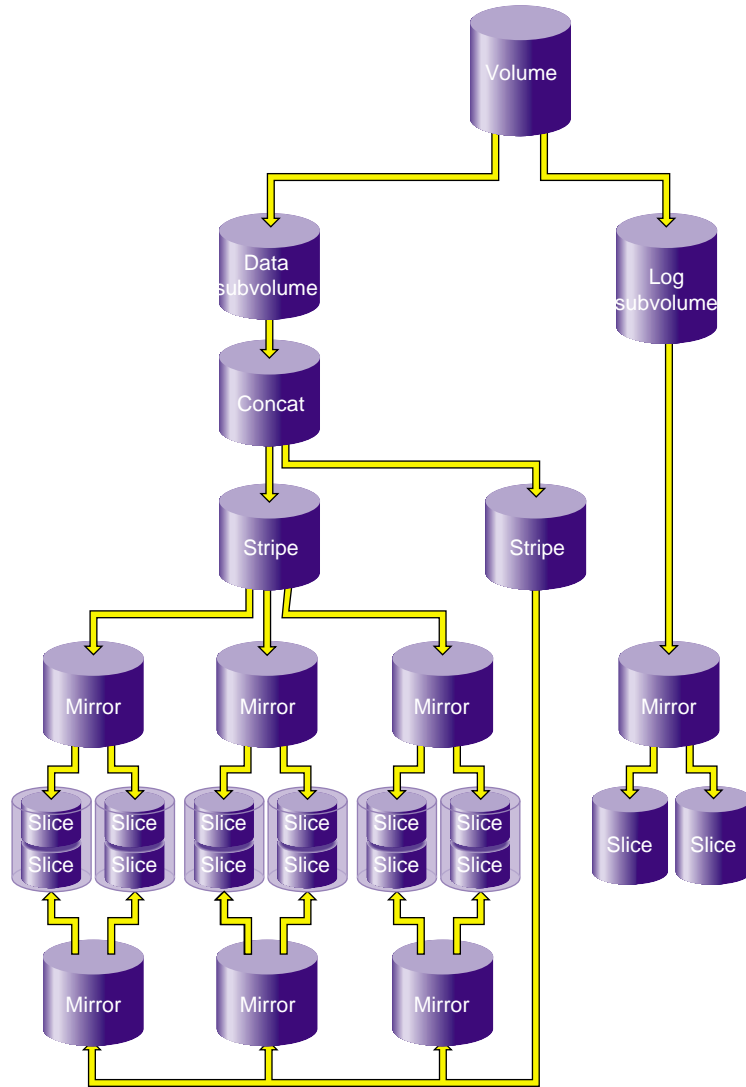


Figure 1-9 XVM Logical Volume after Insertion of Concat

The following subsections describe the XVM volume elements in greater detail.

Volumes

A *volume* is the topmost XVM volume element. It is a collection of subvolumes, which are grouped together into a single volume name.

Each volume can be used as a single filesystem. Volume information used by the system is stored in logical volume labels in the volume header of each disk used by the volume.

You can create volumes, delete volumes, and move volumes between systems.

The subvolumes that make up a volume can be marked as data subvolumes and log subvolumes. These are the system-defined subvolume types, and are described in “Subvolumes” on page 22. You can also mark a subvolume as being of a user-defined type.

You cannot have more than one subvolume of a particular system-defined subvolume type under the same volume. In other words, a volume can contain only one data subvolume and only one log subvolume. This restriction does not apply to subvolumes of user-defined types.

Figure 1-10 shows an XVM volume with system-defined subvolume types.

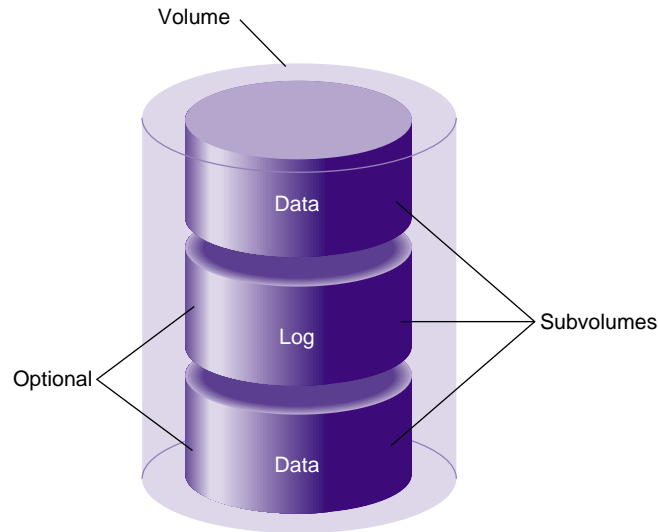


Figure 1-10 XVM Volume with System-Defined Subvolume Types

Figure 1-11 shows an XVM volume with user-defined subvolume types, which have been defined as types 16, 17, and 18. In this example, the volume is named `animation` and the subvolumes are named `wire-data`, `shading`, and `texturemap`. For information on subvolumes, see “Subvolumes” on page 22. For information on XVM object names, see “XVM Object Specification” on page 59.

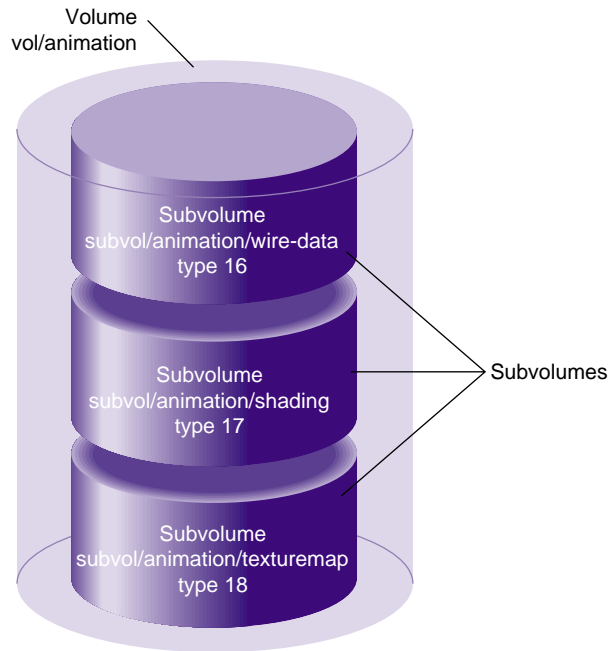


Figure 1-11 XVM Volume with User-Defined Subvolume Types

Subvolumes

A *subvolume* is the entry point for XVM logical volume I/O. Each subvolume is a distinct address space and a distinct type. There can be only one volume element beneath a subvolume in an XVM topology.

Subvolumes can be of the following system-defined types:

Data subvolume

An XFS data subvolume is required for all XVM logical volumes acting as filesystem devices.

Log subvolume

The log subvolume contains a log of XFS filesystem transactions and is used to expedite system recovery after a crash. A log subvolume is optional for an XVM logical volume; if one is not present, the filesystem log is kept in the data subvolume.

System-defined subvolume types cannot have user-defined names.

A subvolume can also be marked as being of a user-defined type. You can specify a name for a subvolume of a user-defined type.

Subvolumes enforce separation among data types. For example, user data cannot overwrite filesystem log data. Subvolumes also enable filesystem data and user data to be configured to meet goals for performance and reliability. For example, performance can be improved by putting subvolumes on different disk drives.

Figure 1-12 shows four examples of the composition of an XVM subvolume, showing that an XVM subvolume can contain only one child volume element.

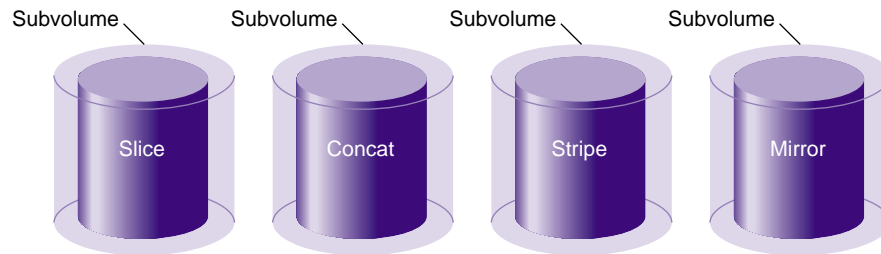


Figure 1-12 XVM Subvolume Examples

Slices

Slices are the lowest level in the hierarchy of XVM logical volumes. Slices define physical storage; they map address space of a physical disk onto a volume element.

Concats

A *concat* is an XVM volume element that combines other volume elements so that their storage is combined into one logical unit. For example, two slices can be combined into a single concat.

Figure 1-13 shows a concat that is composed of two slices.

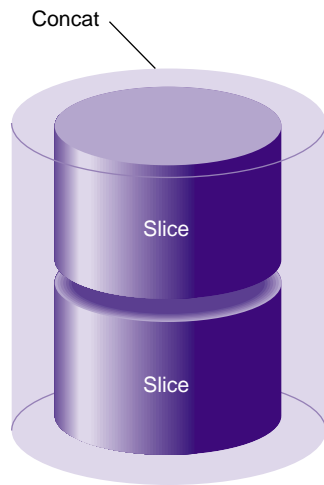


Figure 1-13 Concat Composed of Two Slices

Figure 1-14 shows a concat that is composed of two mirrors.

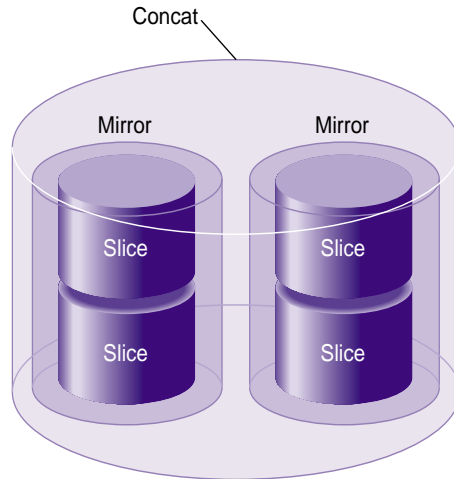


Figure 1-14 Concat Composed of Two Mirrors

Stripes

A *stripe* is an XVM volume element that consists of two or more underlying volume elements. These elements are organized so that an amount of data called the *stripe unit* is written to and read in from each underlying volume element in a round-robin fashion.

Striping can be used to alternate sections of data among multiple disks. This provides a performance advantage by allowing parallel I/O activity.

Figure 1-15 shows a three-way stripe.

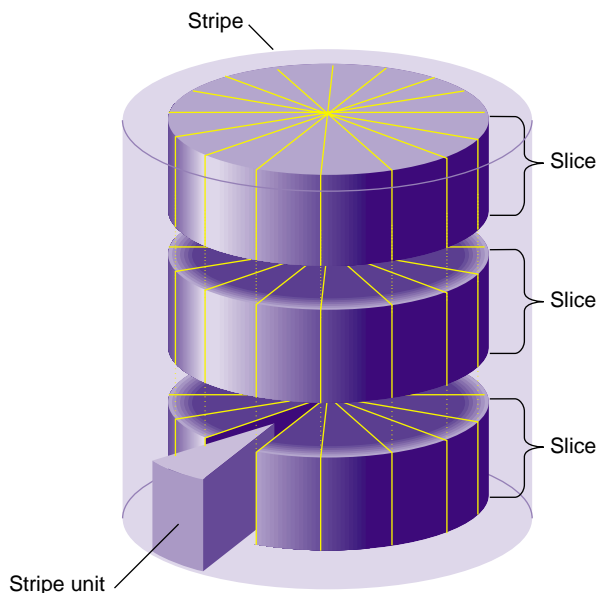


Figure 1-15 Three-Way Stripe

A stripe configured on top of another stripe may provide performance benefits over a single wider stripe. In Figure 1-16, two three-way stripes are created and then striped again using a larger stripe unit size. If configured correctly, disjoint sequential access (where different processes are doing sequential I/O to different parts of the address space) will end up on different halves of the top-level stripe. The advantage of this configuration is that for parallel large accesses, the two halves of the top-level stripe can operate independently, whereas with a single six-way stripe, multiple I/O operations would be outstanding to each disk, causing the need for a disk seek.

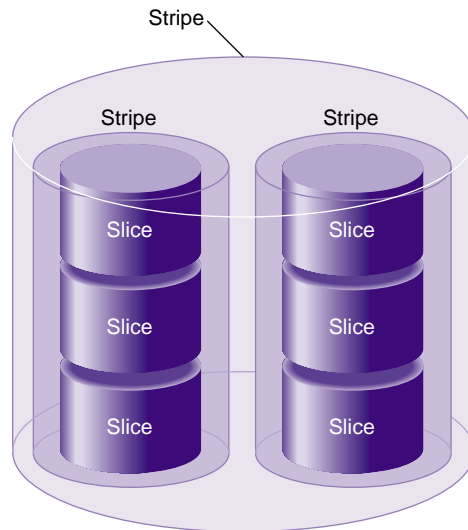


Figure 1-16 Stripe on Stripe Volume Element

Mirrors

A *mirror* is an XVM volume element that maintains identical data images on its underlying volume elements. This data redundancy increases system reliability. The components of a mirror do not have to be identical in size, but if they are not there will be unused space in the larger components.

Figure 1-17 shows a mirror that is composed of two slices.

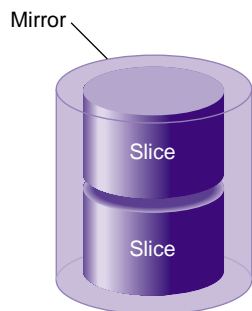


Figure 1-17 Mirror Composed of Two Slices

Figure 1-18 shows a mirror that is composed of two stripes.

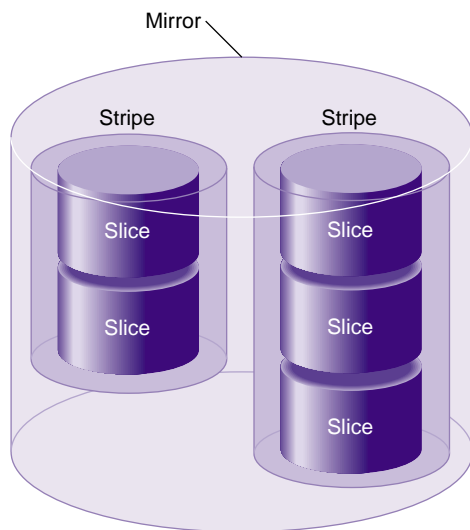


Figure 1-18 Mirror Composed of Two Stripes

Figure 1-19 shows a mirror composed of a stripe and a concat.

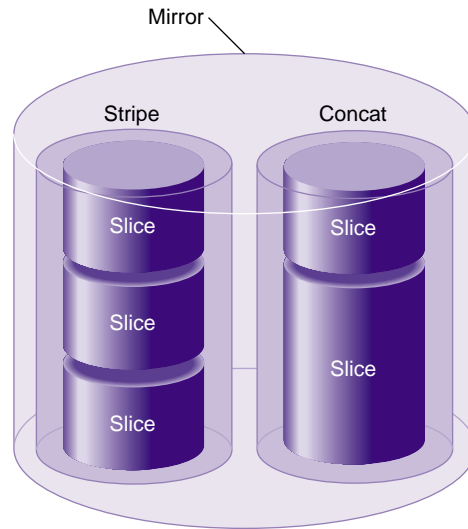


Figure 1-19 Mirror Composed of a Stripe and a Concat

Writing Data to Logical Volumes

A logical volume can include slices from several physical disk drives. If the logical volume is not striped, data is written to the first component of a volume element until that component is full, then to the second component, and so on. Figure 1-20 shows the order in which data is written to a concatenated logical volume. In this figure, each wedge represents a unit of data that is written to disk. Data is written to the first component until it is filled, and then data is written to the second component until it is filled, and so forth.

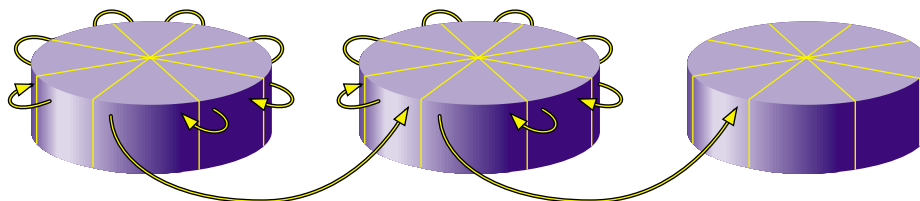


Figure 1-20 Writing Data to a Non-Striped Logical Volume

If the logical volume is striped, an amount of data called the stripe unit is written to each underlying volume element in a round-robin fashion. Figure 1-21 shows the order in which data is written to a striped volume element with a three-way stripe. Each wedge represents a stripe unit of data. One stripe unit of data is written to the first component of the stripe, then one stripe unit of data is written to the second component of the stripe, then one stripe unit of data is written to the third component of the stripe. After this, the next stripe unit of data is written to the first component, and so forth.

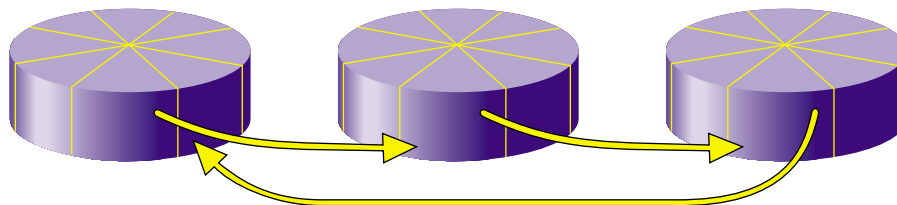


Figure 1-21 Writing Data to a Striped Logical Volume

As an example of configuring stripes to improve performance, consider a situation where your typical I/O activity is 2MB in size and you have 4 disks. If the disks are concatenated, the I/O will all go to one disk. If the 4 disks are striped with a stripe unit of 512KB, then a 2MB I/O activity will use all four disks in parallel, each supplying 512KB of data. You should be able to get the best performance by making your stripe width (the stripe unit times the number of disks) equal to the I/O size that is most common.

If the LUNs you are striping are RAID devices, then it is also advantageous to have your XVM stripes line up on and be a multiple of the RAID stripe boundaries. This will not only allow all your XVM luns to transfer in parallel, but all the disks in the RAID will be accessed in parallel, in units of the same size.

XVM Logical Volumes in a CXFS Cluster

The XVM Volume Manager is used by CXFS filesystems, which are shared among the nodes in a CXFS cluster. Because of this, an XVM physical volume has a *domain*, which can be *cluster* or *local*. An XVM physical volume with a cluster domain is owned by a CXFS cluster, while an XVM physical volume with a local domain is owned by a single node.

An XVM physical volume that has a cluster domain can be configured and modified by any node in the CXFS cluster that owns it. An XVM physical volume that has a local domain can be configured and modified only by the local node that owns it. The XVM logical volumes that are contained on XVM physical volumes with a local domain are considered to be local volumes.

For information on XVM domains, see “XVM Domains” on page 34. For information on CXFS, see *CXFS 6 Administration Guide for SGI InfiniteStorage*.

XVM Logical Volumes and Failover

If your XVM configuration requires that you spread I/O across controllers, you must define a complete failover configuration file. This is necessary to ensure that I/O is restricted to the path that you select. For example, if you want a striped volume to span two host bus adapters, you must configure a failover configuration file to specify the preferred paths.

There are two failover mechanisms that XVM uses to select the preferred I/O path, each with its associated failover configuration file:

- Failover version 1 (V1), which uses the `failover.conf` configuration file
- Failover version 2 (V2) which uses the `failover2.conf` configuration file

XVM failover is described in Chapter 5, “XVM Failover”.

Installing the XVM Volume Manager

When running SGI Foundation Software, you must install the following rpms to use the XVM volume manager:

- `xvm_commands`
- `xvm_standalone-module`

Path Manager

The Path Manager feature on Linux nodes provides a mechanism to add features to improve I/O performance without restricting those features to XVM volumes. Path Manager applies to all supported SUSE Linux Enterprise Server (SLES) servers/clients and Red Hat Enterprise Linux 5 (RHEL 5) or later clients.

The Path Manager feature does **not** require any additional actions on the part of a system administrator.

Path Manager includes the following:

- Automatic selection of an I/O path on an SGI® UV™ system to minimize the nonuniform memory access (NUMA) interconnect traffic
- A new `pathmgr` kernel module that provides the path failover functionality that was formerly part of the `xvm` command. XVM administration has not changed, but output from the `xvm show` command has slight differences.

Limitations and Considerations

Do not run `fsck` for XFS filesystems listed in `/etc/fstab` that use XVM devices (that is, you should set the `fsck` flag to 0), because XVM devices are not always available. If an `fsck` command is run on an XFS filesystem when XVM devices are not available, the system may suspend the system boot sequence and require input from the administrator. XVM includes a helper service that mounts all filesystems listed in `/etc/fstab` that use XVM devices at the time XVM is started during the boot sequence.

XVM Administration Concepts

Before configuring and administering XVM logical volumes, you should be familiar with the concepts that underlie the administration commands. This chapter describes the tasks that the XVM Volume Manager performs on physical and logical disk resources. More complete descriptions of the `xvm` command line interface (CLI) commands are provided in Chapter 4, “XVM Administration Commands,” along with examples of each of the commands.

The major sections in this chapter are:

- “XVM Objects” on page 33
- “XVM Domains” on page 34
- “Physical Disk Administration” on page 37
- “Creating Logical Resources” on page 40
- “Managing Logical Resources” on page 49
- “Destroying Logical Resources” on page 53

XVM Objects

An XVM object can be one of the following:

unlabeled disk

An *unlabeled disk* is a disk that has not been labeled as an XVM disk by the XVM Volume Manager.

A disk that has been labeled as an XVM disk but has not had its labels read by the XVM Volume Manager since the system was last booted is also considered an unlabeled disk by the XVM Volume Manager. This situation could arise, for example, when a previously labeled disk is added to a running system.

physical volume

A disk that has been labeled for use by the XVM Volume Manager is an XVM *physical volume*, or *physvol*.

foreign disk

A *foreign* disk is a disk with an XVM physical volume label but which cannot be administered by the current node because it is owned by a different node or a different cluster.

volume element

A *volume element*, or *ve*, is a building block of an XVM logical volume topology. XVM volumes, subvolumes, concats, stripes, mirrors, and slices are all XVM volume elements.

XVM Domains

An XVM physical volume has a domain, which can be cluster or local. An XVM physical volume with a cluster domain is owned by a CXFS cluster, and it can be controlled by any of the nodes in that cluster, which is defined by the CXFS Cluster Manager. An XVM physical volume with a local domain is owned by a single node, and it can be controlled only by that node.

Only the owner for an XVM physical volume can modify the configuration on that physical volume. There may be XVM physical volumes that are seen by a host, but owned by another host or another cluster. XVM recognizes these disks and marks them as foreign. Disks without an XVM label are shown as unlabeled.

Figure 2-1 illustrates a physical volume that is controlled by a local owner. In this example, the XVM physvol `lucy` has a local domain of node `ricky`. The node `ricky` is part of the CXFS cluster `neighbors` that also includes the node `fred` and the node `ethel`, but neither `fred` nor `ethel` can control `lucy`.

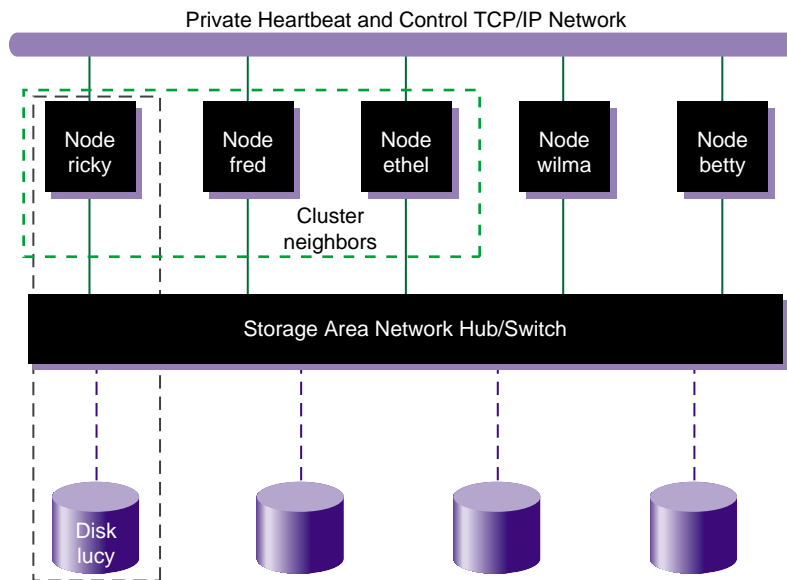


Figure 2-1 XVM Physical Volume in Local Domain

In the configuration illustrated in Figure 2-1, the node `ricky` can see and modify the configuration of physvol `lucy`. The nodes `fred`, `ethel`, `wilma`, and `betty` see `lucy` as a foreign disk, and display only the disk path and not the physvol name itself. (If necessary, you can execute the `show` command on a foreign disk to determine its physvol name, as described in “Displaying Physical Volumes with the `show` Command” on page 69.)

Figure 2-2 illustrates a physical volume that has a cluster domain. In this example, the physvol `lucy` has an owner of cluster `neighbors`, which consists of the nodes `ricky`, `fred`, and `ethel`.

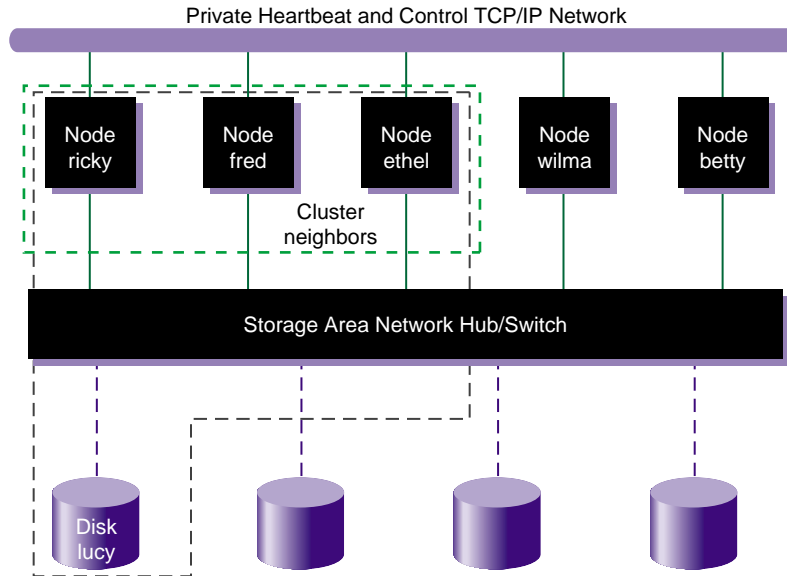


Figure 2-2 XVM Physical Volume in Cluster Domain

In the configuration illustrated in Figure 2-2, the nodes `ricky`, `fred`, and `ethel` can see and modify the configuration of the physvol `lucy`. The nodes `wilma` and `betty` cannot modify the configuration of `lucy`, even though they are connected to `lucy` through a SAN network; they see `lucy` as a foreign disk, and can display only the disk path.

An XVM logical volume that spans physvols may not span domains on a running system. A logical volume that spans local and cluster domains is marked offline.

You can bring up the `xvm` command line interface in local or cluster mode to do some administrative tasks for the volumes in that domain. The `xvm:cluster>` prompt appears by default, indicating that all XVM physical volumes that you create in this XVM session are in the cluster domain. When cluster services are not enabled, the `xvm:local>` prompt appears, indicating that all XVM physical volumes that you create in this XVM session are in the local domain.

You can change the current XVM domain by invoking XVM with the `-domain` option, as described in “Using the XVM CLI” on page 55, or by using the `set` command of the XVM Volume Manager, as described in “Changing the Current Domain with the set Command” on page 68.

When you are running the XVM Volume Manager in the cluster domain, by default you can see and modify only the XVM physvols that are also in the cluster domain, even if you are running from the node that is the owner of a local physvol. To see and modify local disks, you either change your domain to local with the `set domain` command, or you use the `local:` prefix when specifying a physvol name. Similarly, when you are running the XVM Volume Manager in the local domain, you must change your domain to cluster or specify a `cluster:` prefix when specifying the physvol that is owned by the cluster. For information on setting and specifying XVM domains, see “Using the XVM CLI” on page 55.

You can change the owner of an existing XVM physvol by using the XVM `give` command to give that physvol to a different owner, whether that owner is a single node or a cluster. When the node or cluster that currently owns the physical volume is unable to execute the `give` command, you can use the `steal` command to change the domain of an XVM physical volume. For information on the `give` and `steal` commands, see “Removing Disks from the XVM Volume Manager with the unlabel Command” on page 73.

Physical Disk Administration

Underlying XVM logical volumes are the physical volumes that make up the logical volumes. As part of XVM logical volume administration, you create, manage, and destroy XVM physical volumes, as described in the following sections.

Creating Physical Volumes

In order to create XVM logical volumes on a physical disk, you must label the disk as an XVM disk by using the `label` command of the XVM Volume Manager. This command writes out an XVM physical volume label on a disk and allows the XVM volume manager to control the partitioning on the disk. In a CXFS cluster, any XVM physical volumes that will be shared must be physically connected to all nodes in the cluster.

When you label an XVM disk, you can specify whether the disk is an option disk, a system disk with a combined root and `usr` filesystem, or a system disk with separate root and `usr` filesystems. XVM physical volumes are option disks by default. You must be administering XVM in the local domain when you create an XVM system disks, since system disks are always local to the node which boots from them.

Note: XVM on Linux does not support XVM system disks.

If you add a new disk that has already been labeled as an XVM physical volume to a running system, you must manually probe the disk by using the `probe` command of the XVM Volume Manager in order for the system to recognize the disk as an XVM disk. You do not have to do this when you are labeling a new XVM disk on your system, however, since the XVM Volume Manager probes the disk as part of the label process. All disks are probed when the system is booted to determine which disks are XVM disks.

By default, you cannot label a disk as an XVM disk if the disk contains any partitions that are currently in use as mounted filesystems. You can override this restriction with the `-nopartchk` option of the `label` command. Use the `-nopartchk` option with caution, as data corruption or system panics can result from labeling disks with partitions that are in use.

Note: Before you can label a disk as an XVM disk, it must be formatted as an SGI disk. For information on formatting a disk for use in XVM, see “Preparing to Configure XVM Volumes under Linux” on page 107.

Managing Physical Volumes

You can perform the following tasks on physical volumes:

- Display the physical volume
- Change the domain of the physical volume
- Add a physical volume to a running system
- Replace a physical volume
- Rename a physical volume

- Display statistics for a physical volume
- Change a physical volume from a system disk to an option disk

These tasks are described in the following subsections.

Displaying Physical Volumes

Use the XVM `show` command of the XVM Volume Manager to display information about physical volumes, both labeled and unlabeled. You can also use the `show` command to display information about disks that are foreign to the current node. You can use this feature to determine the current owner of a disk that is foreign to you, as described in “Displaying Physical Volumes with the `show` Command” on page 69.

Changing the Domain of a Physical Volume

Use the XVM `give` command to change the owner of an existing XVM `physvol`, giving that `physvol` to a different local or cluster owner. When the node or cluster that currently owns the physical volume is unable to execute the `give` command, you can use the `steal` command to change the domain of an XVM physical volume. For information on the `give` and `steal` commands, see “Removing Disks from the XVM Volume Manager with the `unlabel` Command” on page 73.

Adding a Physical Volume to Running System

When you boot your system, all disks connected to the system are probed to determine whether they are XVM disks. If you add an XVM disk to a system that is already running, you must manually probe the disk by using the `probe` command of the XVM Volume Manager so that the kernel recognizes the disk as an XVM disk.

Replacing a Physical Volume

The XVM Volume Manager allows you to replace a disk on a running system without rebooting the system. When you do this, you must regenerate the XVM label on the replacement disk. Use the `dump` command of the XVM Volume Manager to dump the commands to a file that will regenerate a physical volume label.

Note that when you dump the commands to regenerate a physical volume label, you must separately and explicitly dump the commands to regenerate the volume element

tree that leads to the physical volume, as described in “Reconstructing Volume Elements: Using the dump Command” on page 86.

Renaming a Physical Volume

You can rename a physical volume with the `name` option of the `change` command.

Physical Volume Statistics

The XVM Volume Manager can maintain statistics for physical volumes, subvolumes, stripes, concats, mirrors, and slices. You can use the `stat` option of the `change` command of the XVM Volume Manager to turn statistics on and off and to reset the statistics for a physical volume. See Chapter 7, “Statistics,” for information on the statistics XVM maintains.

In a clustered environment, statistics are maintained for the local node only.

Destroying Physical Volumes

To remove an XVM physical volume from a system, use the `unlabel` command of the XVM Volume Manager to remove the XVM physical volume label from an XVM disk and restore the original partitioning scheme. The `unlabel` command provides a `-force` option which deletes each slice that currently exists on the physical volume, even if the slice is part of an open subvolume and its deletion will cause the subvolume state to go offline.

The swap partition of an XVM system disk cannot be deleted. This is to ensure that the swap partition cannot be deleted accidentally and cause a system panic.

Creating Logical Resources

After you have created the XVM physical volumes you will use for your logical volume, you can create the logical volume elements that will make up the logical volume.

Creating Topologies

XVM topologies can be built top-down or bottom-up. Any tree or subtree you create that does not end in a slice will not have labels written to disk, and therefore will not be persistent across reboots.

While you are building your XVM topology, you may find it useful to display the existing defined topology for a volume element by using the `-topology` option of the `show` command of the XVM Volume Manager.

Automatic Volume and Subvolume Creation

When volume elements other than volumes are created, they must be associated with a volume. You can name and create the volume explicitly when you create the volume element, or you can specify that the volume be automatically generated with a temporary name. A subvolume of type `data` is automatically generated for the volume (unless the volume element you are creating is itself a subvolume of a different type). Automatic volume and subvolume generation ensures that when an object is constructed, it can be immediately used by an application such as `mkfs` to initialize a filesystem.

When you explicitly name a volume, the volume name is stored in the label space and remains persistent across machine reboots. When the system generates a volume and volume name automatically, a new and possibly different name will be generated when the system reboots. Slices, however, are a special case; when the system generates a volume name for a slice, the volume name is not temporary and remains persistent across reboots.

You can make a temporary volume name persistent across reboots by using the `change` command to rename the volume.

Volume Element Naming

Volume elements that compose an XVM volume are named as follows:

- Slices are named automatically when you create them.

Slice names remain persistent across machine reboots. This makes it convenient to reorganize and rebuild logical volumes using slices you have defined for each disk, even after you have rebooted the system.

- You can name stripes, concats, and mirrors explicitly when you create them. If you do not name them explicitly, you must specify that a default temporary name should be generated.

When you name stripes, concats, and mirrors explicitly, the volume element name is stored in the label space and remains persistent across machine reboots.

Information on setting the size of the label space is provided in “Assigning Disks to the XVM Volume Manager with the label Command” on page 68.

- You can name a subvolume explicitly only if it is of a user-defined type. Data subvolumes are named `data` and log subvolumes are named `log`.
- As described in “Automatic Volume and Subvolume Creation” on page 41, volumes can be created and named when you create the elements within those volumes. You can also create an empty volume and give it a name explicitly. If you do not name an empty volume when you create it, you must specify that the system generate a temporary name; this practice is not recommended for general configuration.

You can make temporary volume element names persistent across reboots by using the `change` command to rename the volume element.

It is not necessary to use the name of a volume element when you manipulate it. You can use its relative position in the logical volume instead. These naming options as well as general information on the syntax of volume element names are described in “Object Names in XVM” on page 59.

Attaching Volume Elements

When you create XVM logical volumes by attaching volume elements to one another through volume element creation or through the `attach` command, the following rules and restrictions are enforced by the XVM Volume Manager:

- The source of an attach must be a subvolume, or the child of a subvolume. You cannot attach a volume to another volume element.
- Subvolumes can be attached only to volumes.
- Subvolumes can have only one child.
- A volume cannot have more than one system-defined subvolume of a given type. The system-defined subvolumes are data subvolumes and log subvolumes.
- A mirror cannot have more than eight members.

- If you specify a position when you create or attach a volume element to a target volume element, the target volume element must not already have a volume element in that position.
- When attaching a volume element to a target that is part of an open subvolume, the attachment cannot change the way the data is laid out in the target or any ancestor of the target. Examples of attaches that can affect data layout are:
 - Appending to a stripe
 - Appending to a concat which results in the growth of an ancestor that is not the rightmost volume element under its parents

When you attach a volume element to a mirror, this initiates a mirror revive during which the system mirrors the data. A message is written to the `SYSLOG` when this process is complete. You cannot halt a mirror revive once it has begun except by detaching all but one of the pieces of the mirror.

When you use the `-safe` option of an `xvm` command, you cannot attach volume elements that change the way the data is laid out in the target or any ancestor of the target even if the target does not belong to an open subvolume.

When you attach multiple source volume elements to a single target volume element, they are attached one at a time, in turn. If an attach in the list fails, XVM attempts to restore the volume elements to their previous parents. If a volume element cannot be restored, a warning message is generated and manual intervention is needed.

Detaching Volume Elements

Use the `detach` command of the XVM Volume Manager to detach a volume element from its parent. When you detach a volume element, a new volume (and possibly data subvolume) will be created, just as a volume is created when you create a volume element. You can name the generated volume explicitly, or you can specify that the volume be automatically generated with a temporary name. A subvolume of type `data` is automatically generated for the volume element you are detaching (unless the volume element you are detaching is itself a subvolume of a different type).

If the volume element you detach is part of an open subvolume, its detachment cannot cause the subvolume state to go offline. Any command that would reduce the address space of an open subvolume, such as detaching a slice that is not mirrored, will cause it to go offline. You cannot detach the last valid piece of an open mirror from that mirror, since this will cause the mirror to go offline.

The `detach` command provides a `-force` option to override the restriction that you cannot detach a volume element that will cause a subvolume to go offline. The `detach` command also provides a `-safe` option to impose this restriction even if the subvolume is not open. See “The detach Command” on page 81 for examples of this command.

You cannot detach a mirror leg that is being revived without using the `-force` option. You cannot detach the last leg of a mirror without using the `-force` option. These restrictions hold whether or not the subvolume is open.

Empty Volume Elements

When you create stripes, mirrors, concats, subvolumes, and volumes, you have the option of not specifying which child volume elements will compose these volume elements. If you do not specify the child elements, an empty volume element is created and you can attach volume elements at a later time.

Logical Volume Statistics

The XVM Volume Manager can maintain statistics for physical volumes, subvolumes, stripes, concats, mirrors, and slices. You can use the `stat` option of the `change` command of the XVM Volume Manager to turn statistics on and off and to reset the statistics for a volume element. See Chapter 7, “Statistics,” for information on the statistics XVM maintains.

In a clustered environment, statistics are maintained for the local node only.

Creating Slices

Use the `slice` command to create a slice from a block range of an XVM physical volume. You can specify the starting block of a slice and you can specify the length of a slice. In addition, you can specify the following methods of creating slices:

- You can create a slice out of all of the blocks of a physical volume
- You can divide a specified address range into equal parts, with each part a different slice
- You can slice multiple physical volumes at once
- You can specify that a slice is a system slice of type `root`, `swap`, or `usr`.

- With the `slice - align` option, you can specify that a slice must start on a sector which is an even multiple of some number, relative to the start of the lun. The slice length is rounded to an even multiple of that number. This is useful if the lun is created as a stripe in a raid.

Slices are named automatically and are persistent across machine reboots. You cannot rename slices.

The volume that is generated when you create a slice is persistent across machine reboots. You can specify the name of the volume that is created when you create a slice. By default, the volume name will be the same as the slice object name.

Creating Concats

Use the `concat` command of the XVM Volume Manager to create a concat, which is a volume element that concatenates all of its child volume elements into one address space.

The XVM Volume Manager enforces the rules of attachment during concat creation, as described in “Attaching Volume Elements” on page 42.

Creating Stripes

Use the `stripe` command of the XVM Volume Manager to create a stripe, which is a volume element that stripes a set of volume elements across an address space.

You can create a stripe that is made up of volume elements of unequal size, although this will leave unused space on the larger volume elements.

The XVM Volume Manager enforces the rules of attachment during stripe creation, as described in “Attaching Volume Elements” on page 42.

For information on configuring stripes that span two host bus adaptors, see “XVM Logical Volumes and Failover” on page 31.

Creating Mirrors

Use the `mirror` command of the XVM Volume Manager to create a mirror, which is a volume element that mirrors all of its child volume elements.

When you create a mirror that has more than one piece, a message is written to the `SYSLOG` indicating that the mirror is reviving. This indicates that the system is beginning the process of mirroring the data. Another message is written to the `SYSLOG` when this process is complete. Should the revive fail for any reason, a message will be written to the system console as well as to the `SYSLOG`.

For large mirror components, this revive process may take a long time. When you are creating a new mirror that does not need to be revived, you should consider using the `-clean` option of the `mirror` command, as described in “The `-clean` Mirror Creation Option” on page 48. When you are creating a new mirror that you will use for scratch filesystems that will never need to be revived, you should consider using the `-norevive` option of the `mirror` command, as described in “The `-norevive` Mirror Creation Option” on page 48.

You cannot halt a mirror revive once it has begun except by detaching all but one of the pieces of the mirror. For more information on mirror revives, see “Mirror Revives” in Chapter 8.

The XVM Volume Manager enforces the rules of attachment during mirror creation, as described in “Attaching Volume Elements” on page 42.

When you create a mirror you have the options of setting the following characteristics for the mirror:

- The read policy for the mirror
- The primary leg for the mirror
- Whether the mirror will be synchronized at creation (the `-clean` option)
- Whether the mirror will be resynchronized when the system boots (the `-norevive` option)

The following sections describe each of these options.

Read Policies

The XVM Volume Manager allows you to specify one of the following read policies for a mirror:

- | | |
|--------------------------|---|
| <code>round-robin</code> | Balance the I/O load among the members of the mirror, blindly reading in a round-robin fashion. |
| <code>sequential</code> | Route sequential I/O operations to the same member of the mirror. |

Figure 2-3 illustrates how data is read from the legs of a mirror with a round-robin read policy. The wedges represent units of data that you are reading. The first read operation gets the unit of data from the first leg, the second read operation gets the next unit of data from the second leg, and the next read operation gets the next unit of data from the third leg. The next read operation gets the requested unit of data from the first leg again.

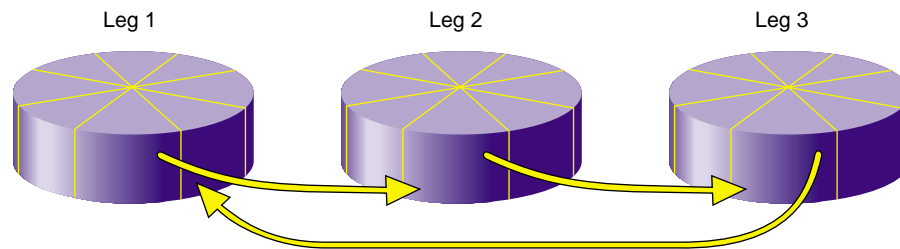


Figure 2-3 Reading Data from a Mirror with a Round-Robin Read Policy

Figure 2-4 illustrates how data is read from the legs of a mirror with a sequential read policy, showing that the different mirror members are not accessed for a single sequential I/O operation.

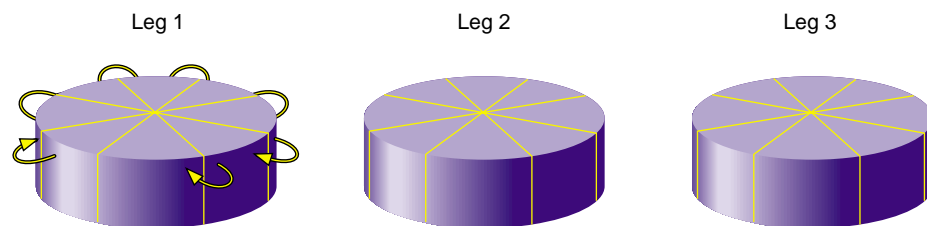


Figure 2-4 Reading Data from a Mirror with a Sequential Read Policy

After you have defined a mirror, you can change the read policy with the `change` command.

Primary Leg

You can specify whether a particular leg of a mirror is to be preferred for reading by marking it as a primary leg. After you have defined a mirror, you can redefine whether a leg is a primary leg with the `change` command.

The `-clean` Mirror Creation Option

When you create a mirror, you can use the `-clean` option of the `mirror` command to specify that the legs of the mirror do not need to be revived on creation. This option is useful when the legs of the mirror are already mirrored or when the mirror is new and all data will be written before being read.

The `-norevive` Mirror Creation Option

When you create a mirror, you can use the `-norevive` option of the `mirror` command to specify that the legs of the mirror do not need to be revived when the system boots. This option is useful when you are creating a mirror for a scratch filesystem such as `/tmp` or `swap`.

If you are creating a mirror for a swap filesystem that is in active use, you can specify `-norevive` (but not `-clean`). If you are creating a mirror for a swap filesystem that is not in active use, then you can specify both `-norevive` and `-clean` as there is no valid data on either mirror leg. If you are creating a mirror for a filesystem that has not yet been used (and therefore contains no valid data on either leg) then specifying `-clean` by itself eliminates the initial revive.

Creating Volumes

Use the `volume` command of the XVM Volume Manager to create an XVM logical volume explicitly. Volumes may also be created automatically when you create a volume element, as described in “Automatic Volume and Subvolume Creation” on page 41.

When you create a volume with the `volume` command, you can specify subvolumes to attach to the volume after it is created. When subvolumes are attached to a volume, the XVM Volume Manager enforces the rules of attachment described in “Attaching Volume Elements” on page 42.

Creating Subvolumes

Use the `subvolume` command of the XVM Volume Manager to create a subvolume explicitly. Subvolumes of type `data` may also be created automatically when you create a volume element, as described in “Automatic Volume and Subvolume Creation” on page 41.

When you create a subvolume with the `subvolume` command, you can specify the volume element to attach to the subvolume. The volume element to attach to the subvolume cannot be a volume or a subvolume. If you do not specify a volume element to attach, an empty subvolume is created.

When you create a subvolume with the `subvolume` command, you can specify the subvolume type. This can be a system-defined subvolume type or a user-defined subvolume type. There are two system-defined subvolume types:

<code>data</code>	A data subvolume
<code>log</code>	A log subvolume

There cannot be more than one subvolume having the same system-defined type under a volume, and you cannot specify a user-defined name for a system-defined subvolume type.

A user-defined subvolume type is in the range 16 through 255 (0 through 15 are reserved for system-defined types). System-defined subvolume types are used to associate an application-dependent type with a subvolume. You can specify more than one subvolume of a specific user-defined type under a volume.

Reorganizing Logical Volumes

As you create a logical volume, you can use the `attach` and `detach` commands of the XVM Volume Manager to organize and reorganize its elements. Additionally, after you have created volume elements you can reorganize the volume elements by using the `remake` command of the XVM Volume Manager. The `remake` command collapses holes in a volume element or rearranges pieces under a volume element. You can use a single `remake` command as a convenient alternative to executing a series of `attach` and `detach` commands.

Managing Logical Resources

After you have created your logical resources, you can perform the following tasks:

- Display volume elements
- Disable volume elements
- Bring volume elements online

- Make online changes to volume elements
- Save the logical volume configuration

The following subsections summarize these procedures.

Displaying Volume Elements

Use the `show` command of the XVM Volume Manager to display information about volume elements.

A volume element can be in one or more of the following states:

online	<p>The volume element is online. The volume element is properly configured. It is able to be opened, or it is already open.</p> <p>For physical volumes, an online state indicates that the system has successfully discovered the disk. After the system has discovered the disk, it will not move back to an offline state.</p>
offline	<p>The volume element is offline. No I/O can be done to that volume element. When a volume element is in this state, you will need to look over the topology of the volume element and note what state each piece of the volume element is in. There should always be at least one other state displayed that can help determine why the volume element is offline.</p> <p>For physical volumes, an offline state means that the system does not know where the physical volume resides. This can occur in a cluster where the system obtains information from another cell but does not have access to the disk.</p>
mediaerr	<p>The volume element has encountered at least one media error.</p>
inconsistent	<p>One or more pieces of the volume element may have changed since the last failure. An element may have been attached or detached, a missing piece may have come back into place, or the state may have changed.</p> <p>If a volume element is in an inconsistent state, you can use the <code>-v</code> option of the <code>show</code> command to display the timestamps of the pieces of the volume element. The inconsistent piece is the piece with the different timestamp. Choose the correct piece to keep and detach the other piece, or use the <code>remake</code> command to accept the current configuration.</p>

<code>tempname</code>	The volume element has a name which may not be persistent across reboots.
<code>reviving</code>	Associated with a mirror leg, a reviving state means that the mirror leg is a target of a revive. The data on this mirror leg is not valid until the revive completes.
<code>reviving:queued</code>	(mirror only) The mirror is targeted for a revive, but the revive has not started yet.
<code>reviving:XX%</code>	(mirror only) The system is in the process of reviving this mirror and is XX% complete.
<code>disabled</code>	The volume element has been disabled with the <code>change disable</code> command. The volume element must be explicitly enabled with <code>change enable</code> before it can be brought online.
<code>incomplete</code>	The volume element is missing one or more pieces. For all volume elements other than mirrors, the missing pieces will need to be attached or the volume element will need to be remade with the <code>remake</code> command before the volume element can be brought back online.
<code>pieceoffline</code>	The volume element has a piece that is offline. For volume elements other than mirrors, the offline pieces will need to be brought back online before the volume element can be brought online.
<code>open</code>	The volume element is part of an open subvolume.
<code>valid</code>	The volume element is up-to-date; the data is readable.
<code>clean</code>	The mirror leg has been created with <code>-clean</code> option of the <code>mirror</code> command to specify that the leg does not need to be revived on creation; it will be revived on subsequent boots.

Note: Once XVM has successfully read labels from a disk, that disk will not be marked offline due to I/O errors. A volume that contains a slice that is not available will be marked offline until initial discovery. After that, I/O errors on that disk will not change the offline/online status.

Disabling Volume Elements

You can use the `change` command of the XVM Volume Manager to manually disable a volume element. When you disable a volume element, no I/O can be done to that volume element until you explicitly enable the element, which you can also do with the `change` command. The object remains disabled until explicitly enabled, even across machine reboots.

Bringing a Volume Element Online

The system kernel may disable a volume element and take that element offline. This could happen, for example, when a mirror member shows an I/O error. You can use the `change` command of the XVM Volume Manager to bring the volume element back online.

Making Online Changes

You can insert a mirror or a concat above another volume element using the `insert` command of the XVM Volume Manager. This command can be used to grow a volume element or to add a mirror to a running system, because the volume element you are inserting can be part of an open subvolume and can have active I/O occurring.

You can remove a layer from a tree by using the `collapse` command of the XVM Volume Manager. Generally you use a `collapse` command to reverse a previous `insert` operation.

Saving and Regenerating XVM Configurations

To save an XVM logical volume configuration, use the `dump` command of the XVM Volume Manager to dump the commands to a file that will regenerate a configuration. This allows you to replace a disk in a running system and to regenerate the XVM configuration on the new disk without rebooting the system.

When you dump and regenerate a device, you do not regenerate the data on the disk you are replacing, but rather you regenerate the XVM configuration on the new disk.

Note that when you dump the commands to regenerate a volume element tree, you must separately and explicitly dump the commands to regenerate the physical volumes that the tree leads to, as described in “Reconstructing Volume Elements: Using the dump Command” on page 86.

It is possible for XVM labels to become corrupted. A good practice is to backup your XVM configuration using the `xvm dump` command any time it is changed just in case you need to recover from potential problems. You should save the `xvm dump` output into a filesystem other than the one being dumped.

For example, the following will dump all `xvm_labels` to the file `/var/xvm_config`:

```
xvm dump -topology -f /var/xvm_config phys/'' vol/''
```

Destroying Logical Resources

The following sections describe how to remove XVM elements, and how to remove configuration information from the kernel when an XVM disk becomes unavailable for I/O.

Deleting Volume Elements

Use the `delete` command of the XVM Volume Manager to delete a volume element. Parents of deleted volume elements remain and have open slots.

In general, if a volume element contains any attached children, it cannot be deleted. However, you can specify that either all of the children or all of the children but the slices be deleted by using the `-all` or `-nonslice` options, respectively. When you specify the `-nonslice` option, the slices are detached and a volume and data subvolume are automatically generated for the slices.

If the volume element you delete is part of an open subvolume, its deletion cannot cause the subvolume state to go offline. The `delete` command provides a `-force` option to override this restriction.

The swap partition of an XVM system disk cannot be deleted. This is to ensure that the swap partition cannot be deleted accidentally and cause a system panic.

Removing Configuration Information for Inaccessible Disks

When an XVM disk becomes physically unavailable, you may not be able to execute standard XVM configuration commands on logical volumes that include that disk. To recover from this situation, you can use the `reprobe` command of the XVM volume manager to remove previous configuration information from the kernel.

For information on using the `reprobe` command, see “Removing Configuration Information: Using the `reprobe` Command” on page 87.

The XVM Command Line Interface

This chapter describes the XVM command line interface (CLI) and the features it provides. The major sections in this chapter are as follows:

- “Using the XVM CLI”
- “Online Help for XVM CLI Commands” on page 57
- “XVM CLI Syntax” on page 58
- “Object Names in XVM” on page 59
- “XVM Device Directories and Pathnames” on page 64
- “Command Output and Redirection” on page 64
- “Safe Versus Unsafe Commands” on page 65

Using the XVM CLI

To use the XVM CLI, enter the following:

```
# xvm
```

If cluster services have been enabled when you enter this command, you should see the following XVM CLI prompt:

```
xvm:cluster>
```

This prompt indicates that the current domain is cluster, and any objects created in this domain can be administered by any node on the cluster.

You must start cluster services before you can see and access XVM cluster configuration objects.

You can specify the XVM domain when you bring up the XVM volume manager by using the `-domain` option of the XVM command:

```
# xvm -domain domain
```

The `domain` variable can be `local` or `cluster`. You may find this option useful for changing the domain of XVM command execution if you are writing a script in which you want to execute a single command in the local domain.

When you are running the XVM Volume Manager in the cluster domain, you can see and modify only the XVM physvols that are also in the cluster domain, even if you are running from the node that is the owner of a local physvol. To see and modify local disks, you either change your domain to `local` with the `set domain` command, or you use the `local:` prefix when specifying a physvol name. Similarly, when you are running the XVM Volume Manager in the local domain, you must change your domain to `cluster` or specify a `cluster:` prefix when specifying a physvol that is owned by the cluster.

For example, if you are running in the cluster domain but wish to see the XVM physical volumes in your local domain, you can use the following format:

```
xvm:cluster> show local:phys/*
```

Similarly, if you are running in the local domain but wish to see the XVM physical volumes in the cluster of which you are a member, you can use the following format:

```
xvm:local> show cluster:phys/*
```

For more information on XVM domains, see “XVM Domains” on page 34.

Once the command prompt displays you can enter XVM CLI commands to configure and manage your XVM logical volumes. These commands are executed interactively, as you supply them.

To configure XVM logical volumes, you need to be logged in as `root`. However, you can display logical volume configuration information even if you do not have `root` privileges. When you have finished executing XVM CLI commands, you return to your shell by entering the `exit` command. You can also use `bye` or `quit` as an alias for the `exit` command.

You can enter an individual XVM command directly from the shell by prefacing the command with `xvm`, as follows:

```
# xvm [command ...]
```

You can redirect a file of XVM commands into the XVM CLI just as you would redirect input into any standard UNIX tool, as follows:

```
# xvm < myscript
```

Alternately, you can enter the following:

```
# cat myscript | xvm
```

For information on using shell substitution to feed the output of one command into another, see “Command Output and Redirection” on page 64.

Online Help for XVM CLI Commands

The XVM CLI includes a `help` command, which you can use to display the syntax for any of the XVM CLI commands. A question mark (?) can be used as an alias for the `help` command.

The `help` command with no arguments produces a list of supported commands. The `help` command followed by an XVM command displays a synopsis of the XVM command you specify. You can precede the XVM command with the `-verbose` option to display full help that shows all of the commands options and examples.

The following command displays the synopsis for the `slice` command:

```
xvm:cluster> help slice
```

The following command displays the full help for the `slice` command:

```
xvm:cluster> help -verbose slice
```

The keywords you can use for the `help` command are any of the XVM CLI commands summarized on the `xvm(1M)` man page. In addition, you can enter the following help commands:

```
help names
```

Displays information on XVM object names

```
help regexp
```

Displays information on the regular expressions that you can use when specifying XVM object names

XVM CLI Syntax

The XVM CLI commands may be abbreviated to any unique substring. Command options may be abbreviated to any substring that is unique among the options supported by the command. Commands and options are not case sensitive.

For example, you can enter the following command:

```
xvm:cluster> volume -volname mainvol vol1/data vol2/log vol3/rt
```

Alternately, you can enter the following abbreviations:

```
xvm:cluster> vol -vol mainvol vol1/data vol2/log vol3/rt
```

Similarly, you can enter the following full command:

```
xvm:cluster> show -verbose slice/freds0
```

You can abbreviate the previous command as follows:

```
xvm:cluster> show -v slice/freds0
```

When you enter XVM commands, the following syntax rules and features apply:

- These keywords are reserved by the XVM CLI and may not be used to name objects: vol, stripe, concat, mirror, raid, slice, phys, unlabeled, subvol.
- Object names consist of alphanumeric characters and the period (.), underscore (_), and hyphen (-) characters.
- Object names cannot begin with a digit.
- XVM CLI tokens between <> characters are interpreted as comments.
- A backslash (\) at the end of a line acts as a continuation character.
- Blank lines and lines beginning with the pound sign (#) are ignored.
- An exclamation point (!) at the beginning of a command passes the command to the shell.

Object Names in XVM

All XVM objects except slices can have user-defined names supplied to them (if user-defined names are not supplied, default names will be generated). The names of XVM objects are limited to 32 characters in length and cannot begin with a digit.

With the exception of subvolumes, objects are specified using the object name. Subvolume objects must be specified by prefixing the subvolume name with its volume name followed by a slash (/). For example: `fred/data`. In this example, `fred` is the name of the volume and `data` is the name of the subvolume.

The following sections describe various ways XVM objects can be specified. The following topics are covered:

- XVM object specification
- Piece syntax
- XVM object name examples
- Regular expressions

XVM Object Specification

XVM objects are specified in a path-like syntax using one of the following forms where *objname* is the name of the object and *vepath* is a path leading from one volume element to another.

[*domain* :] [*type* /] *objname*

[*domain* :] [*type* /] *vspath*

The `domain` option can be `local` or `cluster`. You include the `domain` option when you are specifying an XVM object that is not in the current domain, as described in “Using the XVM CLI” on page 55.

Specifying a path component of “..” for a volume element indicates the parent of the volume element. For example, the following command displays the parent of slice `foos0`:

```
show slice/foos0/..
```

Because user-defined names are allowed, it is possible to have ambiguities in the XVM object namespace. When an ambiguous name is supplied to an XVM command and wildcards are not used, the command will generally produce an error message. For information on using wildcards, see “Regular Expressions” on page 63.

To remove ambiguity in an object name, an object name may be prefixed with an object type followed by a slash, as in the example `concat/concat1`. (If there are two objects of different object types named `concat1`, specifying `concat1` alone is not sufficient to identify the object.) Specifying an object type can also make name resolution faster by providing information about the type of object.

Note that unambiguous subvolumes are a 3-tuple: `subvol/volname/subvol_name`.

The following prefixes are recognized to specify object types. The `phys`, `unlabeled`, and `foreign` object types are described below.

Table 3-1 Prefixes Specifying Object Type

prefix	object type
<code>vol</code>	Volume volume element
<code>subvol</code>	Subvolume volume element
<code>concat</code>	Concat volume element
<code>stripe</code>	Stripe volume element
<code>mirror</code>	Mirror volume element
<code>slice</code>	Slice volume element

Table 3-1 Prefixes Specifying Object Type (continued)

prefix	object type
phys	Physvol
unlabeled	Unlabeled disk
foreign	Foreign disk

A *physvol* is a disk that has been labeled by the XVM Volume Manager as an XVM physical volume and has been probed by the system. For example, `phys/fred` refers to the XVM physvol named `fred`. The path portion of the name is the name that was given to the physvol at the time it was labeled.

An *unlabeled* disk is a disk that does not have an XVM label or has an XVM label but has not been probed by the XVM subsystem. A disk that was transferred to its current owner by means of the `give` or `steal` command is unlabeled until it has been probed, either explicitly with the `probe` command or during a system reboot.

The path portion of an unlabeled disk is the filesystem path to the volume partition. This can be an explicit path (for example, `unlabeled/hw/rdisk/dks0d4vol`) or a relative path (for example, `unlabeled/dks0d4vol`). SAN disk paths have multiple components (for example, `unlabeled/2000006016fe057a/lun4vol/c11p0`).

A *foreign* disk is an XVM disk that cannot be administered by the current owner, either because the disk is owned by another node or another cluster. The format of the path portion of a foreign disk is the same as the path portion of an unlabeled disk.

Piece Syntax

XVM volume elements can also be specified using a path-like syntax where the components of the path are volume element names or piece numbers under the parent. For example: `vol/fred/data/concat0/phys0` refers to the `ve phys0`, whose parent is `concat0`, which is the data subvolume of volume `fred`. Additionally, `concat0/0` refers to the zero child (piece) of the `ve concat0`. The piece syntax is helpful when you want to target a volume element without knowing its name.

Figure 3-1 shows the layout of an XVM logical volume with system-generated names.

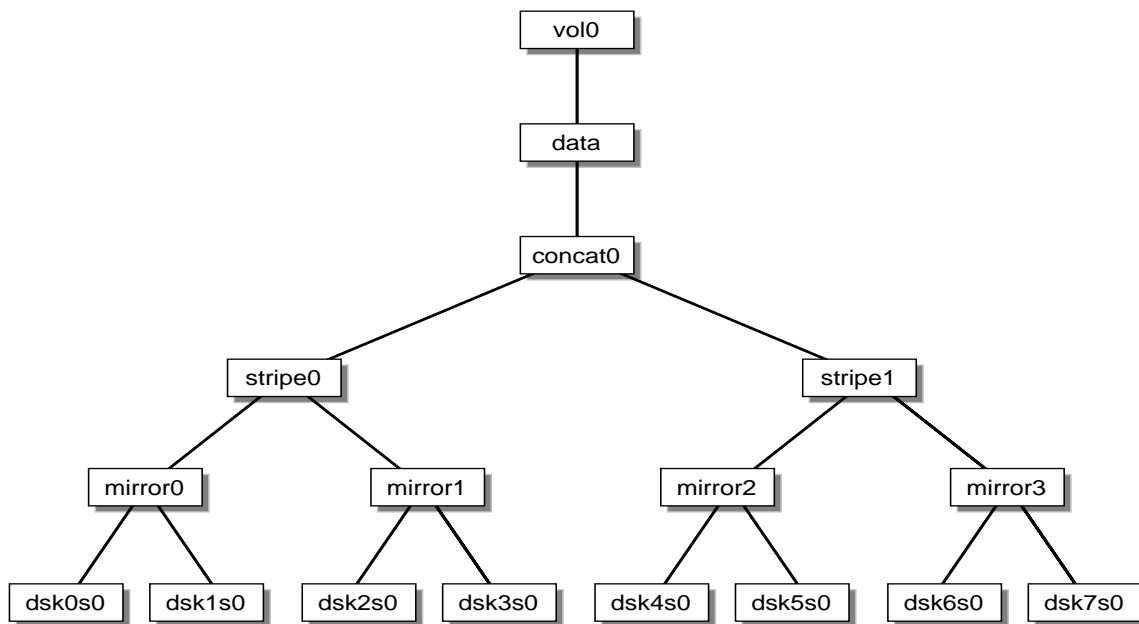


Figure 3-1 XVM Logical Volume with System-Generated Names

Table 3-2 shows examples of how you can use piece syntax to specify individual volume elements in the XVM logical volume illustrated in Figure 3-1.

Table 3-2 Specifying Logical Volume Elements Using Piece Syntax

XVM object	Alternate Object Specification
concat/concat0	vol0/data/0
stripe/stripe0	vol0/data/0/0
stripe/stripe1	vol0/data/0/1
mirror/mirror0	vol0/data/concat0/stripe0/0 vol0/data/0/0/0
mirror/mirror2	vol0/data/concat0/stripe1/0 vol0/data/0/1/0
slice/dsk6s0	vol0/data/0/1/1/0 vol0/data/concat0/1/mirror3/0

XVM Object Name Examples

The following examples show how a variety of XVM objects can be specified.

<code>vol0</code>	The object named <code>vol0</code>
<code>unlabeled/dks0d4</code>	The unlabeled disk on controller 0, drive 4
<code>mirror/mirror6</code>	The mirror volume element named <code>mirror6</code>
<code>concat0/0</code>	The leftmost piece of <code>concat0</code>
<code>vol0/data/0</code>	The child of the data subvolume of volume <code>vol0</code>
<code>stripe0/freds0</code>	The volume element named <code>freds0</code> under the stripe <code>stripe0</code>
<code>unlabeled/2000006016fe0ed0/lun3vol/clip0</code>	The SAN disk whose volume partition path is <code>/dev/rdisk/2000006016fe0ed0/lun3vol/clip0</code>
<code>foreign/dks5d43vol</code>	The disk <code>dks5d43vol</code> , which cannot be administered by the machine displaying this object name

Regular Expressions

Regular expressions can be used in specifying object names in XVM CLI commands. The wildcard (*), ?, and [] characters are recognized and have their standard regular expression meanings as supported through the `fnmatch(3G)` function. Regular expressions in an XVM CLI command are limited to the rightmost component of an XVM object path.

The following examples show how regular expressions can be used in XVM CLI command:

<code>*</code>	Matches all objects
<code>vol/*</code>	Matches all volumes
<code>slice/*s0</code>	Matches all slices ending in <code>s0</code>
<code>concat0/*</code>	Matches all children of <code>concat0</code>

subvol/log* Matches all log subvolumes
fred* Matches all objects beginning with fred
unlabeled/dks[34]d* Matches all unlabeled disks on controllers 3 and 4

Specifying a volume element path ending in / is equivalent to specifying a volume element path ending in /*.

A volume element path which consists of an object type keyword is equivalent to specifying the keyword followed by /*. For example, a show slice command is equivalent to a show slice/* command.

XVM Device Directories and Pathnames

Under Linux, block devices for XVM logical volumes used for a host's local volume are contained in the /dev/lxvm directory. For compatibility with earlier releases, support is also provided for the /dev/xvm/local/vol/*volname*/data/block directory.

For information on names of objects within XVM logical volumes, see "Object Names in XVM" on page 59.

Command Output and Redirection

In general, commands that create or manipulate objects will print out the name of the created or target object upon successful completion, as in the following example:

```
xvm:cluster> concat -tempname slice/wilmas0 slice/barneys0  
</dev/cxvm/vol0> concat/concat0
```

You can use shell substitution to feed the output of one command into another. For example, under the Korn shell the following command would create a concatenated volume element with a volume name of fred and the physvols phys1 and phys2 as the components.

```
$ xvm concat -volname fred $(xvm slice -all phys1) $(xvm slice -all \  
phys2)
```

Under `cs`h or `sh`, the syntax for the command is as follows:

```
$ xvm concat -volname fred `xvm slice -all phys1` `xvm slice -all \
phys2`
```

Commands that fail, or for which the manipulated object does not make sense (such as `delete`, for example), do not print out the target object name.

As shown, commands that create or modify volume elements also display the subvolume block-special name that the target `ve` belongs to inside of `<>` symbols. For example, the command `slice -all phys1` produces the following output (if successful), where `slice/phys1s0` is the name of the slice created, and `/dev/cxvm/phys1s0` is a path to the subvolume block-special that can be opened to gain access to the slice:

```
</dev/cxvm/phys1s0> slice/phys1s0
```

Tokens that appear inside of `<>` symbols are treated as comments by the CLI. This ensures that even though a command that creates a volume element displays the block-special name, that output is inside of `<>` symbols and is ignored by the CLI when you feed the output of one command into another.

Safe Versus Unsafe Commands

The XVM commands can be categorized as safe or unsafe. An unsafe command is one that will in some way affect the address space of the subvolume that the `ve` is under, such as detaching or deleting a child of a `concat` `ve`. Safe commands do not affect the address space of the subvolume, such as detaching or deleting all but the last child of a `mirror` `ve` (detaching or deleting the last child is unsafe).

Safe commands can always be issued regardless of the open state of the effected subvolume, whereas unsafe commands can be issued only if the subvolume is not open. Mounted subvolumes are always open, however a subvolume may also be open without being mounted, for example if an application is accessing the raw subvolume.

Unsafe commands to open subvolumes will result in an error by default, but certain commands have a `-force` option to override that behavior. Conversely, certain commands have a `-safe` option, which will enforce the safe checks even if the subvolume is not open.

XVM Administration Commands

This chapter summarizes the `xvm` command line interface (CLI) commands and provides examples of each command. A full description of the syntax of each individual command is available through the `help` command, as described in “Online Help for XVM CLI Commands” on page 57.

This chapter includes sections on the following topics:

- “Physical Volume Commands”
- “Logical Volume Commands” on page 74

Physical Volume Commands

You can use the following commands to create, manage, and delete XVM physical volumes:

<code>set</code>	Changes the default XVM domain
<code>label</code>	Assigns disks to the XVM Volume Manager
<code>show</code>	Displays XVM physical volumes
<code>change</code>	Modifies XVM physical volumes
<code>probe</code>	Probes an XVM physical volume
<code>dump</code>	Regenerates XVM physical volumes
<code>give</code>	Changes the domain of an XVM physical volume
<code>steal</code>	Changes the domain of an XVM physical volume when the node or cluster that currently owns the physical volume is unable to execute the <code>give</code> command
<code>unlabel</code>	Removes disks from the XVM Volume Manager
<code>foswitch</code>	Changes the path used to access a physical disk

These commands are summarized in the following sections.

Changing the Current Domain with the `set` Command

You use the `set` command to change the current XVM domain while executing XVM CLI commands. The current domain can be `local` or `cluster`. If the current domain is `local`, the XVM objects you are creating belong to the current node you are running from. If the current domain is `cluster`, the XVM objects you are creating belong to the cluster that the current node belongs to. The current domain is displayed as part of the `xvm` prompt, which appears as `xvm:cluster>` or `xvm:local>`. You can also see the current XVM domain by executing the `set` command without specifying the `local` or `cluster` parameter.

You cannot set the domain to `cluster` if cluster services are not started.

The following example changes the current domain from `cluster` to `local`:

```
xvm:cluster> set domain local
```

For information about XVM domains, see “XVM Domains” on page 34.

Assigning Disks to the XVM Volume Manager with the `label` Command

You use the `label` command to assign a disk to the XVM Volume Manager. The `label` command writes out or modifies an XVM physical volume label on a disk. In a clustered environment, you can label only the disks that are attached to the system you are working from.

When you label a disk as an XVM physical volume, the first four bytes of logical block one, when represented as ASCII characters, yield `xlab`. This enables you to determine whether a disk is an XVM physical volume even if you are not running the XVM Volume Manager.

Use the `-name` option to assign a name to the XVM physical volume. If you do not specify a name, the default name will be the base name of the unlabeled disk path (for example, `dfs0d1`). If you specify a name when assigning multiple disks to XVM, the supplied name acts as a prefix for each physical volume name, with a unique numeric suffix added. If you do not specify a name when assigning multiple disks, the unlabeled disk path is used as the prefix for each physical volume name.

When you assign a disk to the XVM Volume Manager, the disk is an XVM option disk by default.

Displaying Physical Volumes with the show Command

You use the `show` command to display information about XVM objects.

The following example shows the results of a `show` command with the `-extend` option enabled to show all the existing physical volumes and their device paths:

```
xvm: cluster> show -extend phys
phys/coreyz      138737184 online,cluster (/dev/rdisk/2000006016fe1f95/lun0vol/c3p0)
phys/jansad      17779016  online,cluster (/dev/rdisk/dks2d19vol)
```

The following example shows the results of a `show` command executed on a specific physical volume, with the `-verbose` option enabled:

```
xvm:cluster> show -v phys/dd5d
XVM physvol phys/dd5d
=====
size: 71670988 blocks sectorsize: 512 bytes state: online,local,accessible
uuid: 5eb0575d-da64-46fd-871c-766c89628ea6
system physvol: no
path manager device: /dev/pm/SGI-ST336753FC-3HX2Y7CJ000075138GK1 on host puffin
using paths:
    /dev/disk/by-path/pci-0002:00:02.1-fc-0x21000011c61ddd5d:0x0000000000000000 <sdbn 68:16> affinity=1 <current>
    /dev/disk/by-path/pci-0002:00:02.1-fc-0x22000011c61ddd5d:0x0000000000000000 <sdbm 68:0> affinity=2
    /dev/disk/by-path/pci-0002:00:02.0-fc-0x22000011c61ddd5d:0x0000000000000000 <sdu 65:64> affinity=2
    /dev/disk/by-path/pci-0002:00:02.0-fc-0x21000011c61ddd5d:0x0000000000000000 <sdv 65:80> affinity=1
Disk has the following XVM label:
Clusterid: 0
Host Name: puffin
Disk Name: dd5d
Magic: 0x786c6162 (balx) Version 2
Uuid: 5eb0575d-da64-46fd-871c-766c89628ea6
last update: Tue Aug 2 14:20:20 2011
state: 0x91<online,local,accessible> flags: 0x0<idle>
secbytes: 512
label area: 8157 blocks starting at disk block 35 (10 used)
user area: 71670988 blocks starting at disk block 8192

Physvol Usage:
Start      Length      Name
-----
0          71670976   slice/dd5ds0
71670976   12         (unused)

Local stats for phys/dd5d since being enabled or reset:
-----
stats collection is not enabled for this physvol
```

The `show` command also allows you to display information about unlabeled disks. The following example shows the results of a `show` command executed on an unlabeled disk:

```
xvm:cluster> show -v unlabeled
Unlabeled disk unlabeled/dev/pm/SGI-ST336753FC-3HX2XYJ100007513Q1PU
=====
using paths:
    /dev/disk/by-path/pci-0002:00:02.1-fc-0x21000011c61dd97e:0x0000000000000000 <sdbn 68:80> affinity=none <current>
    /dev/disk/by-path/pci-0002:00:02.1-fc-0x22000011c61dd97e:0x0000000000000000 <sdbq 68:64> affinity=none
    /dev/disk/by-path/pci-0002:00:02.0-fc-0x21000011c61dd97e:0x0000000000000000 <sdz 65:144> affinity=none
    /dev/disk/by-path/pci-0002:00:02.0-fc-0x22000011c61dd97e:0x0000000000000000 <sdv 65:128> affinity=none
```

The `show` command can display information about disks that the XVM Volume Manager sees as foreign disks. This can be useful if you find yourself in a situation where you need to use the `steal` command to take control of an XVM physvol from its current owner. In this situation, you may need to determine the owner of a disk that you cannot read as a physvol, since it appears as a foreign disk to you. The output of the `show` command will show the owner as the “Host Name” of the physvol. The following example shows all

foreign disks, using the `-extend` option to display the name of the host or cluster that owns the disk and name of the physical volume on that host or cluster. In this example, sample output is shown.

```
xvm:cluster> show -v foreign
Foreign disk foreign//dev/pm/SGI-ST336753FC-3HX2XSST00007513R1LT
=====
using paths:
  /dev/disk/by-path/pci-0002:00:02.1-fc-0x22000011c61dd974:0x0000000000000000 <sdbx 68:96> affinity=none <current>
  /dev/disk/by-path/pci-0002:00:02.1-fc-0x21000011c61dd974:0x0000000000000000 <sdbt 68:112> affinity=none
  /dev/disk/by-path/pci-0002:00:02.0-fc-0x22000011c61dd974:0x0000000000000000 <sdaa 65:160> affinity=none
  /dev/disk/by-path/pci-0002:00:02.0-fc-0x21000011c61dd974:0x0000000000000000 <sdab 65:176> affinity=none
Disk has the following XVM label:
Clusterid: 0
Host Name: clux2
Disk Name: clux4
Magic: 0x786c6162 (balx) Version 2
Uuid: 47alfaf5-7a9e-4f69-848d-a94de981f57d
last update: Wed Aug 31 17:34:08 2011
state: 0xa1<online,cluster,accessible> flags: 0x0<idle>
secbytes: 512
label area: 8157 blocks starting at disk block 35 (10 used)
user area: 71670988 blocks starting at disk block 8192
```

The following example executes the `show` command on `dks5d46`, which is a disk that is foreign to the current node, which is `hugh2`:

```
hugh2 1# xvm
xvm:cluster> show dks5d46
foreign/dks5d46vol * private
xvm:cluster> show -v dks5d46
Foreign disk foreign/dks5d46vol
=====
volume alias: /dev/rdsk/dks5d46vol
volume full path:
/hw/module/2/slot/io7/fibre_channel/pci/0/scsi_ctlr/0/target/46/lun/0/disk/volume/char
Disk has the following XVM label:
Clusterid: 0
Host Name: hugh
Disk Name: disk5
Magic: 0x786c6162 (xlab) Version 2
Uuid: 530138fd-0096-1023-8a7a-0800690592c9
last update: Thu Sep 16 10:25:58 1999
state: 0x11<online,private> flags: 0x0<idle>
secbytes: 512
label area: 1024 blocks starting at disk block 3072 (10 used)
user area: 17779016 blocks starting at disk block 4096
```

Note that in this example, the host name of the foreign disk is `hugh`, the disk's current owner.

For information on foreign disks, see “XVM Domains” on page 34. For information on the `steal` command, see “Removing Disks from the XVM Volume Manager with the `unlabel` Command” on page 73. The `steal` command should be used only when ownership cannot be changed using the `give` command.

The `show` command output indicates whether a physical volume has no physical connection to the system and would return an I/O error when read or write activity is attempted anywhere on the volume. In the following examples, the physical volume `lmctst` has no physical connection on this system.

```
xvm:cluster> show vol
vol/c1                0 online
vol/con1              0 offline,no physical connection
vol/lmcl              0 online,no physical connection
vol/lmctst3s2         0 online
vol/lmctst3s3         0 online
vol/lmctst3s4         0 online

xvm:cluster> show -top vol/con1
vol/con1              0 offline,no physical connection
  subvol/con1/data    400000 offline,pieceoffline
    concat/concat9    400000 offline,tempname,incomplete
      slice/lmctstsl  200000 online
        (empty)      * *

xvm:cluster> show -top vol/lmcl
vol/lmcl              0 online,no physical connection
  subvol/lmcl/data    200000 online
    mirror/mirror7    200000 online,tempname
      slice/lmctst2s1  200000 online
        slice/lmctsts0 200000 online
```

The `show` command can also be used to display information about other volume elements. For more examples of the `show` command see “Displaying Volume Elements: Using the `show` Command” on page 85.

Modifying Physical Volumes with the change Command

You can use the `change` command to change the name of an XVM physical volume and to change the state of statistics collection (to on, off, or reset). For examples of the `change` command see “The change Command” on page 79.

Probing a Physical Volume with the probe Command

The `probe` command probes a disk with XVM labels so that the system is able to recognize the disk as an XVM disk. Disks are probed automatically when the system is booted, but you must manually execute a `probe` command when you add an XVM disk to a running system. If you execute the `probe` command on a disk that has not been previously labeled, an error is returned.

The disk to be probed must first be available in the hardware inventory. Use the `scsiadminswap(1M)` command to introduce a disk to the system.

The following example probes drive 4 on controller 0:

```
xvm:cluster> probe dks0d4
```

The following example re-probes the XVM physical volume named `fred`:

```
xvm:cluster> probe fred
```

The following example probes all SCSI drives:

```
xvm:cluster> probe dks*
```

Regenerating XVM Physical Volumes using the dump command

You use the `dump` command to dump the commands to a file that will regenerate an XVM physical volume. For examples of the `dump` command, see “Reconstructing Volume Elements: Using the dump Command” on page 86.

Removing Disks from the XVM Volume Manager with the unlabel Command

You use the `unlabel` command to remove an XVM label from a disk so that the disk is no longer an XVM disk. This restores the original partitioning scheme to the disk. In a

clustered environment, you can unlabel only a disk that is attached to the system you are working from.

The following example removes the XVM label from the XVM physical volume named `phys1`:

```
xvm:cluster> unlabel phys1
```

The following example forcibly unlabels `phys1`, first deleting any slices that may exist:

```
xvm:cluster> unlabel -force phys1
```

Logical Volume Commands

The following sections describe the `xvm` commands you use to create, modify, display, reconstruct, and delete volume elements.

Creating Volume Elements

There are separate `xvm` commands to create the following logical volume elements:

- Slices
- Concats
- Mirrors
- Stripes
- Subvolumes
- Volumes

These commands are summarized in the following sections.

The slice Command

The `slice` command creates slices from specified block ranges of XVM physical volumes.

The following example creates one slice covering the whole usable space of the XVM physical volume `phys1`:

```
xvm:cluster> slice -all phys1
```

The following example creates four equal-sized slices covering the XVM physical volume `phys1`:

```
xvm:cluster> slice -equal 4 phys1
```

The following example creates a slice starting with block 5,000 with a length of 100,000 blocks:

```
xvm:cluster> slice -start 5000 -length 100000 phys1
```

The following example divides the 100,000-block chunk beginning at block 5,000 into 4 equal-sized slices:

```
xvm:cluster> slice -start 5000 -length 100000 -equal 4 phys1
```

The concat Command

The `concat` command creates a volume element that concatenates all of its child volume elements into one address space. When you create a `concat`, you must specify whether you are naming the generated volume to which it is attached or whether the system will generate a temporary volume name.

The following example concatenates the slices `freds0` and `wilmas0` into a larger address space. The created `concat` volume element has a system-generated temporary name and is contained in a volume with a system-generated temporary name:

```
xvm:cluster> concat -tempname slice/freds0 slice/wilmas0
```

The following example also concatenates the slices `freds0` and `wilmas0` into a larger address space. It explicitly names the resulting `concat` `myconcat` and the volume it belongs to `concatvol`:

```
xvm:cluster> concat -vename myconcat -volname concatvol slice/freds0 \  
slice/wilmas0
```

The mirror Command

The `mirror` command creates a volume element that mirrors all of its child volume elements. When you create a `mirror`, you must specify whether you are naming the generated volume to which it is attached or whether the system will generate a temporary volume name.

When you create a mirror that has more than one piece, a message is written to the `SYSLOG` indicating that the mirror is reviving. This indicates that the system is beginning the process of mirroring the data. Another message is written to the `SYSLOG` when this process is complete. For large mirror components, this may take a long time. You cannot halt a mirror revive once it has begun except by detaching all but one of the pieces of the mirror.

Should the revive fail for any reason, a message will be written to the system console as well as to the `SYSLOG`. For more information, see “Mirror Revives” in Chapter 8.

When you create a mirror, you can define a read policy and a primary leg for the mirror. These features are described in “Creating Mirrors” on page 45.

When you create a mirror, you can specify that the mirror does not need to be resynchronized when it is created. Alternately, you can specify that the mirror will never need to be resynchronized; this is an option that is useful when you are mirroring a scratch filesystem. These features are described in “Creating Mirrors” on page 45.

The components of a mirror do not have to be identical in size, but if they are not there will be unused space in the larger components.

The following example creates a mirror whose members are the slices `freds0` and `wilmas0`. The volume that the mirror will be associated with will be named `mirvol`.

```
xvm:cluster> mirror -volname mirvol slice/freds0 slice/wilmas0
```

The following example creates a mirror, with members `slice/freds0` and `slice/wilmas0` and volume name `newmirvol`. In this example, a revive will not be initiated when the mirror is created.

```
xvm:cluster> mirror -volname newmirvol -clean slice/freds0 slice/wilmas0
```

The following example creates an empty mirror with a sequential read policy. To make the mirror usable, the members of the volume element will have to be explicitly attached using an `attach` command. This command creates a mirror with a system-generated name that is contained in a volume with a system-generated name.

```
xvm:cluster> mirror -tempname -rpolicy sequential
```

The following example creates a two-member mirror with a primary member named `freds0`. All reads will be directed to `freds0`, with writes going to both members. This command creates a mirror with a system-generated name that is contained in a volume with a system-generated name.


```
xvm:cluster> mirror -tempname -primary slice/freds0 slice/freds0 slice/wilmas0
```

The stripe Command

The `stripe` command creates a volume element that stripes a set of volume elements across an address space. When you create a stripe, you must specify whether you are naming the generated volume to which it is attached or whether the system will generate a temporary volume name.

It is legal to create a stripe that consists of volume elements of unequal size, although this may leave some space unused.

A stripe units has the restriction that it must be a multiple of 32 512-byte blocks. You can remove this restriction with the `-noalign` flag.

The actual size of the stripe volume element depends on the stripe unit size and the size of the volume elements that make up the stripe. In the simplest case, the volume elements are all the same size and are an even multiple of the stripe unit size. For example, if the stripe unit is 128 512-byte blocks (the default stripe unit size), and you create a stripe consisting of two slices that are each 256,000 blocks, all the space of each of the slices is used. The stripe size is the full 512,000 blocks of the two slices.

On the other hand, if two slices that make up a stripe are each 250,000 blocks and the stripe unit is 128 blocks, then only 249,984 of the blocks on each slice can be used for the stripe and the size of the stripe will be 499,968 blocks. This situation may arise when you create the slices on a disk by dividing the disk equally, or use the entire disk as a slice, and do not coordinate the resulting stripe size with the stripe unit size.

Even if one of the two slices that make up the two-slice stripe in the second example is 256,000 blocks (while the other is 250,000 blocks), the stripe size will be 499,968 blocks, since the same amount of space in each volume element that makes up the slice is used.

The general formula for determining what the stripe size will be is the following, where `stripe_width` is the number of volume elements that make up the stripe:

$$\text{stripe_size} = (\text{smallest_stripe_member} / \text{stripe_unit}) * \text{stripe_unit} * \text{stripe_width}$$

Note that this formula uses integer arithmetic.

You can view the stripe unit of an existing stripe with the `show -extend (or -v) stripe` command (where `stripe` is the name of the existing stripe).

For information on configuring stripes that span two host bus adaptors, see Chapter 5, “XVM Failover.”

The following example stripes the slices `freds0` and `wilmas0`. The volume that the stripe is associated with will be named `stripedvol`.

```
xvm:cluster> stripe -volname stripedvol slice/freds0 slice/wilmas0
```

The following example stripes the mirrors `mirror0` and `mirror1` using a stripe unit size of 512 blocks:

```
xvm:cluster> stripe -tempname -unit 512 mirror[01]
```

The following example creates an empty stripe with room for four slices. Four volume elements must be attached to the stripe before it will come online.

```
xvm:cluster> stripe -tempname -pieces 4
```

The subvolume Command

The `subvolume` command creates a subvolume and, optionally, attaches a specified volume element to the subvolume. The volume element attached to the subvolume cannot be a volume or another subvolume.

When you create a subvolume, you must specify whether you are naming the generated volume to which it is attached or whether the system will generate a temporary volume name.

You can create a subvolume of a system-defined type of `data`, `log`, or you can create a subvolume of a user-defined type. You cannot specify a subvolume name for a subvolume of a system-defined type.

The following example creates a `log` subvolume and attaches `concat0` to it. The volume associated with this subvolume will be named `myvol`.

```
xvm:cluster> subvolume -volname myvol -type log concat0
```

The following example creates a subvolume and attaches `concat0` to it, setting the `uid` and `mode` of the block and character special files corresponding to the subvolume:

```
xvm:cluster> subvolume -tempname -uid 1823 -mode 0644 concat0
```

The following example creates a subvolume with a user-defined type of `100`:

```
xvm:cluster> subvolume -tempname -type 100 concat0
```

The volume Command

The `volume` command creates an XVM volume and, optionally, attaches specified subvolumes to the volume.

The following example creates an empty volume named `fred`:

```
xvm:cluster> volume -volname fred
```

The following example groups `data` and `log` subvolumes under a volume. The created volume has a system-generated temporary name.

```
xvm:cluster> volume -tempname vol0/data vol1/log
```

Modifying Volume Elements

The XVM Volume manager provides the following commands to modify volume elements after you have created them:

- `change`
- `attach`
- `detach`
- `remake`

These commands are described in the following sections.

The change Command

The `change` command changes the attributes of an XVM physical volume or volume element that you have previously defined. You can change a variety of attributes of an XVM object using the `change` command, depending on the object. You can use the `change` command to enable statistics collection for an object, to bring a volume element back online that the kernel has disabled, and to manually disable and re-enable a volume element.

You can use the `change` command to rename an existing object. The name you give an object with this command remains persistent across reboots. You cannot change the name of a slice.

For a full list of the attributes that you can modify using the `change` command, see the help screen for this command.

The following example enables statistics for XVM physical volume `pvol0` and the data subvolume of `vol/fred`:

```
xvm:cluster> change stat on phys/pvol0 vol/fred/data
```

The following example resets statistics for all objects that have statistics enabled:

```
xvm:cluster> change stat reset *
```

The attach Command

The `attach` command attaches an existing volume element to another existing volume element. For information on the restrictions that the XVM Volume Manager imposes on attachments, see “Attaching Volume Elements” on page 42.

You can specify where to attach a volume element. If you do not explicitly indicate where to attach a volume element, the source volume element will be attached to the first (leftmost) hole in the target volume element. If there are no holes, the source volume element will be appended to the end (right).

You can attach multiple source volume elements to a single target volume element by using the `attach` command. When attaching multiple source volume elements, the position you specify for the attachment applies only to the first volume element; remaining volume elements will be placed to the right, filling holes or appending.

When you attach multiple source volume elements to a single target volume element, they are attached one at a time, in turn. If an attach in the list fails, XVM attempts to restore the volume elements to their previous parents. If a volume element cannot be restored, a warning message is generated and manual intervention is needed.

The following example attaches the slice `freds0` to `concat0` at the first available position:

```
xvm:cluster> attach slice/freds0 concat0
```

The following example attaches all subvolumes of `vol0` to `vol1`:

```
xvm:cluster> attach vol0/* vol1
```

The following example attaches `slice/freds0` to `concat0`, performing checks as if `concat0` and `slice/freds0` were part of open subvolumes, even if they are not:

```
xvm:cluster> attach -safe slice/freds0 concat0
```

The detach Command

The `detach` command detaches a volume element from its parent. When you detach a volume element, a new volume (and possibly data subvolume) will be created, just as a volume is created when you create a volume element. You can name the generated volume explicitly, or you can specify that the volume be automatically generated with a temporary name. A subvolume of type `data` is automatically generated for the volume element you are detaching (unless the volume element you are detaching is itself a subvolume of a different type).

You cannot detach the last valid piece of an open mirror from that mirror, since this will cause the mirror to go offline.

If the volume element you detach is part of an open subvolume, its detachment cannot cause the subvolume state to go offline. The `detach` command provides a `-force` option to override this restriction and a `-safe` option to impose this restriction even if the subvolume is not open.

The following example detaches the volume element `concat0` from its parent. The volume that `concat0` is associated with after the detach will be named `fred`.

```
xvm:cluster> detach -volname fred concat0
```

The following example detaches `concat0`, even if it is part of an open subvolume, and the subvolume would go offline as a result:

```
xvm:cluster> detach -force -tempname concat0
```

The following example detaches `concat0`, but ensures that the detachment will not cause the subvolume to go offline, even if the corresponding subvolume is not currently open:

```
xvm:cluster> detach -safe -tempname concat0
```

The remake Command

The `remake` command reorganizes volume elements in an XVM logical volume by collapsing holes in a volume element or by rearranging pieces under a volume element. You can use a single `remake` command as a convenient alternative to executing a series of `attach` and `detach` commands.

When you rearrange the pieces in a volume element, you can specify one of the following rearrangement methods with an option of the `remake` command:

<code>swap</code>	Swaps the positions of two volume elements under a volume element
<code>reorder</code>	Reorders the children under a volume element

The following example collapses any holes in the `ve concat0`:

```
xvm:cluster> remake concat0
```

The following example reorganizes `concat0`, swapping pieces 0 and 1:

```
xvm:cluster> remake concat0 swap concat0/0 concat0/1
```

The following example reorganizes `concat0`, reversing the order of its 3 pieces:

```
xvm:cluster> remake concat0 reorder concat0/2 concat0/1 concat0/0
```

Modifying Volume Elements on a Running System

The XVM Volume Manager allows you to modify volume elements on a running system using the `insert` command and the `collapse` command, as described in the following sections.

The insert command

The `insert` command inserts a mirror or a concat volume element above another volume element. You cannot insert a volume element above a volume or a subvolume.

The `insert` command allows you to grow a volume element on a running system by inserting a concat or to add mirroring on a running system by inserting a mirror. The volume element you are growing or mirroring can be part of an open subvolume and can have active I/O occurring.

For example, if you begin with a simple logical volume named `tinyvol` that contains a single slice named `freds0`, the topology of the logical volume is as follows:

```
xvm:cluster> show -top tinyvol
vol/tinyvol          0 online
  subvol/tinyvol/data 177792 online,open
    slice/freds0      177792 online,open
```

You can insert a concat in the volume above the slice:

```
xvm:cluster> insert concat slice/freds0
</dev/cxvm/tinyvol> concat/concat0
```

The topology of the logical volume is now as follows:

```
xvm:cluster> show -top tinyvol
vol/tinyvol                0 online
  subvol/tinyvol/data      177792 online,open
    concat/concat0        177792 online,tempname,open
      slice/freds0        177792 online,open
```

You can now grow the volume by attaching another slice to the concat.

The following example inserts a one-member mirror over the volume element `concat0`. This allows other members to be attached and `concat0` to be mirrored without having to take it offline.

```
xvm:cluster> insert mirror concat0
```

The following example inserts a one-member concat above the slice `freds0`. This allows other members to be attached, and allows the corresponding subvolume to be grown without having to take it offline.

```
xvm:cluster> insert concat slice/freds0
```

The collapse Command

The `collapse` command removes a layer from a logical volume tree by collapsing a volume element, linking the child of the volume element to the volume element's parents.

The `collapse` command can be used to collapse a mirror or concat in an open subvolume. Generally, this is used to reverse a previous insert operation.

For example, the following command sequence inserts a mirror above an existing concat named `concat1` and then displays the topology of the resulting logical volume:

```
xvm:cluster> insert mirror concat1
</dev/cxvm/vol9> mirror/mirror2
xvm:cluster> show -top vol9
vol/vol9                0 online,tempname
  subvol/vol9/data      5926338 online
    mirror/mirror2      5926338 online,tempname
```

```
concat/concat1          5926338 online,tempname
  slice/pebbless0       2963169 online
  slice/bettys0         2963169 online
```

The following sequence of commands reverses this insert operation by collapsing the mirror and displays the topology of the resulting logical volume:

```
xvm:cluster> collapse mirror/mirror2
xvm:cluster> show -top vol9
vol/vol9
  subvol/vol9/data      0 online,tempname
    concat/concat1     5926338 online
      slice/pebbless0  2963169 online
      slice/bettys0    2963169 online
```


Displaying Volume Elements: Using the show Command

The `show` command displays information about volume elements as well as physical volumes and unlabeled disks. This command includes an `-extend` option, which shows additional information about XVM physical volumes, slices, and stripes, and the command also includes a `-verbose` option, which displays as much information as possible about the indicated object.

The following example shows name, size, and state information for the object named `concat0`:

```
xvm:cluster> show concat0
```

The following example shows all XVM slices:

```
xvm:cluster> show slice/*
```

The following example shows the names of all XVM volumes:

```
xvm:cluster> show -name vol/*
```

The following example shows the names of all unlabeled disks on dks controller 0:

```
xvm:cluster> show -name unlabeled/dks0*
```

The following example shows statistics for the XVM physical volume named `fred`:

```
xvm:cluster> show -stat phys/fred
```

The following example shows the topology of the volume `vol/myslice`. This example uses the `-extend` option to display the stripe unit size of the stripes in the volume as well as the name and device path of the physical volumes associated with any slices in the volume. In this example, sample output is shown.

```
xvm:cluster> show -topology -extend vol/myslice
vol/myslice                0 online
  subvol/myslice/data      17778688 online
    stripe/stripe0        17778688 online,tempname (unit size: 128)
      slice/jansads0      4444754 online (jansad:/dev/rdisk/dks2d19vol)
      slice/jansads1      4444754 online (jansad:/dev/rdisk/dks2d19vol)
      slice/jansads2      4444754 online (jansad:/dev/rdisk/dks2d19vol)
      slice/jansads3      4444754 online (jansad:/dev/rdisk/dks2d19vol)
```

Reconstructing Volume Elements: Using the dump Command

The `dump` command dumps XVM configuration commands to a file.

You can use the `dump` command to dump configuration information for an individual volume element, or to dump the configuration information for all of the volume elements under the volume element you specify.

You can also use the `dump` command to dump configuration commands for a physical volume. You must explicitly dump the physical volume separately from a volume element tree.

The following example dumps a one-line creation command for the volume named `fred`:

```
xvm:cluster> dump vol/fred
```

The following example dumps information for all volume elements under each volume:

```
xvm:cluster> dump -topology vol/*
```

The following example dumps the contents of all XVM physical volumes and volume trees to the file `foo`:

```
xvm:cluster> dump -topology -file foo phys/* vol/*
```

Deleting Volume Elements: Using the delete Command

The `delete` command deletes a volume element. Parents of deleted volume elements remain and have open slots.

You cannot delete a volume element that is part of an open subvolume if doing so would cause the subvolume state to go offline. You can override this restriction with the `-force` option of the `delete` command.

If a volume element contains any attached children, it cannot be deleted. However, the `delete` command provides two options that override this restriction: the `-all` option, which deletes a volume element and all volume elements below it, and the `-nonslice` option, which deletes a volume element and all non-slice volume elements below it, detaching and keeping the slices. The `-all` and `-nonslice` options are mutually exclusive.

Removing Configuration Information: Using the `reprobe` Command

If a disk becomes inaccessible and needs to be replaced, you must tear down the existing configuration information for that disk. In this circumstance, you may not be able to execute standard XVM configuration commands on logical volumes that include that disk. To recover from this situation, you can use the `reprobe` command of the XVM volume manager to remove previous configuration information from the kernel.

Figure 4-1 XVM Logical Volumes for Root and Swap

The following example creates a system disk with 5 root partitions. In this example, the `-clrparts` option is used so that the command will work even if the target disk is a non-XVM system disk with a different partition layout.

```
xvm:local> label -clrparts -rootslices 5 -type root dks1d2
```

This command creates the slices and volumes shown in the following output to the `show -top` command:

```
xvm:local> show -top vol/dks*
vol/dks1d2_root0          0 online
  subvol/dks1d2_root0/data 3503056 online
    slice/dks1d2s0         3503056 online

vol/dks1d2_root2          0 online
  subvol/dks1d2_root2/data 3503056 online
    slice/dks1d2s1         3503056 online

vol/dks1d2_root3          0 online
  subvol/dks1d2_root3/data 3503056 online
    slice/dks1d2s2         3503056 online

vol/dks1d2_root4          0 online
  subvol/dks1d2_root4/data 3503056 online
    slice/dks1d2s3         3503056 online

vol/dks1d2_root5          0 online
  subvol/dks1d2_root5/data 3503056 online
    slice/dks1d2s4         3503056 online

vol/dks1d2_swap1          0 online
  subvol/dks1d2_swap1/data 262144 online
    slice/dks1d2s5         262144 online
```

After you execute this command, a physvol named `wilma` is created and two slices are defined on the physvol, as shown in Figure 4-2.

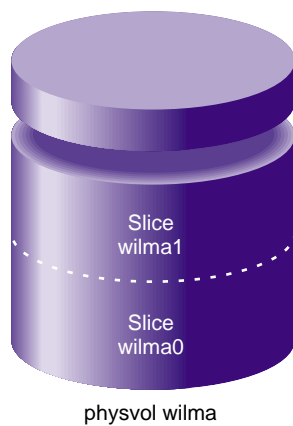


Figure 4-2 XVM Mirrored Root Physvol

The slices that are created on physvol `wilma` are inserted as mirrors into the existing root and swap logical volumes, as shown in Figure 4-3.

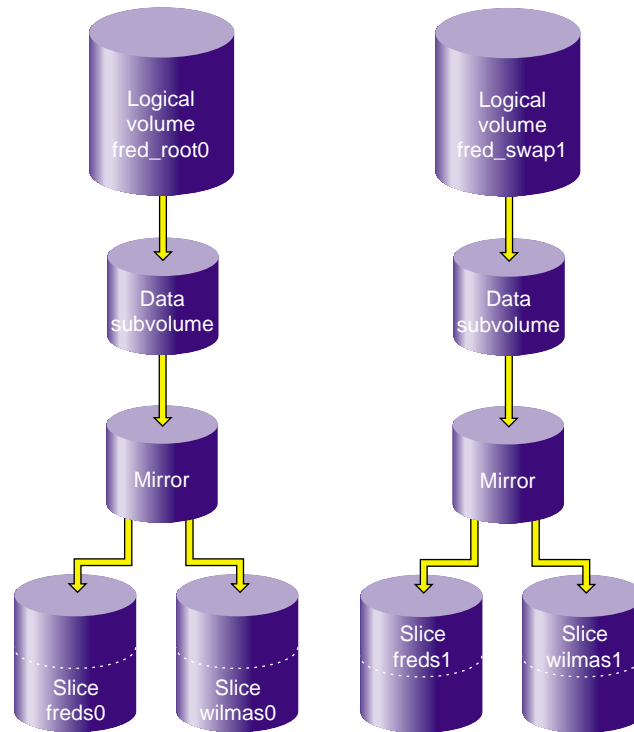


Figure 4-3 Root and Swap Mirrored Logical Volumes

The following series of commands changes physvol `xvmroot2` to an option disk, deletes the slices that make up the root and swap volumes (leaving the volumes as empty volumes), and unlabels the disk.

```
xvm:local> change option xvmroot2
xvm:local> delete xvmroot2/xvmroot2s0
xvm:local> delete xvmroot2/xvmroot2s1
xvm:local> unlabel xvmroot2
```

The following series of commands changes physvol `xvmroot` to an option disk, deletes the root and swap volumes and their child slices, and unlabels the disk.

```
xvm:local> change option xvmroot
xvm:local> delete -all vol/xvmroot_root
```


XVM Failover

Failover creates an infrastructure for the definition and management of multiple paths to a single disk device or LUN. XVM uses this infrastructure to select the path used for access to logical volumes created on the storage devices.

If your XVM configuration requires that you spread I/O across controllers, you must define a complete failover configuration file. This is necessary to ensure that I/O is restricted to the path that you select. For example, if you want a striped volume to span two host bus adapters, you must configure a failover configuration file to specify the preferred paths.

You should configure failover paths in order to get the maximum bandwidth and avoid LUN ownership movement (or changes) between RAID controllers; accessing the same LUN through different RAID controllers can degrade performance considerably. In general, you want to evenly distribute the I/O to LUNs across all available host bus adapters and RAID controllers and attempt to avoid blocking in the SAN fabric.

The ideal case, from performance standpoint, is to use as many paths as connection endpoints between two nodes in the fabric with as few blocking paths as possible in the intervening SAN fabric.

There are two failover mechanisms that XVM uses to select the preferred I/O path, each with its associated failover configuration file:

- Failover version 1 (V1), which uses the `failover.conf` configuration file
- Failover version 2 (V2), which uses the `failover2.conf` configuration file

These failover mechanisms are described in the following sections. Information on failover V2 can also be found in the `xvm` man page.

For information on using XVM failover with CXFS, see *CXFS 6 Administration Guide for SGI InfiniteStorage*.

Selecting a Failover Version

Whether you use failover V1 or failover V2 depends on a number of considerations, including the RAID mode you are running.

The TP9100 and RM610/660 RAID units do not have any host type failover configuration. Each LUN should be accessed via the same RAID controller for each node in the cluster because of performance reasons. These RAID configurations behave and have the same characteristics as the `SGIAVT` mode discussed below.

The TP9300, TP9500, and TP9700 RAID units will behave differently depending on the host type that is configured:

- `SGIRDAC` mode requires all I/O for a LUN to take place through the RAID controller that currently owns the LUN. Any I/O sent to a RAID controller that does not own the LUN will return an error to the host that sent the request. In order for the LUN to be accessed via the alternate controller in a RAID array, it requires the failover driver software on a host to send a command to the backup controller instructing it to take ownership of the specified LUN. At that point, the ownership of the LUN is transferred to the other LUN and I/O can take place via the new owner. Other hosts in the cluster will detect this change and update their I/O for the LUN to use a path to the RAID controller that now owns the LUN. Only XVM failover V1 can successfully control RAID configurations in `SGIRDAC` mode.
- `SGIAVT` mode also has the concept of LUN ownership by a single RAID controller. However, LUN ownership change will take place if any I/O for a given LUN is received by the RAID controller that is not the current owner. The change of ownership is automatic based on where I/O for a LUN is received and is not done by a specific request from a host failover driver. The concern with this mode of operation is that when a host in the cluster changes I/O to a different RAID controller than that used by the rest of the cluster, it can result in severe performance degradation for the LUN because of the overhead involved in constantly changing ownership of the LUN. Either XVM failover V1 or V2 can successfully control RAID configurations in `SGIAVT` mode (TP9400 does not accept `SGIAVT` mode).

If you are using XVM failover version 2, note the following:

- TP9100 1 GB and 2 GB:
 - `SGIRDAC` mode requires that the array is set to multiport
 - `SGIAVT` mode requires that the array is set to multitype

- TP9300/9500/S330 Fiber or SATA use of SGIAVT requires 06.12.18.xx code or later be installed
- TP9700 use of SGIAVT requires that 06.15.17xx. code or later be installed

SGIRDAC mode is supported under all revisions in RAID firmware section for these models.

Note that Failover V1 is not available on all operating systems:

- Linux supports Failover V2 only
- CXFS clients support Failover V2 only.

For information on choosing an appropriate failover version for a CXFS cluster, see *CXFS 6 Administration Guide for SGI InfiniteStorage*.

Failover V1

Failover V1 is the original failover mechanism. Failover V1 can be used with SGI TP9100, SGI TP9300, SGI TP9400, SGI TP9500, and SGI TP9700 RAID devices. It can also be used with third party storage by defining the path entries for the storage device in the `failover.conf` file.

When using failover V1, you can manually specify failover groups with the `failover.conf` file. See the `failover(7M)` man page for information on the `failover.conf` file, as well as additional information on failover V1.

XVM uses failover V1 when either of the following conditions is met:

- The SGI RAID device that contains the XVM physvol is not set to Automatic Volume Transfer (AVT) mode
- A `failover.conf` file has been defined (even if you have defined a `failover2.conf` file as well)
- The operating system supports Failover V1, as indicated in “Selecting a Failover Version” on page 92. If the operating system does not support Failover V1, it will always use Failover V2 whether there is a `failover.conf` file or not.

It is not necessary to define a `failover.conf` file in order to use failover V1.

Failover V1 is the version of failover supported by the XLV logical volume manager. If you are upgrading from XLV to XVM, you must replace the `failover.conf` file with a `failover2.conf` file if you choose to use failover V2, as described in “The `failover2.conf` File” on page 95.

Note: If a `failover.conf` file is missing or is not correctly defined on a host system, you may see “Illegal request” messages such as the following:

```
Mar 114:44:20 6A:h0uu19 unix: dksc 200200a0b80cd8da/lun1vol/c8p1: [Alert]
Illegal request: (asc=0x94, asq=0x1) CDB: 28 0 0 0 0 0 0 1 0
```

This message indicates that the host is trying to access a LUN by means of the alternate controller on a TP9500, TP9400, or TP9300 RAID. This message does not indicate a problem. You can eliminate this message by supplying a `failover.conf` file or by addressing existing errors in the `failover.conf` file.

This message can be generated by running the `XVM probe` command. The `XVM probe` command is run once automatically for every CXFS membership transition.

Failover V2

Failover V2 can be used with SGI TP9100, SGI TP9300, SGI TP9400, SGI TP9500, SGI TP9700 or 3rd-party RAID devices. When using failover V2, you can manually specify the attributes associated with a storage path by using the `failover2.conf` file, as described in “The `failover2.conf` File” on page 95.

XVM uses failover V2 when both of the following conditions are met:

- The SGI RAID device that contains the XVM physvol is set to Automatic Volume Transfer (AVT) mode
- There is no `failover.conf` file

It is not necessary to define a `failover2.conf` file in order to use failover V2.

The failover2.conf File

The configuration file for failover V2 is `/etc/failover2.conf`. The entries in this file define failover attributes associated with a path to the storage. Entries can be in any order.

In a `failover2.conf` file, you use the `preferred` keyword to specify the preferred path for accessing each XVM physvol; there is no default preferred path. The paths to a physvol are assigned an `affinity` value. This value is used to associate paths to a specific RAID controller, and to determine priority order in which groups of paths from a node to a LUN will be tried in the case of a failure: all affinity 0 paths are tried, then all affinity 1, then all affinity 2, etc.

Usually, all paths to the same controller are configured with the same affinity value and thus only two affinity values are used. You can, however, use more than two affinity values. What is important is that an affinity group for a LUN should not contain paths that go to different RAID groups.

The valid range of affinity values is 0 (lowest) through 15 (highest); the default is affinity 0. Paths with the same affinity number are all tried before failover V2 moves to the next highest affinity number; at 15, failover V2 wraps back to affinity 0 and starts over.

The paths to one controller of the RAID device should be affinity 0, which is the default affinity value. You should set the paths to the second controller to affinity 1.

Since the default affinity is zero, it would be sufficient to include entries only for those paths that are a non-zero affinity. It would also be sufficient to include an entry for the preferred path only. SGI recommends including definitions for all paths, however.

In a multi-host environment, it is recommended that the affinity values for a particular RAID controller be identical on every host in the CXFS cluster.

You can use the affinity value in association with the XVM `fswitch` command to switch an XVM physvol to a physical path of a defined affinity value, as described in “Switching physvol Path Interactively” on page 99.

If a `failover.conf` file is configured, XVM will employ failover V1 and use the configuration information in that file, even if you configure a `failover2.conf` file and the RAID devices are in AVT mode. You should remove or comment out an existing `failover.conf` file when you configure a `failover2.conf` file.

For instructions on generating a `failover2.conf` file, see “How to Create a `failover2.conf` File” on page 101.

Using the Appropriate Affinity Values in `/etc/failover2.conf` for XVM Device

The affinity setting in the `/etc/failover2.conf` file groups all of the device paths to a particular RAID controller that can be used in harmony without causing LUN ownership changes for a LUN between RAID controllers, which would result in poor disk performance.

You may find it useful to specify affinities starting at 1. This makes it easy to detect paths that have not yet been configured because they are assigned a default of `affinity=0`. For example, if you added a new HBA but forgot to add its paths to the `failover2.conf` file, all of its paths would have an `affinity=0`, which could result in LUN ownership changes if some paths point to controller A and others point to controller B. Using this convention would not avoid this problem, but would make it easier to notice. If you use this convention, you must do so for the entire cluster

Note: If you use the method where you do not use `affinity=0` and you do not define all of the paths in the `failover2.conf` file, you will have a affinity group using an unknown controller. If in the example where you are using `affinity=1` and `affinity=2`, if you are using `affinity=2` as your current path and there is a failover, you will failover to `affinity=0`, which could use the same RAID controller and thus fail again or might use the other RAID controller. If there are multiple unspecified paths in the `affinity=0` group, you might be mixing different RAID controllers in the same affinity group. This is only a performance issue, but you should fix any paths using the default `affinity=0` value by adding them to the `failover2.conf` file and using an appropriate affinity value.

Example `failover2.conf` Files

The following example for SGI groups the paths for `lun3` and the paths for `lun4`. The order of paths in the file is not significant.

Paths to the same LUN are detected automatically if the LUN has been labeled by `XVM.A` label command. It initiates a probe to discover new alternate paths. If storage that has

already been labeled is connected to a live system, you must run an XVM probe for XVM to recognize the disk as an XVM disk.

Without this file, all paths to each LUN would have affinity 0 and there would be no preferred path.

Setting a preferred path allows the administrator to guarantee that not all traffic to multiple LUNs goes through the same HBA, for example, by selecting preferred paths that spread the load out. Otherwise, the path used is the first one discovered and usually leads to almost all of the load going through the first HBA discovered that is attached to a specific RAID controller.

If no path is designated as `preferred`, the path used to the LUN is arbitrary based on the order of device discovery. There is no interaction between the preferred path and the affinity values.

This file uses affinity to group the RAID controllers for a particular path. Each controller has been assigned an affinity value. It shows the following:

- There is one PCI card with two ports off of the HBA (`pci04.01.1` and `pci04.01.0`)
- There are two RAID controllers, `node200800a0b813b982` and `node200900a0b813b982`
- Each RAID controller has two ports that are identified by `port1` or `port2`
- Each LUN has eight paths (via two PCI cards, two RAID controllers, and two ports on the controllers)
- There are two affinity groups for each LUN, `affinity=1` and `affinity=2`
- There is a preferred path for each LUN

```

/dev/xscsi/pci04.01.1/node200900a0b813b982/port1/lun3/disc,  affinity=1
/dev/xscsi/pci04.01.1/node200900a0b813b982/port2/lun3/disc,  affinity=1
/dev/xscsi/pci04.01.0/node200900a0b813b982/port1/lun3/disc,  affinity=1
/dev/xscsi/pci04.01.0/node200900a0b813b982/port2/lun3/disc,  affinity=1      preferred
/dev/xscsi/pci04.01.1/node200800a0b813b982/port1/lun3/disc,  affinity=2
/dev/xscsi/pci04.01.0/node200800a0b813b982/port1/lun3/disc,  affinity=2
/dev/xscsi/pci04.01.1/node200800a0b813b982/port2/lun3/disc,  affinity=2
/dev/xscsi/pci04.01.0/node200800a0b813b982/port2/lun3/disc,  affinity=2

/dev/xscsi/pci04.01.1/node200900a0b813b982/port1/lun4/disc,  affinity=1
/dev/xscsi/pci04.01.1/node200900a0b813b982/port2/lun4/disc,  affinity=1
/dev/xscsi/pci04.01.0/node200900a0b813b982/port1/lun4/disc,  affinity=1

```

```
/dev/xscsi/pci04.01.0/node200900a0b813b982/port2/lun4/disc, affinity=1
/dev/xscsi/pci04.01.1/node200800a0b813b982/port1/lun4/disc, affinity=2
/dev/xscsi/pci04.01.1/node200800a0b813b982/port2/lun4/disc, affinity=2 preferred
/dev/xscsi/pci04.01.0/node200800a0b813b982/port1/lun4/disc, affinity=2
/dev/xscsi/pci04.01.0/node200800a0b813b982/port2/lun4/disc, affinity=2
```

Given the above, failover will exhaust all paths to lun3 from RAID controller node200900a0b813b982 (with affinity=1 and the preferred path) before moving to RAID controller node200800a0b813b982 paths (with affinity=2)

The following example shows an additional grouping of PCI cards. The preferred path has an affinity of 2. If that path is not available, the failover mechanism will try the next path on the same PCI card (with affinity=2). If that is not successful, it will move to affinity=3, which is the other PCI port on the same RAID controller (node200800a0b813b982).

```
/dev/xscsi/pci04.01.1/node200900a0b813b982/port1/lun4/disc, affinity=1
/dev/xscsi/pci04.01.1/node200900a0b813b982/port2/lun4/disc, affinity=1
/dev/xscsi/pci04.01.0/node200900a0b813b982/port1/lun4/disc, affinity=1
/dev/xscsi/pci04.01.0/node200900a0b813b982/port2/lun4/disc, affinity=1
/dev/xscsi/pci04.01.1/node200800a0b813b982/port1/lun4/disc, affinity=3
/dev/xscsi/pci04.01.1/node200800a0b813b982/port2/lun4/disc, affinity=2 preferred
/dev/xscsi/pci04.01.0/node200800a0b813b982/port1/lun4/disc, affinity=3
/dev/xscsi/pci04.01.0/node200800a0b813b982/port2/lun4/disc, affinity=2
```

Parsing the failover2.conf File

The configuration information in the `failover2.conf` file becomes available to the system and takes effect when the system is rebooted. You can also parse the `failover2.conf` file on a running system by means of the XVM `foconfig` command:

```
xvm:cluster> foconfig -init
```

You can also execute the `foconfig` command directly from the shell prompt:

```
% xvm foconfig -init
```

The XVM `foconfig` command allows you to override the default `/etc/failover2.conf` filename with the `-f` option. The following command parses the failover information in the file `myfailover2.conf`:

```
xvm:cluster> foconfig -f myfailover2.conf
```

Additionally, the XVM `foconfig` command provides a `-verbose` option.

Running the `foconfig` command does not change any paths, even if new preferred paths are specified in the new failover file. To change the current path, use the `foswitch` command, as described in “Switching physvol Path Interactively” on page 99.

Switching physvol Path Interactively

When using failover V2, you can switch the path used to access an XVM physvol by using the XVM `foswitch` command. This enables you to set up a new current path on a running system, without rebooting.

Note: The XVM `foswitch` command does not switch paths for storage being managed by XVM V1. For XVM V1, use the `(x)scsifo` command.

Returning to a preferred path

The following command switches all XVM physvols back to their preferred path:

```
xvm:cluster> foswitch -preferred phys
```

You can also execute the `foswitch` command directly from the shell prompt:

```
% xvm foswitch -preferred phys
```

You may need to use the `-preferred` option of the `foswitch` command, for example, when a hardware problem may cause the system to switch the path to an XVM physvol. After addressing the problem, you can use this option to return to the preferred path.

Switching to a new device

The following command switches physvol `phys/lun22` to use device 345 (as indicated in the output to a `show -v` command):

```
xvm:cluster> foswitch -dev 345 phys/lun22
```

Setting a new affinity

The following command switches physvol `phys/lun33` to a path of affinity 2 if the current path does not already have that affinity. If the current path already has that affinity, no switch is made.

```
xvm:cluster> foswitch -setaffinity 2 phys/lun33
```

The following command switches physvol `phys/lun33` to the next available path of affinity 2, if there is one available.

```
xvm:cluster> foswitch -setaffinity 2 -movepath phys/lun33
```

The `-affinity` option of the `foswitch` command is being deprecated. Its functionality is the same as using `-setaffinity x -movepath`, as in the above example.

Switching paths for all nodes in a cluster

You can use the `-cluster` option of the `foswitch` command to perform the indicated operation on all nodes in a cluster.

The following command switches physvol `phys/lun33` to a path of affinity 2 for all nodes in the cluster if the current path does not already have that affinity. Where the current path already has that affinity, no switch is made.

```
xvm:cluster> foswitch -cluster -setaffinity 2 phys/lun33
```

The following command switches to the preferred path for `phys/lun33` for all nodes in the cluster:

```
xvm:cluster> foswitch -cluster -preferred phys/lun33
```

Automatic Probe after Labeling a Device

Under failover V2, after you label a device XVM must probe the device to locate the alternate paths to the device. Disks are probed when the system is booted and when you execute an XVM probe command.

When using failover V2, unlabeled disks are probed automatically when the XVM command exits after you label a device. This allows XVM failover to discover alternate paths for newly-labeled devices.

A probe can be slow, and it is necessary to probe a newly-labeled device only once. XVM allows you to disable the automatic probe feature of failover V2.

You can disable automatic probe in the following ways:

- Use the `-noprobe` option of the label command when you label the disk as a XVM physvol.
- Use the `set autoprobe` command to set autoprobe to disabled (or 0), as in the following example:

```
xvm:cluster> set autoprobe disabled
```

You can re-enable the automatic probe feature with the XVM `set autoprobe enabled` (or `set autoprobe 1`) command.

How to Create a failover2.conf File

This section provides a procedure for creating a `failover2.conf` file.

Create an initial `/etc/failover2.conf` file

You can easily create a `failover2.conf` file which can be edited to change affinity and preferred path settings with the following command.

```
xvm show -v phys | grep affinity > /etc/failover2.conf
```

Values in `< >` within the file are considered comments and can be deleted or ignored.

The entries in the file only apply to already labeled devices. You might want to run the command in both xvm domains, local and cluster, in order to get all defined devices.

Set affinity for each path in `/etc/failover2.conf`

To make it easier to understand and maintain the `/etc/failover2.conf` file, it is best to follow a consistent strategy for setting path affinity. Path failover will occur with preference toward another path of the same affinity as the current path regardless of the affinity value. Here is a simple strategy that works well for most sites:

- Set to `affinity=2` all paths to a physvol that go through controller A

- Set to `affinity=2` all paths to a physvol that go through controller B.

Note: For SGI Infinite Storage platforms, the WWN of a controller A path always starts with an even number in the first 4 digits (e.g. 2002, 2004, 2006) and the WWN of a controller B path always starts with an odd number in the first 4 digits (e.g. 2003, 2005, 2007).

Set the preferred path for each physvol

Make sure that for each LUN in a cluster the same controller is used. Otherwise, a LUN ownership change could result with each I/O process if the hosts are accessing the LUN at the same time.

When setting the preferred path you should have it match the preferred controller owner for the LUN. You can get this information from the TPSSM GUI or from the RAID Array profile.

Initialize the XVM configuration in the kernel

If you want to have XVM initialize its pathing configuration without going through a reboot, you can do that with the following command.

```
# xvm foconfig -init
```

This can be done on a live system without any ill effect because it will not initiate any path failovers. The current path will stay the current path even though the defined preferred path and path affinities may change.

Pay attention to any messages that are generated by this command as it will tell you if you goofed in defining your `/etc/failover2.conf` file.

Set all LUNs to their preferred path

You can set all physvols to their preferred path using the following command. In a cluster configuration be sure all of your `/etc/failover2.conf` files are correct and consistent and do this from every system to avoid trespass “storms”.

```
# xvm foswitch -preferred phys
```

Sample /etc/failover2.conf file

```
# failover v2 configuration file
#
# Note: All controller A paths are affinity=1
#       All controller B paths are affinity=2
# Make sure preferred path matches the preferred owner of the LUN
#

# RAID Array A
/dev/xscsi/pci02.01.1/node200500a0b813c606/port2/lun0/disc affinity=2
/dev/xscsi/pci02.01.1/node200400a0b813c606/port2/lun0/disc affinity=1
/dev/xscsi/pci02.01.0/node200500a0b813c606/port1/lun0/disc affinity=2
/dev/xscsi/pci02.01.0/node200400a0b813c606/port1/lun0/disc affinity=1 preferred

/dev/xscsi/pci02.01.1/node200500a0b813c606/port2/lun1/disc affinity=2
/dev/xscsi/pci02.01.1/node200400a0b813c606/port2/lun1/disc affinity=1
/dev/xscsi/pci02.01.0/node200500a0b813c606/port1/lun1/disc affinity=2 preferred
/dev/xscsi/pci02.01.0/node200400a0b813c606/port1/lun1/disc affinity=1

/dev/xscsi/pci02.01.1/node200500a0b813c606/port2/lun2/disc affinity=2
/dev/xscsi/pci02.01.1/node200400a0b813c606/port2/lun2/disc affinity=1 preferred
/dev/xscsi/pci02.01.0/node200500a0b813c606/port1/lun2/disc affinity=2
/dev/xscsi/pci02.01.0/node200400a0b813c606/port1/lun2/disc affinity=1

/dev/xscsi/pci02.01.1/node200500a0b813c606/port2/lun3/disc affinity=2 preferred
/dev/xscsi/pci02.01.1/node200400a0b813c606/port2/lun3/disc affinity=1
/dev/xscsi/pci02.01.0/node200500a0b813c606/port1/lun3/disc affinity=2
/dev/xscsi/pci02.01.0/node200400a0b813c606/port1/lun3/disc affinity=1
```


XVM Administration Procedures

This chapter contains examples of common XVM administration procedures. After an overview of some things to keep in mind before you begin, it provides the following procedures:

- “Preparing to Configure XVM Volumes under Linux” on page 107
- “Creating a Logical Volume with a Three-Way Stripe” on page 108
- “Striping a Portion of a Disk” on page 112
- “Creating a Logical Volume with a Data and Log Subvolume” on page 116
- “Creating a Logical Volume with a Data and Log Subvolume” on page 118
- “Creating a Volume from the Top Down” on page 121
- “Creating an XVM Logical Volume with Striped Mirrors” on page 123
- “Giving Away a System Disk from the Miniroot” on page 126
- “Online Reconfiguration Using Mirroring” on page 127
- “Online Modification of a Logical Volume” on page 133
- “Making an XVM Volume Using a GPT Label” on page 143
- “Converting an SGI DVH XVM Label to a GPT Label Suitable for XVM” on page 149
- “Tuning XVM” on page 155

Before You Begin

Before configuring an XVM logical volume, you may need to assess the status of your disks and your system:

- Before you can label a disk as an XVM disk, it must be formatted as a DVH/SGI disk or as a GPT disk:

- For information on formatting a disk for use in XVM under Linux, see “Preparing to Configure XVM Volumes under Linux” on page 107.
- For information on how the partitions must be configured on a GPT disk, see “XVM Partition Layout with GPT Disk Format” on page 8
- If you attempt to use the XVM Volume Manager to label a disk that is not a DVH/SGI disk or a GPT disk, you will get an error message indicating that the disk volume header partition is invalid.
- Before beginning any of the procedures in this chapter, you may find it useful to execute an `xvm show unlabeled/*` command to view the names of the disks on the system that have not been assigned to the XVM Volume Manager.
- You will not be able to label disks as XVM disks if they contain partitions currently in use as mounted filesystems. In a CXFS cluster, any XVM physical volumes that will be shared must be physically connected to all nodes in the cluster.
- In general, you will find it useful to use the options of the `show` command to view your system configuration and status. For example, the `show -v stripe0` command displays the stripe unit (in this case for `stripe0`).
- To configure XVM logical volumes, you need to be logged in as root. However, you can display logical volume configuration information even if you do not have root privileges.

Note: As you configure an XVM logical volume, keep in mind that at any time you can view extended help information for an XVM command by entering the `help` command with the `-v[erbose]` option. For example, you can view the full help screen that includes the options for the `slice` command by entering the following:

```
xvm:cluster> help -v slice
```

Preparing to Configure XVM Volumes under Linux

These instructions only apply to standalone servers running SGI Performance Suite. For information on CXFS clients, see *CXFS 6 Client-Only Guide for SGI InfiniteStorage*. Before configuring logical volumes under Linux, you may need to run through the following checklist:

1. Under SGI Performance Suite, XVM is not installed by default. You should ensure that the `xvm-cmds` rpm is installed.
2. To use XVM under SGI Performance Suite, you must obtain and install the appropriate XVM license. XVM licenses are usually nodelocked and reside on the local machine in `/etc/lk/keys.dat`. You must have the `XVM_STD_IPF` feature. If you have the `XVM_PLEX_IPF` license only, XVM will report a license error. `XVM_PLEX_IPF` is required for the local mirroring feature. `XVM_PLEX_CLUSTER` is required for cluster mirror volumes.
3. Confirm that the `xvm-standalone` kernel module is loaded. You can use the `lsmod` command, as in the following example:

```
[root]# lsmod | grep xvm
xvm-standalone      717208    0
```
4. You can determine what disks are potentially available on the system by executing `cat /proc/xscsi/dksc`. You will not be able to configure the system disk as an XVM disk.
5. You may need to determine whether any of the disks on the system are dual-ported, which means that each disk is connected to two controllers. If a disk is dual-ported, what appear to be two different disk paths could reference the same disk. You need to be careful that you don't mount the same disk device as a filesystem outside of XVM that you are also using as part of an XVM volume.

Use the `ls` command to get a list of the disks, as in the following example:

```
ls /dev/xscsi/pci*/target*/lun*/disc
```

6. You must format each disk you will use for the XVM volume as a DVH/SGI disk or as a GPT disk. To format the disk as a DVH disk, use the `parted` command as in the following example.

```
parted /dev/xscsi/pci05.01.0/target98/lun0/disc mklabel dvh
```

For information on how the partitions must be configured on a GPT disk, see “XVM Partition Layout with GPT Disk Format” on page 8

7. If you are going to set up Command Tagged Queuing (CTQ) or if you are going to enable write-caching, you should do so at this point.

After you are familiar with the general requirements for creating a simple XVM volume on Linux, you should be able to use the examples in this chapter to determine how to configure more complex volumes.

Creating a Logical Volume with a Three-Way Stripe

The following example shows the procedure for creating a simple logical volume that stripes data across three disks. In this example, the entire usable space of each disk is used for the slice.

Figure 6-1 shows the logical volume this example creates.

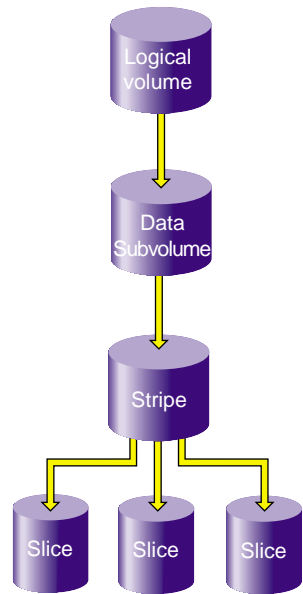


Figure 6-1 XVM Logical Volume with Three-Way Stripe

1. Assign disks to XVM to manage. This example assigns three disks to the XVM Volume Manager. You need to perform this procedure only once for each disk that you will be using to create XVM logical volumes.

```
# xvm
xvm:cluster> label -name disk0 dks2d70
disk0
xvm:cluster> label -name disk1 dks2d71
disk1
xvm:cluster> label -name disk2 dks2d72
disk2
```

2. You may want to view all the disks that have been assigned to the XVM volume manager as XVM physical volumes to verify what you have labeled:

```
xvm:cluster> show phys/*
phys/disk0          35542780 online
phys/disk1          35542780 online
phys/disk2          35542780 online
```

3. Create a slice that consists of all of the usable blocks of each of the XVM physical volumes:

```
xvm:cluster> slice -all disk*
</dev/cxvm/disk0s0> slice/disk0s0
</dev/cxvm/disk1s0> slice/disk1s0
</dev/cxvm/disk2s0> slice/disk2s0
```

4. Create a stripe that consists of the three slices you have defined. In this example, the generated volume will be named `stripedvol` explicitly. A data subvolume will automatically be generated as well.

The following command names the generated volume `stripedvol`:

```
xvm:cluster> stripe -volname stripedvol slice/disk0s0 slice/disk1s0 slice/disk2s0
</dev/cxvm/stripedvol> stripe/stripedvol
```

In this example:

- `/dev/rcxvm/stripedvol` is the name of the volume on which you can execute the `mkfs` command
- `stripe/stripedvol` is the name of the stripe object

In this example, the name of the stripe object is subject to change on subsequent boots but the name of the volume is not.

5. View the topology of the logical volume you have created:

```
xvm:cluster> show -top stripedvol
vol/stripedvol                0 online
  subvol/stripedvol/data      106627968 online
    stripe/stripe0           106627968 online,tempname
      slice/disk0s0           35542780 online
      slice/disk1s0           35542780 online
      slice/disk2s0           35542780 online
```

6. Exit the xvm tool by typing `exit` (or `quit` or `bye`). You can then execute the `mkfs` command on the volume.

```
xvm:cluster> exit
# mkfs /dev/cxvm/stripedvol
meta-data=/dev/cxvm/stripedvol  isize=256    agcount=51, agsize=261344 blks
data      =                    bsize=4096  blocks=13328496, imaxpct=25
          =                    sunit=16    swidth=48 blks, unwritten=1
naming    =version 1          bsize=4096
log       =internal log      bsize=4096  blocks=1168
realtime  =none              extsz=65536 blocks=0, rtextents=0
```

7. You can now mount the filesystem. For a shared filesystem in a CXFS cluster, you mount the filesystem with the CXFS GUI or the `cmgr(1M)` command, as described in *CXFS 6 Administration Guide for SGI InfiniteStorage*.

For a local filesystem that is not part of a cluster, you can put a logical volume in the `fstab` file and use the `mount` command to mount the filesystem you created.

Striping a Portion of a Disk

The following example shows the procedure for creating a stripe on the outer third of a disk. It also includes some advice on naming volume elements.

Figure 6-2 shows the logical volume this example creates.

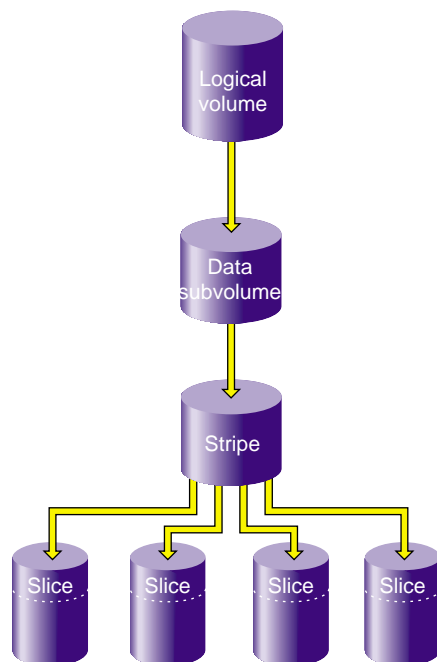


Figure 6-2 Striping a Portion of a Disk

1. Assign disks to XVM to manage. This example assigns four disks to XVM. Note that four separate controllers are chosen for better stripe performance.

```
xvm:cluster> label -name lucy dks21d0
lucy
xvm:cluster> label -name ricky dks22d0
ricky
xvm:cluster> label -name ethyl dks23d0
ethyl
xvm:cluster> label -name fred dks24d0
fred
```

2. In this example, you use one-third of each disk for the stripe.

There are two ways to partition a disk into thirds. You can allocate the entire disk, but only use the last third. For example, for disk `lucy` you could do the following (and use `slice/lucys2` for the stripe):

```
xvm:cluster> slice -equal 3 lucy
</dev/cxvm/lucys0> slice/lucys0
</dev/cxvm/lucys1> slice/lucys1
</dev/cxvm/lucys2> slice/lucys2
```

Alternately, you can confine the block range explicitly to one-third of the disk. For example, you can do the following to allocate the last third of the other disks (`ricky`, `ethyl`, and `fred`):

```
xvm:cluster> slice -start 11852676 -length 5926340 ricky
</dev/cxvm/rickys0> slice/rickys0
xvm:cluster> slice -start 11852676 -length 5926340 ethyl
</dev/cxvm/ethyls0> slice/ethyls0
xvm:cluster> slice -start 11852676 -length 5926340 fred
</dev/cxvm/freds0> slice/freds0
```

3. Verify the allocation.

The following example shows the allocation on `lucy`, the disk divided into three equal stripes:

```
xvm:cluster> show -v lucy
XVM physvol phys/lucy
=====
...
-----
0 5926338 slice/lucys0
5926338 5926338 slice/lucys1
11852676 5926340 slice/lucys2
Local stats for phys/lucy since being enabled or reset:
```

```
-----
stats collection is not enabled for this physvol
```

The following example verifies the allocation on `ricky`, one of the disks that was allocated explicitly:

```
xvm:cluster> show -v ricky
XVM physvol phys/ricky
=====
...
-----
0 11852676 (unused)
11852676 5926340 slice/rickys0
-----
```

4. Create the stripe. In this example, the generated volume is explicitly named `I_Love_Lucy`.

```
xvm:cluster> stripe -volname I_Love_Lucy -unit 128 slice/lucys2 \
slice/rickys0 slice/ethyls0 slice/freds0
</dev/cxvm/I_Love_Lucy> stripe/stripe0
```

5. Sometimes it may be useful to categorize portions of a complex volume by name. For example, you may want to name a portion of a volume `faststripe` so that a search can be done for volumes that have fast stripe objects. The following command names a stripe as well as the volume:

```
xvm:cluster> stripe -volname I_Love_Lucy -vename faststripe0 \
-unit 128 slice/lucys2 slice/rickys0 slice/ethyls0 slice/freds0
</dev/cxvm/I_Love_Lucy> stripe/faststripe0
```

When you name the stripe as in the preceding example, you can use wildcards to show all fast stripes:

```
xvm:cluster> show -top stripe/fast*
stripe/faststripe0      23705088 online
  slice/lucys2          5926340 online
  slice/rickys0         5926340 online
  slice/ethyls0         5926340 online
  slice/freds0          5926340 online
```

You can also use wildcards to show all objects starting with 'I', as in the following example:

```
xvm:cluster> show I*
vol/I_Love_Lucy 0 online
```

6. Exit the xvm tool by typing `exit` (or `quit` or `bye`). You can now execute the `mkfs` command on the volume.

```
xvm:cluster> exit
hugh3 2# mkfs /dev/cxvm/I_Love_Lucy
meta-data=/dev/rxvm/I_Love_Lucy  isize=256    agcount=26,
agsize=256416 blks
data      =                               bsize=4096  blocks=6666528,
imaxpct=25
          =                               sunit=16   swidth=48 blks,
unwritten=1
naming    =version 1                      bsize=4096
log       =internal log                   bsize=4096  blocks=1168
realtime  =none                            extsz=65536 blocks=0, rtextents=0
```

7. Mount the filesystem. For a shared filesystem in a CXFS cluster, you mount the filesystem with the CXFS GUI or the `cmgr(1M)` command, as described in *CXFS 6 Administration Guide for SGI InfiniteStorage*.

If your XVM volume is a local volume, you can put a logical volume in the `fstab` file and use the `mount` command to mount the filesystem you created.

Creating a Logical Volume with a Data and Log Subvolume

The following example creates an XVM logical volume that includes both a data subvolume and a log subvolume. In this example, the data subvolume consists of all the usable space of two disks, and the log subvolume consists of all the usable space of a third disk.

Figure 6-3 shows the logical volume this example creates.

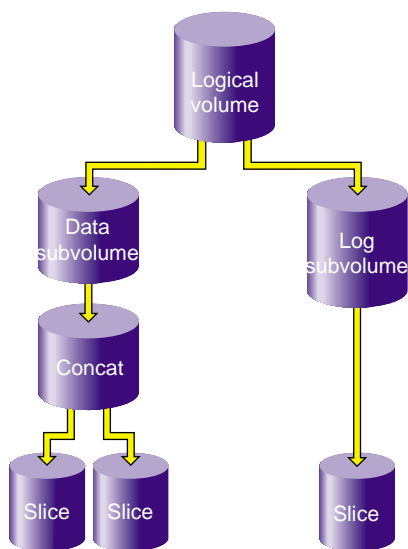


Figure 6-3 XVM Logical Volume with a Log Subvolume

1. Assign three disks to XVM to manage.

```
# xvm
xvm:cluster> label -name disk0 dks0d2
disk0
xvm:cluster> label -name disk1 dks0d3
disk1
xvm:cluster> label -name disk2 dks5d42
disk2
```


2. Create a slice that consists of all of the usable blocks of each of the XVM physical volumes you have created:

```
xvm:cluster> slice -all disk*
</dev/xvm/disk0s0> slice/disk0s0
</dev/xvm/disk1s0> slice/disk1s0
</dev/xvm/disk2s0> slice/disk2s0
```

3. Combine two of the slices into a concat. In this example, the generated volume is named `concatvol`.

```
xvm:cluster> concat -volname concatvol slice/disk0s0 slice/disk1s0
</dev/cxvm/concatvol> concat/concat3
```

You can view the configuration of the volume you have defined that does not yet contain a log subvolume:

```
xvm:cluster> show -top vol/concatvol
vol/concatvol          0 online
  subvol/concatvol/data 35554848 online
    concat/concat3      35554848 online,tempname
      slice/disk0s0      17777424 online
      slice/disk1s0      17777424 online
```

4. Create the log subvolume consisting of the third slice you created. Use the `-tempname` option to indicate that the system will generate a temporary name for the volume. You will not need to name this volume, as you will be attaching the log subvolume to the existing `concatvol` volume.

```
xvm:cluster> subvol -tempname -type log slice/disk2s0
</dev/cxvm/vol7_log> subvol/vol7/log
```

5. Attach the log subvolume to the existing `concatvol` volume.

```
xvm:cluster> attach subvol/vol7/log vol/concatvol
vol/concatvol
```

6. Display the logical volume:

```
xvm:cluster> show -top vol/concatvol
vol/concatvol          0 online
  subvol/concatvol/data 35554848 online
    concat/concat3      35554848 online,tempname
      slice/disk0s0      17777424 online
      slice/disk1s0      17777424 online
  subvol/concatvol/log  17779016 online
    slice/disk2s0        17779016 online
```

Creating a Logical Volume with a Data and Log Subvolume

The following example creates an XVM logical volume that includes a data subvolume and a log subvolume. Two similar ways of performing this procedure are shown.

Figure 6-4 shows the logical volume this example creates.

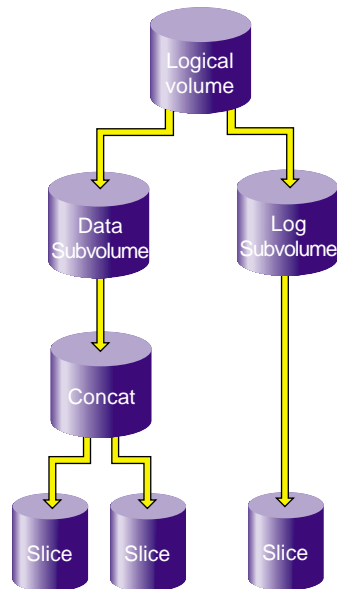


Figure 6-4 Logical Volume with Data and Log Subvolumes

This example assumes that you have already assigned disks to XVM to manage and that you have previously created the five slices you will use to build the logical volume:

- slice/disk1s0
- slice/disk2s0
- slice/disk3s0
- slice/disk4s0
- slice/disk5s0.

1. Create the concat that will comprise the data subvolume:

```
xvm:cluster> concat -tempname slice/disk1s0 slice/disk2s0
```

```
</dev/cxvm/vol0> concat/concat0
```

2. Create the data subvolume:

```
xvm:cluster> subvolume -tempname -type data concat/concat0  
</dev/cxvm/vol2> subvol/vol2/data
```

3. Create the log subvolume:

```
xvm:cluster> subvolume -tempname -type log slice/disk5s0  
</dev/cxvm/vol4_log> subvol/vol4/log
```

4. Create the logical volume that contains the two subvolumes:

```
xvm:cluster> volume -volname myvol subvol/vol2/data \  
subvol/vol4/log  
vol/myvol
```

5. Display the logical volume:

```
xvm:cluster> show -top myvol  
vol/myvol  
    subvol/myvol/data          0 online  
        concat/concat0      35558032 online  
            slice/disk1s0    17779016 online  
            slice/disk2s0    17779016 online  
    subvol/myvol/log          8192 online  
        slice/disk5s0        8192 online  
            stripe/stripel   35557888 online,tempname  
            slice/disk3s0    17779016 online  
            slice/disk4s0    17779016 online
```

The following sequence of commands generates the same volume, but with one less step since the volume name is established with the `concat` command. The log subvolume is subsequently attached.

```
xvm:cluster> concat -volname myvol slice/disk1s0 slice/disk2s0
</dev/cxvm/myvol> concat/concat1
xvm:cluster> stripe -tempname slice/disk3s0 slice/disk4s0
</dev/cxvm/vol6> stripe/stripe2
xvm:cluster> subvolume -tempname -type log slice/disk5s0
</dev/cxvm/vol8_log> subvol/vol8/log
xvm:cluster> show -top myvol
vol/myvol                                0 online
  subvol/myvol/data                       35558032 online
    concat/concat1                        35558032 online,tempname
      slice/disk1s0                        17779016 online
      slice/disk2s0                        17779016 online
  subvol/myvol/log                         8192 online
    slice/disk5s0                          8192 online
      stripe/stripe2                       35557888 online,tempname
      slice/disk3s0                        17779016 online
      slice/disk4s0                        17779016 online
```

Creating a Volume from the Top Down

When you configure an XVM logical volume, you can create the volume's hierarchy from the bottom up or from the top down. The example in this section creates the same XVM logical volume as in the example in "Creating a Logical Volume with a Data and Log Subvolume" on page 116 and shown in Figure 6-3, but it creates an empty volume first before attaching the child volume elements for that volume.

1. Assign three disks to XVM to manage:

```
# xvm
xvm:cluster> label -name disk0 dks0d2
disk0
xvm:cluster> label -name disk1 dks0d3
disk1
xvm:cluster> label -name disk2 dks5d42
disk2
```

2. Create a slice that consists of all of the usable blocks of each of the XVM physical volumes you have created:

```
xvm:cluster> slice -all disk*
</dev/cxvm/disk0s0> slice/disk0s0
</dev/cxvm/disk1s0> slice/disk1s0
</dev/cxvm/disk2s0> slice/disk2s0
```

3. Create an empty volume named topdownvol:

```
xvm:cluster> volume -volname topdownvol
vol/topdownvol
```

4. Display the volume:

```
xvm:cluster> show -top vol/top*
vol/topdownvol                                0 offline
    (empty)                                    * *
```

5. Create an empty concat volume element and display the result:

```
xvm:cluster> concat -tempname
</dev/cxvm/vol8> concat/concat5
xvm:cluster> show -top vol/vol8
vol/vol8                                        0 offline,tempname
    subvol/vol8/data                            0 offline,pieceoffline
        concat/concat5                          0 offline,tempname
            (empty)                              * *
```

6. Attach the generated data subvolume that contains the concat to topdownvol and display the result:

```
xvm:cluster> attach subvol/vol8/data vol/topdownvol
vol/topdownvol
xvm:cluster> show -top vol/topdownvol
vol/topdownvol                0 offline
  subvol/topdownvol/data        0 offline,pieceoffline
  concat/concat5                0 offline,tempname
  (empty)                       * *
```

7. Attach two slices to fill the empty concat and display the result:

```
xvm:cluster> attach slice/disk0s0 slice/disk1s0 concat/concat5
</dev/cxvm/topdownvol> concat/concat5
xvm:cluster> show -top vol/topdownvol
vol/topdownvol                0 online
  subvol/topdownvol/data        35554848 online
  concat/concat5                35554848 online,tempname
  slice/disk0s0                 17777424 online
  slice/disk1s0                 17777424 online
```

8. Create a log subvolume:

```
xvm:cluster> subvol -tempname -type log
</dev/cxvm/vol9_log> subvol/vol9/log
```

9. Attach the log subvolume to topdownvol and display the result:

```
xvm:cluster> attach subvol/vol9/log vol/topdownvol
vol/topdownvol
xvm:cluster> show -top vol/topdownvol
vol/topdownvol                0 offline
  subvol/topdownvol/data        35554848 online
  concat/concat5                35554848 online,tempname
  slice/disk0s0                 17777424 online
  slice/disk1s0                 17777424 online
  subvol/topdownvol/log         0 offline
  (empty)                       * *
```

10. Attach the third slice to the log subvolume and display the results:

```
xvm:cluster> attach slice/disk2s0 subvol/topdownvol/log  
</dev/cxvm/topdownvol_log> subvol/topdownvol/log  
xvm:cluster> show -top vol/topdownvol  
vol/topdownvol                0 online  
  subvol/topdownvol/data       35554848 online  
    concat/concat5             35554848 online,tempname  
      slice/disk0s0             17777424 online  
      slice/disk1s0             17777424 online  
  subvol/topdownvol/log       17779016 online  
    slice/disk2s0              17779016 online
```

Creating an XVM Logical Volume with Striped Mirrors

The following example creates a logical volume with striped mirrors. In this example the logical volume contains a stripe that consists of two mirrors, each mirroring a slice that Figure 6-5 shows the logical volume this example creates.

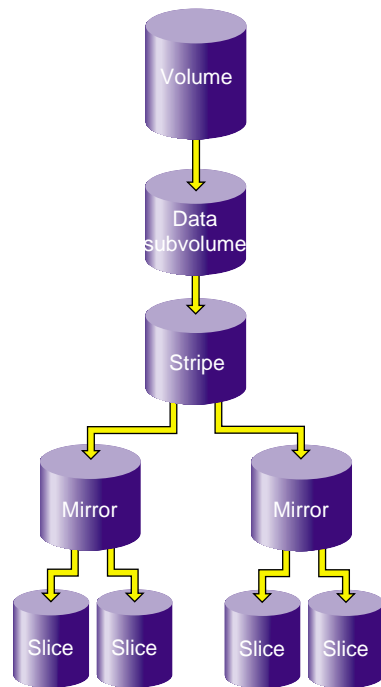


Figure 6-5 XVM Logical Volume with Striped Mirrors

1. Assign four disks to XVM to manage:

```
xvm:cluster> label -name disk0 dks2d70  
disk0  
xvm:cluster> label -name disk1 dks2d71  
disk1  
xvm:cluster> label -name disk2 dks2d72  
disk2  
xvm:cluster> label -name disk3 dks2d73  
disk3
```


2. Create a slice out of all of the usable blocks on each XVM physical volume:

```
xvm:cluster> slice -all disk*
</dev/cxvm/disk0s0> slice/disk0s0
</dev/cxvm/disk1s0> slice/disk1s0
</dev/cxvm/disk2s0> slice/disk2s0
</dev/cxvm/disk3s0> slice/disk3s0
```

3. Create two mirrors, each consisting of two of the slices you have defined. Since you are creating new mirrors that will be written to before they are read, you can specify the `-clean` option. This indicates that the mirrors do not need to be synchronized on creation.

If you do not specify the `-clean` option, executing this command initiates a mirror revive, which synchronizes the data on the slices. A message indicating that a revive has begun would be written to the `SYSLOG`, and another message would be written to the `SYSLOG` when the revive completes.

You will not need to define a persistent name for the volume that will be generated.

```
xvm:cluster> mirror -tempname -clean slice/disk0s0 slice/disk1s0
</dev/cxvm/vol2> mirror/mirror1
xvm:cluster> mirror -tempname -clean slice/disk2s0 slice/disk3s0
</dev/cxvm/vol3> mirror/mirror2
```

4. Create a stripe that consists of the two mirrors you have defined, naming the volume that will be generated to contain the stripe. This command attaches the mirrors to the stripe.

```
xvm:cluster> stripe -volname mirvol mirror/mirror1 mirror/mirror2
</dev/cxvm/mirvol> stripe/stripe2
```

5. Display the XVM logical volume:

```
xvm:cluster> show -top mirvol
vol/mirvol          0 online
  subvol/mirvol/data 71085312 online
    stripe/stripe2   71085312 online,tempname
      mirror/mirror1 35542780 online,tempname
        slice/disk0s0 35542780 online
        slice/disk1s0 35542780 online
      mirror/mirror2 35542780 online,tempname
        slice/disk2s0 35542780 online
        slice/disk3s0 35542780 online
```

Giving Away a System Disk from the Miniroot

When running from an XVM root partition, changing the system name in `/etc/sysid` creates a conflict between the host name in the XVM volume header and the name of the system. This will result in a system panic at the next system reboot unless you first boot the miniroot and run the XVM `give` command to assign the system disk physical volume to the new system name,

To give away a physical volume from the miniroot, use the following procedure.

1. Reset the machine to bring up the **System Maintenance** menu.
2. Select **Install System Software**.
3. Choose remote or local install and enter the path and IP address of the system software, as appropriate for your system.
4. Start the system. This loads the `Inst` utility into the `swap` partition.
5. At the `Inst` prompt, enter `sh` to exit to a shell.
6. From the shell prompt in the miniroot, you can run the XVM `give` command:
 - Enter `xvm`
 - Use the `give` command to give the old physical volume name to the new `sysid` name:

```
xvm> give newhostname physvol
```
7. Exit from XVM.
8. Exit from the shell to return to the `Inst` menu.
9. Quit the `Inst` utility.

You should now be able to reboot the system.

Online Reconfiguration Using Mirroring

The following procedure reconfigures a filesystem while the filesystem is online by mirroring the data in a new configuration, then detaching the original configuration. It is not necessary to unmount the filesystem to perform this procedure.

Note: Figure 6-6 shows the configuration of the original filesystem that has been built and mounted.

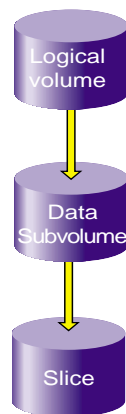


Figure 6-6 Original Online Filesystem

In the example, the original filesystem is a filesystem that consists of a single slice. It is named `myfs`, and is configured as follows:

```
xvm:cluster> show -top myfs
vol/myfs                                0 online
  subvol/myfs/data                       102400 online,open
    slice/disk5s0                         102400 online,open
```

This procedure reconfigures this filesystem into one that consists of a four-way stripe.

1. Create the slices that will make up the four-way stripe. The stripe that you are creating should be the same size as the existing filesystem, so in this example each slice is one-quarter the size of the filesystem.

```
xvm:cluster> slice -length 25600 phys/disk[1234]
</dev/cxvm/disk1s0> slice/disk1s0
</dev/cxvm/disk2s0> slice/disk2s0
</dev/cxvm/disk3s0> slice/disk3s0
</dev/cxvm/disk4s0> slice/disk4s0
```

Create the four-way stripe. This example does not specify a stripe unit, which accepts the default stripe unit of 128 blocks. In this case, using the default stripe unit uses all the blocks of each slice, since the slices are multiples of the stripe unit in size.

```
xvm:cluster> stripe -tempname slice/disk[1234]s0
</dev/cxvm/vol5> stripe/stripes5
```

Display the stripe configuration:

```
xvm:cluster> show -top stripes5
stripes5/stripes5          102400 online,tempname
  slice/disk1s0             25600 online
  slice/disk2s0             25600 online
  slice/disk3s0             25600 online
  slice/disk4s0             25600 online
```

2. Insert a temporary mirror above the point that will be reconfigured. In this example, that point is slice/disk5s0.

```
xvm:cluster> insert mirror slice/disk5s0
</dev/cxvm/myfs> mirror/mirror5
```

Display the logical volume:

```
xvm:cluster> show -top myfs
vol/myfs                   0 online
  subvol/myfs/data         102400 online,open
    mirror/mirror5         102400 online,tempname,open
      slice/disk5s0        102400 online,open
```

Figure 6-7 shows the configuration of the filesystem `myfs` after the insertion of the mirror.

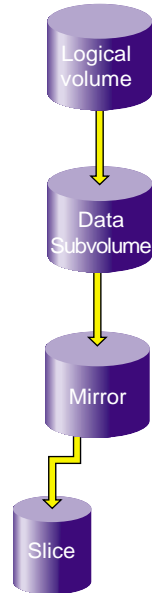


Figure 6-7 Filesystem after Insertion of Mirror

3. Attach the stripe to the mirror, which is `mirror5` in this example. This will initiate a revive, which replicates the data of `slice/disk5s0` on `stripe5`.

```
xvm:cluster> attach stripe/stripe5 mirror/mirror5
</dev/cxvm/myfs> mirror/mirror5
```

Display the logical volume:

```
xvm:cluster> show -top myfs
vol/myfs                                0 online
  subvol/myfs/data                       102400 online,open
    mirror/mirror5                       102400 online,tempname,reviving:26%
      slice/disk5s0                       102400 online,open
        stripe/stripe5                    102400 online,tempname,reviving
          slice/disk1s0                    25600 online,open
          slice/disk2s0                    25600 online,open
          slice/disk3s0                    25600 online,open
          slice/disk4s0                    25600 online,open
```

Figure 6-8 shows the configuration of the filesystem `myfs` after the stripe has been attached to the mirror.

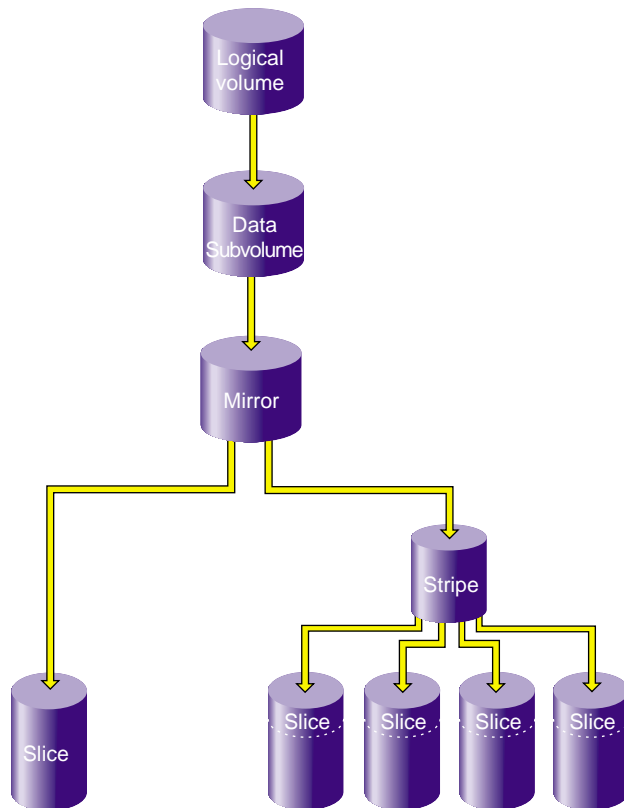


Figure 6-8 Filesystem after Attaching Stripe to Mirror

4. Detach `slice/disk5s0` from the mirror. You must wait for the mirror revive to complete before you can do this, since you cannot detach the last valid piece of an open mirror (unless you use the `-force` option), and until the revive completes the slice is the only valid leg of the mirror.

```
xvm:cluster> detach slice/disk5s0  
</dev/cxvm/disk5s0> slice/disk5s0
```

Figure 6-9 shows the configuration of the filesystem `myfs` after the original slice has been detached.

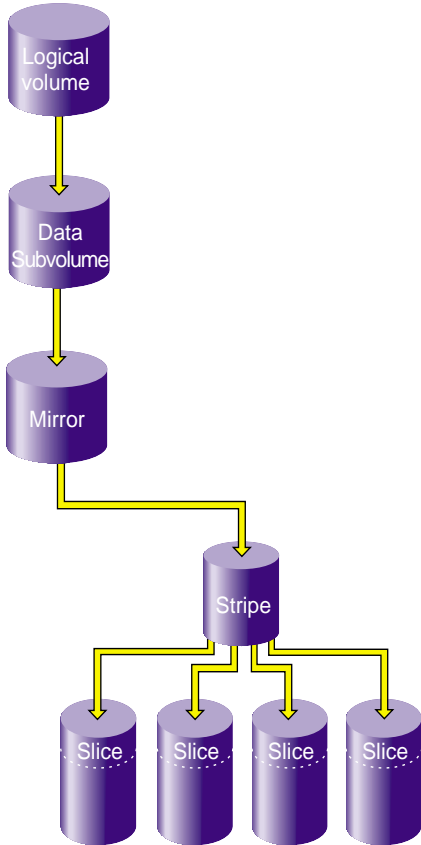


Figure 6-9 Filesystem after Detaching Original Slice

5. Remove the mirror layer from the tree by collapsing around the mirror:

```
xvm:cluster> collapse mirror/mirror5
```

The filesystem is now configured as a four-way stripe. Display the logical volume:

```
xvm:cluster> show -top myfs
vol/myfs                                0 online
  subvol/myfs/data                       102400 online,open
    stripe/stripe5                       102400 online,tempname,open
      slice/disk1s0                       25600 online,open
      slice/disk2s0                       25600 online,open
      slice/disk3s0                       25600 online,open
      slice/disk4s0                       25600 online,open
```

Figure 6-10 shows the final configuration of the filesystem `myfs`.

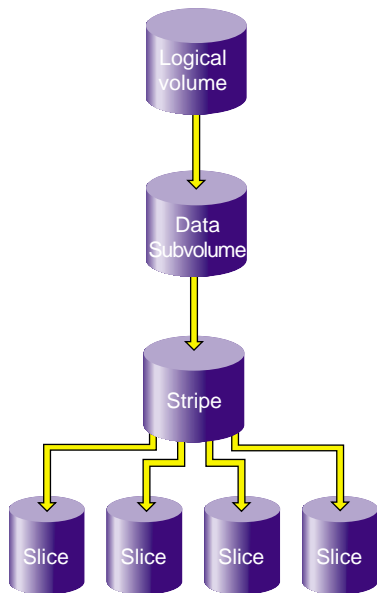


Figure 6-10 Reconfigured Filesystem

Online Modification of a Logical Volume

The following sections describe the procedure for creating and modifying a logical volume. In this procedure, the modifications to the logical volume are made after you have made a filesystem on the logical volume and mounted the filesystem.

This procedure is divided into the following steps:

- Creating the logical volume
- Growing the logical volume
- Mirroring data on the logical volume
- Converting a concat to striped data using mirroring
- Removing a mirror
- Mirroring individual stripe members

These steps are described in the following sections.

Creating the Logical Volume

The following procedure creates a simple logical volume that contains a single slice. Figure 6-11 shows the original XVM logical volume this procedure creates.

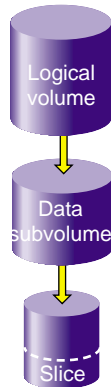


Figure 6-11 Original XVM Logical Volume

1. Create a slice on the physvol `pebble`, naming the generated volume that contains the slice `tinyvol`:

```
xvm:cluster> slice -volname tinyvol -start 17601210 -length 177792 \
pebble </dev/cxvm/tinyvol> slice/pebbles0
```

2. Exit the XVM CLI by typing `exit` and then create a filesystem:

```
xvm:cluster> exit
# mkfs /dev/cxvm/tinyvol
meta-data=/dev/cxvm/tinyvol  isize=256  agcount=5, agsize=4445 blks
data      =                  bsize=4096  blocks=22224, imaxpct=25
          =                  sunit=0     swidth=0 blks, unwritten=1
naming    =version 1         bsize=4096
log       =internal log     bsize=4096  blocks=1168
realtime  =none             extsz=65536 blocks=0, rtextents=0
```

3. Mount the filesystem. For a shared filesystem in a CXFS cluster, you mount a filesystem with the CXFS GUI or the `cmgr(1M)` command, as described in *CXFS 6 Administration Guide for SGI InfiniteStorage*.

For a local filesystem, you can put a logical volume in the `fstab` file and use the `mount` command to mount the filesystem you created.

Growing the Logical Volume

The following procedure grows the logical volume you have created. Figure 6-12 shows the logical volume after the insertion of a concat to grow the logical volume.

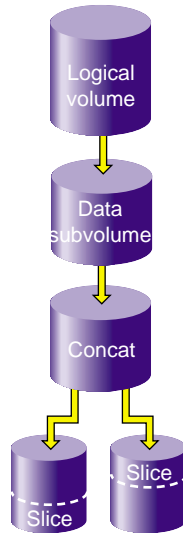


Figure 6-12 XVM Logical Volume after Insert

1. Display the logical volume `tinyvol`, showing the topology of the volume:

```
xvm:cluster> show -top tinyvol
vol/tinyvol                                0 online
  subvol/tinyvol/data                       177792 online,open
    slice/pebbles0                          177792 online,open
```

2. Change the volume `tinyvol` to include a concat container:

```
xvm:cluster> insert concat slice/pebbles0
</dev/cxvm/tinyvol> concat/concat3
```

3. Display the results of the insert command:

```
xvm:cluster> show -top tinyvol
vol/tinyvol                                0 online
  subvol/tinyvol/data                       177792 online,open
    concat/concat3                          177792 online,tempname,open
      slice/pebbles0                        177792 online,open
```

4. Find or make a free slice on the physvol bambam:

```
xvm:cluster> slice -start 0 -length 177792 bambam
</dev/xvm/bambams0> slice/bambams0
```

5. Attach the slice to tinyvol. There are two different ways to specify the concat volume element to which you are attaching the slice.

The following command attaches the slice by the relative location of the volume element:

```
xvm:cluster> attach slice/bambams0 tinyvol/data/0
</dev/cxvm/tinyvol> concat/concat3
```

The following command attaches the slice by referring to the object name of the volume element:

```
xvm:cluster> attach slice/bambams0 concat3
```

For information on referring to object names and relative locations in XVM commands, see “Object Names in XVM” on page 59.

6. Display the results of the attach command:

```
xvm:cluster> show -top tinyvol
vol/tinyvol                0 online
  subvol/tinyvol/data      355584 online,open
    concat/concat3        355584 online,tempname,open
      slice/pebbles0      177792 online,open
      slice/bambams0      177792 online,open
```

7. Exit the XVM CLI by typing `exit` and then grow the filesystem. Use the mount point where you mounted the filesystem with the CXFS GUI. In this example, the mount point is `/clusterdisk`:

```
xvm:cluster> exit
# xfs_growfs /clusterdisk
meta-data=/clusterdisk      isize=256      agcount=5, agsize=4445 blks
data      =                  bsize=4096   blocks=22224, imaxpct=25
          =                  sunit=0       swidth=0 blks, unwritten=1
naming    =version 1        bsize=4096
log       =internal        bsize=4096   blocks=1168
realtime  =none            extsz=65536  blocks=0, rtextents=0
data blocks changed from 22224 to 44448
```

Mirroring Data on the Logical Volume

The following procedure creates a mirror for the data in the filesystem. Figure 6-13 shows the XVM logical volume after the insertion of the mirror.

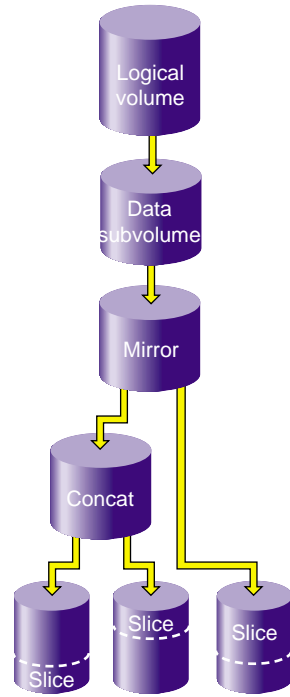


Figure 6-13 XVM Logical Volume After Mirroring

1. Change `tinyvol` to include a mirror container:

```

# xvm
xvm:cluster> insert mirror tinyvol/data/0
</dev/cxvm/tinyvol> mirror/mirror3

```

2. Display the results of the mirror insert:

```
xvm:cluster> show -top tinyvol
vol/tinyvol          0 online
  subvol/tinyvol/data 355584 online,open
    mirror/mirror3    355584 online,tempname,open
      concat/concat3  355584 online,tempname,open
        slice/pebbles0 177792 online,open
        slice/bambams0 177792 online,open
```

3. Find free space or make a new slice of the same size:

```
xvm:cluster> slice -start 0 -length 355584 wilma
</dev/cxvm/wilmas0> slice/wilmas0
```

4. Attach the slice to the mirror:

```
xvm:cluster> attach slice/wilmas0 tinyvol/data/0
</dev/cxvm/tinyvol> mirror/mirror3
```

5. Display the results of the attach. In this example, the revive that was initiated when the slices were attached to the mirror has not yet completed:

```
xvm:cluster> show -top tinyvol
vol/tinyvol          0 online
  subvol/tinyvol/data 355584 online,open
    mirror/mirror3    355584 online,tempname,open
      concat/concat3  355584 online,tempname,open
        slice/pebbles0 177792 online,open
        slice/bambams0 177792 online,open
        slice/wilmas0  355584 online,reviving:11%
```

Converting a Concat to a Stripe Using Mirroring

The following procedure converts the previously created concat to a stripe that replaces the concat in the mirror. Figure 6-14 shows the resulting XVM logical volume.

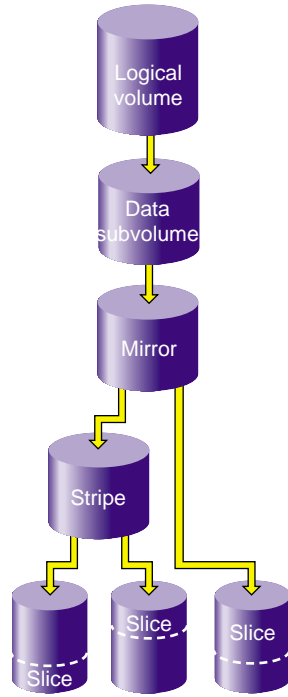


Figure 6-14 XVM Logical Volume after Conversion from Concat to Mirror

1. Break the mirror:

```
xvm:cluster> detach -tempname mirror3/0
</dev/cxvm/vol6> concat/concat3
```

2. Delete the concat object, detaching and keeping the slices that make it up:

```
xvm:cluster> delete -nonslice concat3
</dev/cxvm/pebbles0> slice/pebbles0
</dev/cxvm/bambams0> slice/bambams0
```

3. Create a stripe using the slices:

```
xvm:cluster> stripe -tempname -unit 128 slice/pebbles0 slice/bambams0  
</dev/cxvm/vol7> stripe/stripe0
```

4. Attach the stripe to the mirror:

```
xvm:cluster> attach stripe0 mirror3
```

5. Display the results of the attach. In this example, the revive that was initiated when the stripes were attached to the mirror has not yet completed.

```
xvm:cluster> show -top tinyvol  
vol/tinyvol          0 online  
  subvol/tinyvol/data 355584 online,open  
    mirror/mirror3    355584 online,tempname,open  
      stripe/stripe0  355584 online,tempname,reviving:5%  
        slice/pebbles0 177792 online,open  
        slice/bambams0 177792 online,open  
        slice/wilmas0  355584 online,open
```


Removing a Mirror

The following procedure removes the mirror layer from the logical volume. Figure 6-15 shows the XVM logical volume after the mirror has been removed.

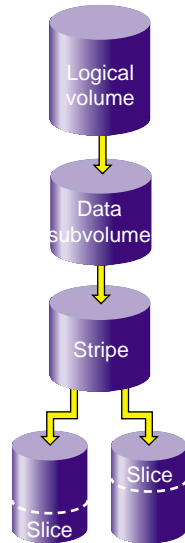


Figure 6-15 XVM Logical Volume after Mirror Removal

1. Detach the slice on which the data is mirrored:

```
xvm:cluster> detach -tempname slice/wilmas0
</dev/cxvm/wilmas0> slice/wilmas0
```

2. Remove the mirror layer:

```
xvm:cluster> collapse mirror3
```

3. Display the results of the collapse command:

```
xvm:cluster> show -top tinyvol
vol/tinyvol                0 online
  subvol/tinyvol/data      355584 online,open
    stripe/stripe0        355584 online,tempname,open
      slice/pebbles0      177792 online,open
      slice/bambams0      177792 online,open
```

Mirroring Individual Stripe Members

The following procedure mirrors the individual slices that make up the stripe. Figure 6-16 shows the XVM logical volume this example yields.

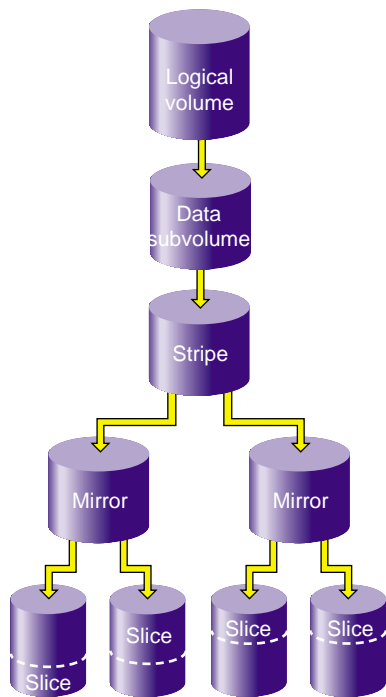


Figure 6-16 XVM Logical Volume after Mirroring Slices

1. Place the slices within mirror containers. The following examples demonstrate alternate methods of specifying slices:

```
xvm:cluster> insert mirror tinyvol/data/0/0
</dev/cxvm/tinyvol> mirror/mirror4
```

```
xvm:cluster> insert mirror slice/bambams0
</dev/cxvm/tinyvol> mirror/mirror5
```

2. Display the results of the two insert commands:

```
xvm:cluster> show -top tinyvol
vol/tinyvol          0 online
  subvol/tinyvol/data 355584 online,open
    stripe/stripe0    355584 online,tempname,open
      mirror/mirror4  177792 online,tempname,open
        slice/pebbles0 177792 online,open
      mirror/mirror5  177792 online,tempname,open
        slice/bambams0 177792 online,open
```

3. Find some free space or reuse some unused slices:

```
xvm:cluster> slice -start 0 -length 177792 betty
</dev/cxvm/bettys0> slice/bettys0

xvm:cluster> show slice/wilmas0
slice/wilmas0          355584 online,autoname
```

4. Attach the slices to the mirrors. Note that wilmas0 is larger than pebbles0. The mirror will continue to use the smallest size.

```
xvm:cluster> attach slice/wilmas0 tinyvol/data/0/0
</dev/cxvm/tinyvol> mirror/mirror4

xvm:cluster> attach slice/bettys0 stripe/0/1
</dev/cxvm/tinyvol> mirror/mirror4
```

5. Display the results of the attach:

```
xvm:cluster> show -top tinyvol
vol/tinyvol          0 online
  subvol/tinyvol/data 355584 online,open
    stripe/stripe1    355584 online,tempname,open
      mirror/mirror4  177792 online,tempname,open
        slice/pebbles0 177792 online,open
        slice/wilmas0  355584 online,open
      mirror/mirror5  177792 online,tempname,open
        slice/bambams0 177792 online,open
        slice/bettys0  177792 online,open
```

Making an XVM Volume Using a GPT Label

SGI storage solutions (TP-9XXX and IS-XXX series) based on LSI RAID are designed to use Automatic Volume Transfer (AVT) failover. This means that as soon as you use one of the alternate controller's path, a failover of the LUN is initiated.

To prevent the LUN from doing controller ping-pong during boot time (when the kernel discovers the disk labels), LSI has created an exclusion zone of 8192 blocks at the beginning and the end of each LUN assigned with a Host Type of SGI AVT.

It is very important when planning your XVM volumes and GPT labels to use this exclusion zone to minimize path failover during boot time. Figure 6-17 shows the optimal way of creating XVM volumes and a GPT label on a LUN using LSI raid:

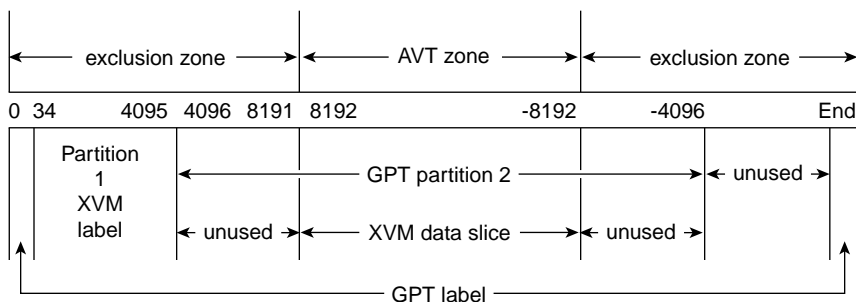


Figure 6-17 Creating XVM Volumes and a GPT Label on a LUN

The `xvm label` command automatically lays out the LUN to meet these requirements.

To stop this automatic layout feature, you can use an option for the `xvm label` command: `-use-exclusion-zones`. Executing `xvm label -use-exclusion-zones` causes `xvm` to allocate the user data area without any consideration of the exclusion zones.

Making a GPT Label

Note: GPT label itself reserves blocks 0-34 for its own label and also 34 blocks at the end of the disk for a backup of the label. More information on the GPT label is available at: http://en.wikipedia.org/wiki/GUID_Partition_Table

To make the GPT label, use the Linux `parted(8)` command. First, you need to create the label and then create one or two partitions. If there are two partitions, one partition will contain the XVM label and the other the XVM user data. If there is only one partition, it will contain first the XVM label data followed by the user data.

Note: The first partition must start on or before block 63. XVM scans the first 64 blocks of the disk to see if there is an XVM label. If no label is found, XVM ignores the disk.

Follow this guideline to make a single partition:

- Create a partition starting at sector 34 and ending at (DISK_SIZE - 8192)

Or follow these guidelines to make two partitions:

- Create a partition 1 starting at sector 34 and ending at sector 8191
- Create a partition 2 starting at sector 8192 and ending at (DISK_SIZE - 8192)

Example Using the `parted` Command

An example using the `parted` command is, as follows:

```
root@xo-xe1 # parted
/dev/xscsi/pci07.00.0/node201000a0b80f200a/port1/lun4/disc
GNU Parted 1.6.25.1
Copyright (C) 1998 - 2005 Free Software Foundation, Inc.
This program is free software, covered by the GNU General Public
License.
```

```
This program is distributed in the hope that it will be useful, but
WITHOUT ANY
WARRANTY; without even the implied warranty of MERCHANTABILITY or
FITNESS FOR A
PARTICULAR PURPOSE. See the GNU General Public License for more
details.
```

```
Using /dev/sdk
(parted) mklabel gpt
(parted) unit s
(parted) print
Disk geometry for /dev/sdk: 0s - 70083583s
Disk label type: gpt
Number  Start   End      Size    File system  Name
Flags
(parted)
```

The example `parted` command, above, creates have a LUN with a GPT label and you know that the LUN is 70083583s long.

Making the Partitions

Create a valid GPT label with the correct partitions:

```
(parted) unit s
(parted) mkpart primary 34 4095
(parted) mkpart primary 4096 70079487
(parted) print
Disk geometry for /dev/sdk: 0s - 70083583s
Disk label type: gpt
Number  Start    End      Size     File system  Name
Flags
1       34s     4095s   4062s
2       4096s   70079487s 70075392s xfs
(parted)
```

Making the XVM Label and Slices

To make the XVM label and slices, perform the following steps:

1. Create the XVM label, as follows:

```
system-a:~/root # xvm
xvm:local> show unlabeled
unlabeled/dev/xscsi/pci-0002:00:01.0-sas-phy4-0x500a0b82fbd13004-lun
0          * *
unlabeled/dev/xscsi/pci-0002:00:01.0-sas-phy4-0x500a0b82fbd13004-lun
1          * *
unlabeled/dev/xscsi/pci0001:00:03.0-1/target1/lun0/disc          * *
unlabeled/dev/xscsi/pci0001:00:03.0-1/target2/lun0/disc          * *
unlabeled/dev/xscsi/pci0002:00:02.0/node20000011c61ddd5d/port1/lun0/
disc          * *
unlabeled/dev/xscsi/pci0002:00:02.0/node20000011c61ddd5d/port2/lun0/
disc          * *
unlabeled/dev/xscsi/pci0002:00:02.1/node20000011c61ddd5d/port1/lun0/
disc          * *
unlabeled/dev/xscsi/pci0002:00:02.1/node20000011c61ddd5d/port2/lun0/
disc          * *
xvm:local> label -name fs2_lun1
unlabeled/dev/xscsi/pci0002:00:02.1/node20000011c61ddd
5d/port2/lun0/disc
fs2_lun1
xvm:local> show phys/fs2_lun1
phys/fs2_lun1          71670988 online,local,accessible
```

2. The next step is to make slices on the lun. Any number of slices may be made, but the usual case is just one slice.

Due to some performance issues in XVM, it is strongly recommended that the XVM slice be completely outside the exclusion zone. The label command by default places the user data area entirely outside of the exclusion zones, so you do not need to consider the exclusion zones in allocating slices. (The `-use-exclusion-zones` flag can be used on the label command if it is necessary to put user data in the exclusion zones.)

In some cases, especially where striped volumes are used, it is important for performance to allocate a slice that is aligned on certain boundaries of the underlying lun. The `-align` option of the slice command adjusts the slice or slices that are created to line up in this manner. For following example, it is assumed that the boundary of interest is every 768KB, which is 1536 sectors.

```
xvm:local> slice -align 1536 phys/fs2_lun1 </dev/lxvm/fs2_lun1s0> slice/fs2_lun1s0
xvm:local> show -v phys/fs2_lun1 XVM physvol phys/fs2_lun1
size: 71670988 blocks sectorsize: 512 bytes state: online,local,accessible
uuid: b4677698-77e0-4434-9155-55e92b2ffa22
system physvol: no
physical drive: /dev/xscsi/pci0002:00:02.1/node20000011c61ddd5d/port2/lun0/disc on host alta
preferred path: unspecified
available paths:
                /dev/xscsi/pci0002:00:02.1/node20000011c61ddd5d/port2/lun0/disc <dev 17280>
affinity=0 <current path>
alternate paths: none
Disk has the following XVM label:
  Clusterid: 0
  Host Name: alta
  Disk Name: fs2_lun1
  Magic: 0x786c6162 (balx)      Version 2
  Uuid: b4677698-77e0-4434-9155-55e92b2ffa22
  last update: Thu May 28 10:54:50 2009
  state: 0x91<online,local,accessible> flags: 0x0<idle>
  secbytes: 512
  label area: 8157 blocks starting at disk block 35 (10 used)
  user area: 71670988 blocks starting at disk block 8192

Physvol Usage:
Start          Length          Name
-----
0              1024             (unused)
1024          71668224         slice/fs2_lun1s0
71669248      1740             (unused)

Local stats for phys/fs2_lun1 since being enabled or reset:
-----
stats collection is not enabled for this physvol
```

The start of the user area is at the end of the exclusion zone at sector 8192, and $8192 + 1024 = 9216$, which is exactly $6 * 1536$.

Converting an SGI DVH XVM Label to a GPT Label Suitable for XVM

A XVM labeled DVH labeled disk has the following format:

Table 6-1 XVM DVH disk layout

Block	Size (in blocks)	Contents
0	1	Label - partition info, etc
1	3071	Usually unused
3072	1024	XVM Metadata
4096	data_length	User Data
	Unused_length	May be unused space at end due to 2MB rounding

A GPT label for XVM has the following format. Partition 1 could start at block 34, but we start it at block 40 for a nice round number. The first GPT partition for the XVM metadata must start within the first 64 blocks of the start of the disk.

Table 6-2 XVM GPT disk layout

Block	Size (in blocks)	Contents
0	1	Master Boot Record
1	1	GUID Partition Table Header
2	32	GUID Partition Entries
40	4056	Partion 1 - XVM Metadata (~2MB)
4096	data_length	Partition 2 - User Data
End - 33	32	Backup GUID Partition Entries
End - 1	1	Backup GUID Partition Table Header

The problem with converting at DVH label to a GPT labeled disk is that a DVH labeled disk can have data going all the way to the end of the disk, where a GPT labeled disk must have up to 33 blocks at the end of the disk: one block for the backup GUID partition table header and up to 32 blocks for the backup GUID partition entries. So you have to

have almost 33 blocks (17KB if 512 bytes/block) free at the end of a disk to have room for the backup GPT.

In some cases, this will be available and the partitioning tools can place the backup GPT in the used space at the end of the disk and be done. This is not normally the case, but can happen especially if the disk is part of a stripe. Usually stripes will not align exactly with the end of a disk and so there may be enough unused space at the end to place the backup GPT header and backup partition entries.

To convert to GPT, from your DVM labeled disk you will need to remember:

start_of_userdata = the block number where the user data started

and

end_block_number = the ending block number of the user data.

Or

data_length = the number of blocks of user data.

For the case where there is room at the end of the disk your new GPT disk will be formatted as follows:

Table 6-3 XVM GPT disk layout example 1

Block	Size (in blocks)	Contents
0	1	Master Boot Record
1	1	GUID Partition Table Header
2	32	GUID Partition Entries
40	4056	Partition 1 - XVM Metadata (~2MB)
4096	data_length	Partition 2 - User Data
End - 33	32	Backup GUID Partition Entries
End - 1	1	Backup GUID Partition Table Header

If you don't have 17KB free at the end of the disk, you can still convert, but it will be more complicated as you will first have to free up 17KB from the end of the disk. Since the DVH user data partition is using this space, we will pick up the last 512KB (or whatever size >17KB that you want) and move it into the first 4096 blocks of the DVH label that is not used, say starting at block 3072 to keep everything on nice boundaries,

So if your DVH labeled disk looks like the following:

Table 6-4 XVM DVH disk layout example 1

Block	Size (in blocks)	Contents
0	1	Label - partition info, etc
1	3071	Usually unused
3072	1024	XVM Metadata
4096	data_length	User Data

You can convert to a GPT labeled disk that will look like the following:

Table 6-5 XVM GPT disk layout example 2

Block	Size (in blocks)	Contents
0	1	Master Boot Record
1	1	GUID Partition Table Header
2	32	GUID Partition Entries
40	3032	Partion 1 - XVM Metadata (~1.5 MB)
3072	data_length	Partition 2 - User data starts here
3072	1024	512KB of data copied from end of disk (make this slice0)
4096	data_length - 1024	Original data less 256KB at end that was moved (make this slice1)
End - 33	33	Backup GUID Partition Entries and header

Now make a concat of `slice1` (the original user data less 256KB at the end of the disk) followed by `slice0` (the last 512KB of data now in the old DVH labels unused space) and the volume will be identical to the volume that was on the DVH labeled disk. These figures show a disk that has only a single slice. This will normally be the case, but in some cases the dvh labeled disk may have more than 1 slice. If this is the case, the XVM `show` command will still show you the the block information needed to rebuild the slices with the GPT partition commands. The offsets into the GPT slice will be identical to the offsets in the DVH slice if no data is needed to be copied to make room for the backup GPT label. If data needs to be copied and 1024 block is the amount that is copied, the Offsets will be +1024 from the DVH numbers, with blocks 0-1024 being the 1024 blocks copied from the end of the slice.

For an example, take a DVH labeled XVM disk with the phyvol name `Volume_B4`. The physvol data can be examined with the `show -v` command:

```
xvm:local> show -v phys/volume_B4
XVM physvol phys/volume_B4
=====
size: 489172992 blocks sectorsize: 512 bytes state: online,local
uuid: 5f41ca2e-c319-1028-8cbe-080069055325
system physvol: no
physical drive: /dev/xscsi/pci01.02.1/node200600a0b8160a4a/port1/lun4/disc on host mvcxfs23
preferred path: unspecified
available paths:
    /dev/xscsi/pci01.02.1/node200600a0b8160a4a/port1/lun4/disc <dev 16688> affinity=0
<current path>
alternate paths: none
Disk has the following XVM label:
Clusterid: 0
Host Name: mvcxfs23
Disk Name: volume_B4
Magic: 0x786c6162 (balx) Version 2
Uuid: 5f41ca2e-c319-1028-8cbe-080069055325
last update: Wed Apr 12 15:06:58 2006
state: 0x11<online,local> flags: 0x0<idle>
secbytes: 512
label area: 1024 blocks starting at disk block 3072 (10 used)
user area: 489172992 blocks starting at disk block 4096
Physvol Usage:
Start      Length      Name
-----
0          489172960  slice/volume_B4s0
489172960   32         (unused)
Local stats for phys/volume_B4 since being enabled or reset:
```

```
-----
stats collection is not enabled for this physvol
```

The output shows us that the user area (shown in bold in the example) starts at block 4096. From the Physvol Usage section we can see that the slice is 489172960 blocks long. So we know that the address of the first block after the data area will be $4096 + 489172960 = 489177056$. We can also see that there are 32 unused blocks after slice/volume_B4s0. This would be 16K, which is not enough for the backup GPT. Thus we will need to copy some data from the end of the user area to free up some space. If we copy the last 512KB (1024 blocks), we would copy starting from the block at 489176032 ($489177056 - 1024$). We could place this in the space that used to belong to the DVH label starting at block 3072 (start of user area - 1024, or in this case $4096 - 1024 = 3072$).

Make sure the volume is unmounted and use the `dd` command to copy the data.

```
xmvcxfs23:~ # dd if=/dev/xscsi/pci01.02.1/node200600a0b8160a4a/port1/lun4/disc
of=/dev/xscsi/pci01.02.1/node200600a0b8160a4a/port1/lun4/disc skip=489176032 seek=3072
count=1024
1024+0 records in
1024+0 records out
```

Skip is the number of blocks to skip to get to the start of the 1024 blocks that we will copy.

Calculate the skip number from the values from the `show -v` that is done on the DVH labeled disk:

$$\begin{aligned} \text{Skip} &= \text{“user data length”} - \text{“amount to be copied”} + \text{“start of user area”} \\ &= 489172960 - 1024 + 4096 \\ &= 489176032 \end{aligned}$$

The seek value is the starting block where the data will be copied to. Again using the values from the `show -v` command:

$$\begin{aligned} \text{Count} &= \text{“start of user area”} - \text{“amount to be copied”} \\ &= 4096 - 1024 \\ &= 3072 \end{aligned}$$

Now that the data is in the right place we can use `parted` to partition and label the disk with GPT. We want our first partition, the one for the XVM metadata to start right after the GPT at the beginning of the disk and go up to where the data will start. Since the starting block for the second partition will be the same as the Count value from the `dd` command the end of the first partition will be count-1. With the `mkpartsect` command

used in `parted`, you specify the block number of the start of the partition and the block number of the end of the partition. So the first partition will end at block 3071. The second partition (for the data) will start at block 3072 (where the data was copied to) and run for the “user data length” from the `show -v` command above. So it starts at 3072 and ends at $3072 + \text{“user data length”} - 1$ which is $3072 + 489172960 - 1$ or 489176031.

```
mvxfs23:~ # parted /dev/xscsi/pci01.02.1/node200600a0b8160a4a/port1/lun4/disc
GNU Parted 1.6.21
Copyright (C) 1998 - 2004 Free Software Foundation, Inc.
This program is free software, covered by the GNU General Public License.
```

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

```
Using /dev/sdt
(parted) mklabel gpt
(parted) mkpartsect primary 40 3071
(parted) mkpartsect primary 3072 489176032
(parted) print
Disk geometry for /dev/sdt: 0.000-238856.000 megabytes
Disk label type: gpt
Minor  Start      End      Filesystem Name      Flags
1       0.020      1.500
2       1.500     238855.484
(parted)quit
```

Now go into XVM and make the slices. The first slice will be the copied data, or the first 1024 blocks in the user data. The second slice is the rest of the partition or the “user data length - 1024”. You then concat the copied data to the end of the original data (minus the copied data), to get a volume back that will exactly match the volume that you started out with.

```
mvxfs23:~ # xvm
xvm:cluster> set domain local
xvm:local> label -name volume_B4 -use-exclusion-zones /unlabeleddev/xscsi/pci01.02.1/node200600a0b81
60a4a/port1/lun4/disc
volume_B4
xvm:local>
xvm:local> slice -start 0 -length 1024 phys/volume_B4
</dev/lxvm/volume_B4s0> slice/volume_B4s0
xvm:local> slice -start 1024 -length 489171936 phys/volume_B4
</dev/lxvm/volume_B4s1> slice/volume_B4s1
xvm:local>
xvm:local> concat -volname testvol slice/volume_B4s1 slice/volume_B4s0
</dev/lxvm/testvol> concat/concat0
xvm:local> show -t vol/testvol
vol/testvol          0 online
  subvol/testvol/data 489172960 online
  concat/concat0     489172960 online,tempname
```

```
slice/volume_B4s1    489171936 online
slice/volume_B4s0    1024 online
```

Looking at the “concat/concat0” line, we can see that our new volume “testvol” is the same size as it was with the DVH label.

Tuning XVM

This section describes some XVM tuning variables in `/proc/sys/dev/xvm` that can be modified using the `sysctl(8)` command.

`xvm_mr_daemon_max` Variable

The `xvm_mr_daemon_max` and `xvm_mr_daemon_min` variables define the number of mirror daemons started for processing mirror retries and mirror completions that cannot be done from completion threads. Adjusting the values can keep more threads alive in the pool rather than spawning and exiting threads which can cause performance issues.

name	default	minimum	maximum
<code>xvm_mr_daemon_min</code>	1	0	255
<code>xvm_mr_daemon_max</code>	25	0	255

Statistics

The XVM Volume Manager can maintain statistics for physvols, subvolumes, stripes, concats, mirrors, and slices.

You use the `stat` option of the `change` command to turn statistics off and on and to reset the statistics for a volume element. Statistics are on by default. When you turn statistics on with the `change` command, statistics are collected only for the layer you specify. If you want to collect statistics for more than one layer of an XVM logical volume, you must specify each layer explicitly.

In a clustered environment, statistics are maintained for the local node only.

Statistics for all volume elements and physvols show the number of read and write operations as well as the number of 512-byte blocks read and written. The following sections describe the specific statistics for each type of volume element:

- “Physical Volume Statistics” on page 158
- “Subvolume Statistics” on page 158
- “Stripe Statistics” on page 158
- “Concat Statistics” on page 160
- “Mirror Statistic” on page 161
- “Slice Statistics” on page 162

Physical Volume Statistics

The following example displays the results of a `show` command to display the statistics of a physical volume for which statistics have been turned on:

```
xvm:cluster> show -stat betty
Local stats for phys/betty since being enabled or reset:
-----
client read requests:          3
client write requests:        42
client 512 byte blks read:    257
client 512 byte blks written: 4681
```

Subvolume Statistics

The following example displays the results of a `show` command to display the statistics of a subvolume for which statistics have been turned on:

```
xvm:cluster> show -stat tinyvol/data
Local stats for subvol/tinyvol/data since being enabled or reset:
-----
read requests:                12
write requests:               109
512 byte blks read:          1034
512 byte blks written:       9533
```

Stripe Statistics

Stripe statistics show the size of the operations versus the size of the stripe width and whether the operations are aligned on a 512-byte boundary. The best performance is obtained when the greatest number of requests are aligned at both start and end.

The following example displays the results of a show command on a stripe for which statistics have been turned on:

```
xvm:cluster> show -v stripe/stripe0
XVM ve stripe/stripe0
=====
volname: vol/cxfsvol  subvolume: subvol/cxfsvol/data
size: 35557888  iou: 1  pieces: 2  open: no
state: 0xa (valid,online) user-flags: online,autoname
uuid: 31350c60-7aa4-1022-85f1-0800690592c9
tid: 922806085 (03/30/99_09:01:25)

Type-specific information:
-----
stripe unit size: 128

Local stats for stripe/stripe0 since being enabled or reset:
-----
read requests:          1826
write requests:         0
512 byte blks read:    15236
512 byte blks written: 0
Requests aligned at both start and end
    equal to stripe width: 10
    greater than stripe width: 10
    less than stripe width: 0
Requests aligned at start
    greater than stripe width: 0
    less than stripe width: 20
Requests aligned at end
    greater than stripe width: 0
    less than stripe width: 19
Requests unaligned
    equal to stripe width: 0
    greater than stripe width: 0
    less than stripe width: 1767

Pieces:
#      Size      Timestamp          Type/Name          State
-----
0      17779016  03/30/99_09:00:49  slice/cxfsdsk1s0   valid,online
1      17779016  03/30/99_09:01:01  slice/cxfsdsk2s0   valid,online
```

Concat Statistics

Concat statistics show the number of operations that are straddled, which are operations that cross the boundary between one piece and the next.

The following example displays the results of a `show` command on a concat for which statistics have been turned on:

```
xvm:cluster> show -v concat/concat3
XVM ve concat/concat3
=====
volname: vol/vol3  subvolume: subvol/vol3/data
size: 11110890  iou: 1  pieces: 5  open: no
state: 0xa (valid,online) user-flags: online,autoname
uuid: 39798cff-9c4f-1022-8544-0800690565c0
tid: 926520978 (05/12/99_09:56:18)

Type-specific information:
-----
(n/a)

Local stats for concat/concat3 since being enabled or reset:
-----
read requests:          10
write requests:         40
512 byte blks read:     640
512 byte blks written: 14080

reads straddling slices:      0
writes straddling slices:     0

Pieces:
#      Size      Timestamp      Type/Name      State
-----
0      2222178 05/12/99_06:48:56 slice/maules0   valid,online
1      2222178 05/12/99_06:48:56 slice/maules1   valid,online
2      2222178 05/12/99_06:51:01 concat/concat0   valid,online
3      2222178 05/12/99_06:48:56 slice/maules3   valid,online
4      2222178 05/12/99_06:48:56 slice/maules4   valid,online
```

Mirror Statistic

In addition to the read and write requests for the mirror, mirror statistics show the mirror synchronization reads and writes.

The following example displays the results of a `show` command on a mirror for which statistics have been turned on:

```
xvm:cluster> show -v mirror/mirror0
XVM ve mirror/mirror0
=====
volname: vol/vol3  subvolume: subvol/vol3/data
size: 11110890  iou: 1  pieces: 1  open: no
state: 0xa (valid,online) user-flags: online,autoname
uuid: 39798d05-9c4f-1022-8544-0800690565c0
tid: 926509854 (05/12/99_06:50:54)

Type-specific information:
-----
rpolicy: 0 (round-robin)  config: 0x0 (none)
drl flush frequency (sec): (n/a) primary piece: 0 reviving: no

Local stats for mirror/mirror0 since being enabled or reset:
-----
read requests:          10
write requests:         40
512 byte blks read:     640
512 byte blks written: 14080
Mirror synchronization reads:      0
Mirror synchronization writes:     0

Leg          Reads          Writes
0            10            40

Pieces:
#    Size    Timestamp          Type/Name          State
-----
0    11110890  05/12/99_09:56:18  concat/concat3    valid,online
```

Slice Statistics

Slice statistics show the number of read and write operations as well as the number of 512K blocks read and written.

The following example displays the results of a `show` command on a slice for which statistics have been turned on:

```
xvm:cluster> show -v slice/cxfsdsk1s0
XVM ve slice/cxfsdsk1s0
=====
volname: vol/cxfsvol  subvolume: subvol/cxfsvol/data
size: 17779016  iou: 1  pieces: 0  open: no
state: 0xa (valid,online) user-flags: online,autoname
uuid: 31350c53-7aa4-1022-85f1-0800690592c9
tid: 922806049 (03/30/99_09:00:49)

Type-specific information:
-----
physvol: cxfsdsk1
start: 0  length: 17779016

Local stats for slice/cxfsdsk1s0 since being enabled or reset:
-----
read requests:          956
write requests:         0
512 byte blks read:    7684
512 byte blks written: 0

Pieces:
#      Size      Timestamp          Type/Name          State
-----
(Ve has no pieces)
```

XVM Volume Manager Operation

This chapter describes various aspects of the way the XVM Volume Manager operates. It includes the following sections:

- “Cluster System Startup”
- “Mirror Revives” on page 164
- “Mirror Revives on Recovery in a Cluster” on page 165
- “XVM Mirror Revive Resources” on page 165
- “XVM Subsystem Parameters” on page 167

Cluster System Startup

When you boot a cluster system that includes XVM logical volumes, the following operations take place:

1. The system boots and probes all disks (SGI SAN disks and FC-hub disks, internal SCSI, etc.)
2. If booting from an XVM system disk, the XVM Volume Manager reads all the XVM labels and creates a local view of all volumes. Cluster volumes are not visible at this point.
3. An `rc` script initializes third-party SAN devices, such as PRISA.
4. If not booting from an XVM system disk, an `rc` script initiates the reading of all the labels and creates a view of all local volumes. Cluster volumes are not visible at this point.
5. The cluster is initialized.
6. On each node in the cluster, the XVM Volume Manager reads all of the labels on the disk and creates a cluster-wide view of all volumes including the third-party SAN volumes.

Note that this procedure implies that XVM volumes are not visible until the cluster has been initialized (i.e., volumes are unavailable in single-user mode). The order of XVM initialization requires that your root device cannot be on a third-party SAN disk.

Mirror Revives

A mirror revive is the process of synchronizing data on the members of a mirror. A mirror revive is initiated at the following times:

- A mirror with more than one piece is initially constructed.
- A piece is attached to a mirror.
- The system is booted with mirrors that are not synchronized.
- A node in a cluster crashes when the mirror is open. For information on this situation, see “XVM Mirror Revive Resources” on page 165.

A message is written to the `syslog` when a mirror begins reviving. Another message is written to the `syslog` when this process is complete. Should the revive fail for any reason, a message will be written to the system console as well as to the `syslog`.

For large mirror components, the process of reviving may take a long time. You cannot halt a mirror revive once it has begun except by detaching all but one of the pieces of the mirror.

There are some mirrors that may not need to revive on creation or when the system reboots. For information on creating these mirrors, see “The `-clean` Mirror Creation Option” on page 48 and “The `-norevive` Mirror Creation Option” on page 48.

While a mirror is in the process of reviving, you can configure the XVM logical volume that contains the mirror, and you can perform I/O to the mirror. Displaying the mirror volume element will show what percentage of the mirror blocks have been synchronized.

If a mirror revive is required while a previously-initiated mirror revive is still occurring, the mirror revive can be queued; this is displayed as the state of the mirror when you display its topology.

You can modify the system performance of mirror revives with the XVM tunable parameters. For information on the XVM tunable parameters that affect mirror revives, see “XVM Mirror Revive Resources” on page 165.

Mirror Revives on Recovery in a Cluster

When a node in a cluster crashes, a mirror in a node may start reviving. This happens when the node that crashed was using the mirror and may have left the mirror in a dirty state, with the pieces of the mirror unequal. When this occurs, it is necessary for the XVM Volume Manager to forcibly resynchronize all of the pieces.

Full mirror resynchronization is performed when a node crashes while the node was using a mirror. This may take some amount of time.

XVM Mirror Revive Resources

If your system performance of mirror revives seems slow, you may need to reconfigure the mirror revive resources. The mirror revive resources are dynamic variables that are set by XVM tunable parameters

You can increase the number of threads and decrease the number of parallel I/O processes that are used for the revive process; this number is controlled by the `xvm_max_revive_rsc` parameter. Decreasing the resources causes less interference with an open file system at the cost of increasing the total time to revive the data.

As a general guideline:

- Increase the `xvm_max_revive_rsc` variable if you want to revive as quickly as possible and do not mind the performance impact on normal I/O processes
- Decrease the `xvm_max_revive_rsc` variable if you want to have a smaller impact on a particular filesystem

Modifying Mirror Revive Resources

Under Linux, you can execute the `modinfo(1M)` command on the XVM module to view descriptions of the XVM tunable parameters, along with their minimum, maximum, and default values.

To change the values of the mirror revive resources while loading the XVM module, you add the tunable parameters to the `insmod(1M)` command. For example, to change the values of `xvm_max_revive_rsc` in the `xvm-standalone.o` module, use the following command:

```
insmod xvm-standalone.o xvm_max_revive_rsc=8
```

To view the current values of the tunable parameters, you can view the contents of the files in `/proc/sys/dev/xvm`, using the `cat` or the `sysctl` command:

```
# cat /proc/sys/dev/xvm/xvm_max_revive_rsc
4

# sysctl dev.xvm.xvm_max_revive_rsc
dev.xvm.xvm_max_revive_rsc = 4
```

If the values are dynamically tunable, then you can change the value by writing to those `/proc/sys/dev/xvm` files:

```
# echo 6 > /proc/sys/dev/xvm/xvm_maxfs_revive
# cat /proc/sys/dev/xvm/xvm_maxfs_revive
6
```

You can also use the `sysctl` command to change the value:

```
# sysctl -w dev.xvm.xvm_maxfs_revive
dev.xvm.xvm_maxfs_revive = 6
```

XVM Subsystem Parameters

The XVM subsystem maintains a set of subsystem parameters that reflect aspects of the XVM kernel that is currently running. These parameters are as follows:

<code>apivers</code>	The version of the library that the kernel is compatible with
<code>config gen</code>	A marker that indicates whether the XVM configuration has changed since the last time the subsystem information was checked
<code>privileged</code>	Indicates whether the current invocation of the XVM cli is privileged and thus capable of making configuration changes (otherwise only viewing is permitted)
<code>clustered</code>	Indicates whether the kernel is cluster-aware
<code>cluster initialized</code>	Indicates whether the cluster services have been initialized

You can view the status of these parameters by using the `-subsystem` option of the `show` command, as in the following example:

```
xvm:local> show -subsystem
XVM Subsystem Information:
-----
apivers:                19
config gen:             15
privileged:             1
clustered:              0
cluster initialized:    0
```

The XVM Manager GUI

When the XVM Volume Manager is used in a clustered environment with CXFS filesystems, you can set up and administer logical volumes with the CXFS Manager graphical user interface (GUI). For information on the CXFS GUI, see the *CXFS 6 Administration Guide for SGI InfiniteStorage*.

In an environment without cluster services enabled, you can set up and administer logical volumes with the XVM Manager graphical user interface (GUI) as a standalone product.

Note: The XVM GUI is not supported on the Red Hat Enterprise Linux Server.

This chapter provides an overview of the XVM Manager GUI. It includes sections on the following topics:

- “Installing the XVM Manager GUI” on page 170
- “Starting the XVM Manager GUI” on page 171
- “The XVM Manager GUI Window” on page 172
- “Configuring the System” on page 178
- “Selecting Items to View or Modify” in Chapter 9
- “Configuring the System Quickly” on page 182
- “Analyzing I/O Performance” on page 183
- “Using Drag-and-Drop for XVM Configuration” on page 184
- “Viewing Log Messages” on page 185
- “Important GUI and CLI Differences” on page 185

This chapter also includes sections on the following topics:

- “Disks Tasks” on page 186

- “Volume Element Tasks” on page 194
- “Filesystem Tasks” on page 205
- “Privileges Tasks” on page 209

Installing the XVM Manager GUI

Table 9-1 shows the set of rpms that you need to install to run the XVM GUI as a standalone system under Linux.

Table 9-1 XVM GUI rpms under Linux

RPM	Description
<code>sysadm_base-client</code>	Admin GUI client, utilities
<code>sysadm_cluster_base-client</code>	Cluster Management GUI software
<code>sysadm_base-lib</code>	System environment, daemons
<code>sysadm_base-server</code>	Admin GUI server
<code>sysadmin_base-tcpmux</code>	GUI communication service
<code>sysadm_xvm-client</code>	XVM volumes admin GUI client, <code>xvmgr</code> executable, desktop icons
<code>sysadm_xvm-server</code>	XVM volumes admin GUI server
<code>sysadm_xvm-web</code>	XVM volume web-based admin

In addition, you need to install the Java J2SE 1.4.2 SDK software available from <http://java.sun.com>.

You should install all of the `sysadm_*` rpms from the same release of SGI Performance Suite 1.3. When upgrading, and you should upgrade all of the rpms at the same time.

If you want to use a Web-based version of the GUI, you must also install a web server (such as Apache) on the system where disks to be administered are attached.

Starting the XVM Manager GUI

There are several methods to start the GUI.

Starting the GUI on Linux

To start the GUI on a system running Linux where the XVM GUI rpms and the Java J2SE 1.4.2 SDK software are installed, enter the following command line:

```
# /usr/bin/xvmgr
```

Starting the GUI on a PC

To use a Web-based version of the GUI on a PC or from a remote location via VPN or WAN, do the following:

1. Install the Java2 v1.4.1 plug-in on your PC.
2. Add the following to your `httpd.conf` file:

```
<Location "/XVMManager">  
    Options Includes ExecCGI FollowSymLinks  
    DirectoryIndex index.html index.shtml  
</Location>
```
3. Close any existing Java windows and restart the Web browser on the PC.
4. Enter the URL `http://server/XVMManager/` (where *server* is the name of the system where disks to be administered are attached).
5. At the resulting webpage, click the XVM Manager icon.

The XVM Manager GUI runs in its own windows outside of your browser. If you launch the XVM Manager GUI from a web browser and exit your browser, all your XVM Manager GUI windows will exit as well.

Logging In

To ensure that the required privileges are available for performing all of the tasks, you should log in to the GUI as root. However, some or all privileges can be granted to any other user by the system administrator using the GUI privilege tasks.

After you start the XVM manager GUI, a dialog box appears prompting you to log in to a host. You can choose one of the following connection types:

Local runs the server-side process on the local host instead of going over the network.

Direct creates a direct socket connection using the `tcpmux` TCP protocol (`tcpmux` must be enabled).

Remote Shell connects to the server via a user-specified command shell, such as `rsh(1C)` or `ssh(1)`. For example:

```
ssh -l root servername
```

Note: For secure connection, choose **Remote Shell** and type a secure connection command using a utility such as `ssh(1)`. Otherwise, XVM Manager GUI will not encrypt communication and transferred passwords will be visible to users of the network.

Proxy connects to the server through a firewall via a proxy server.

Note: You should only make changes from one instance of the GUI running at any given time; changes made by a second GUI instance (a second invocation of `xvmgr`) may overwrite changes made by the first instance. However, multiple XVM Manager windows accessed via the **File** menu are all part of the same application process; you can make changes from any of these windows.

The XVM Manager GUI Window

Figure 9-1 shows the XVM Manager GUI window.

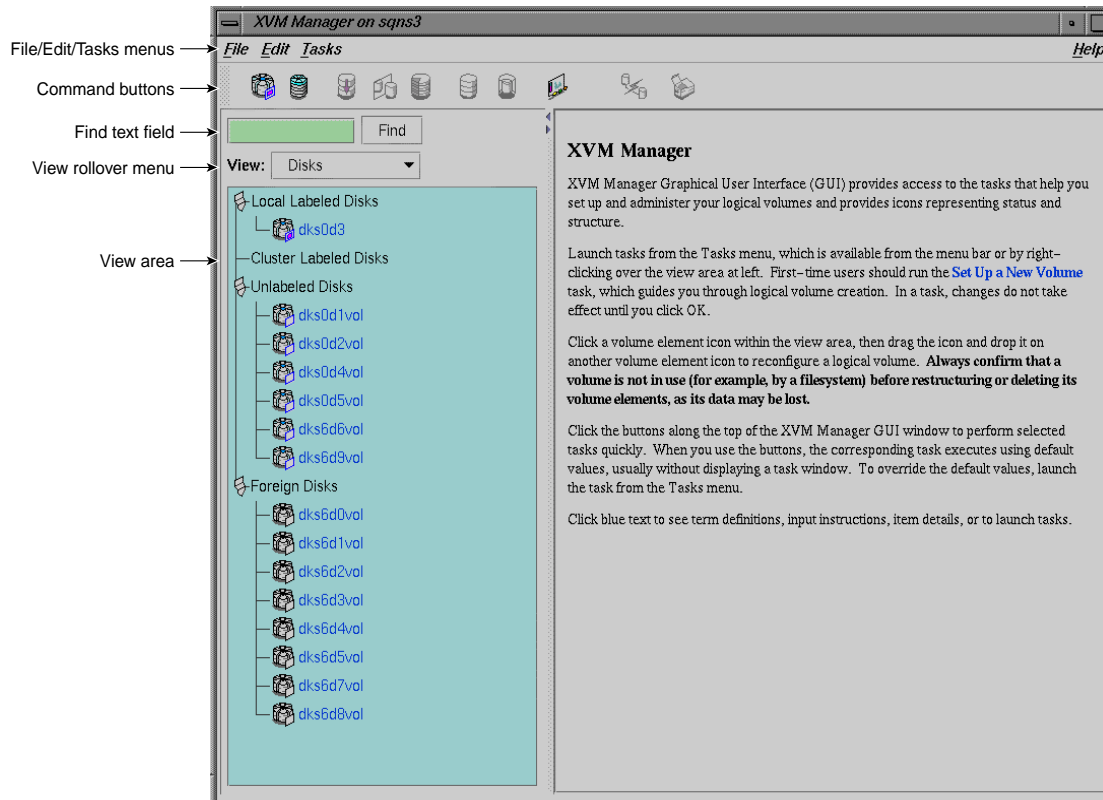


Figure 9-1 XVM Manager GUI Window

The menu bar provides **File**, **Edit**, **Tasks**, and **Help** menus:

- The **File** menu lets you display multiple windows for this instance of the GUI, the `/var/adm/SYSLOG` system log file, and the `/var/sysadm/salog` system administration log file (which shows the commands run by the GUI as it runs them). It also lets you close the current window and exit the GUI completely.
- The **Edit** menu lets you expand and collapse the contents in the View area. You can choose to automatically expand the display to reflect new nodes added to the pool or cluster. You can also use this menu to select all of the current XVM components, or to deselect all currently selected components.

- The **Tasks** menu lets you perform XVM administrative tasks. See “Configuring the System” on page 178 for information on the **Tasks** menu.
- The **Help** menu provides an overview of the GUI and a key to the icons. You can also get help for certain items in blue text by clicking on them.

When you are running XVM as a standalone product, the **Domain** is set to **local**, indicating that the XVM volume you create in this session are in the local domain. For more information about XVM domains, see “XVM Domains” on page 34.

The command buttons along the top of the window provide a method of performing tasks quickly. When you click a button, the corresponding task executes using default values, usually without displaying a task window. For more information on the command buttons, see “Configuring the System Quickly” on page 182.

By default, the window is divided into two sections: the View area to the left and the details area to the right. You can use the arrows in the middle of the window to shift the display.

Use the **Find** text field to view and select single or multiple items, as described in “Selecting Items to View or Modify.”

Choose what you want to appear in the View area from the **View** rollover menu.

To view the details of any XVM volume element, select the item. For information on selecting items, see “Selecting Items to View or Modify.” The configuration and status details for the item will appear in the details area to the right, along with the **Applicable Tasks** list, which displays tasks you may wish to launch after evaluating the item’s configuration details. Click a task to launch it; based on the item selected, default values will appear in the task window. For information on launching tasks, see “Configuring the System” on page 178.

Figure 9-2 shows an XVM Manager GUI window with a selected component and the details area showing the details for that component.

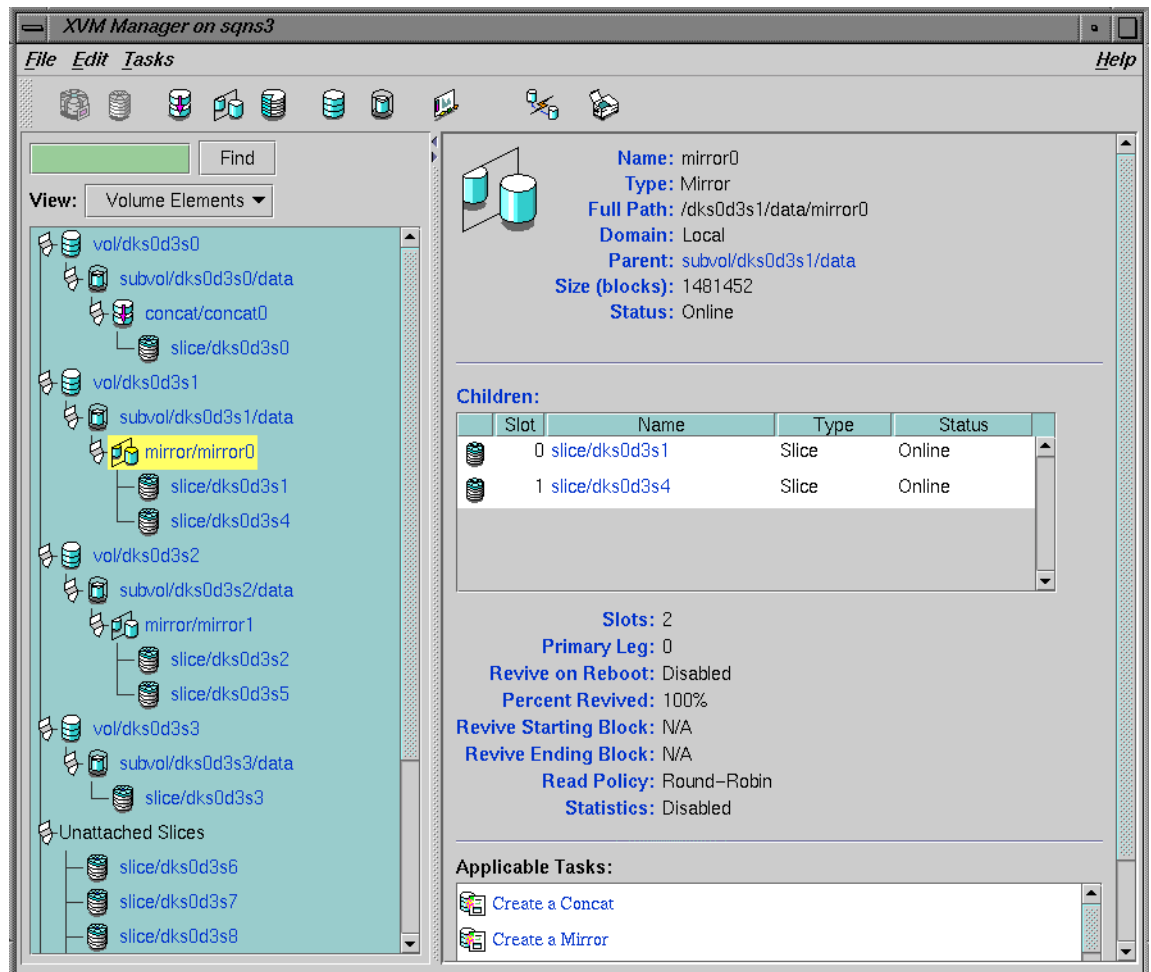


Figure 9-2 XVM Manager GUI Window with Item Selected

To see the configuration and status details about an item in the details area, select its name (which will appear in blue); details will appear in a new window.

In general, clicking on blue text yields a new window display, which could contain one of the following:

- Item details

- Term definitions
- Input instructions
- Task windows

Table 9-2 shows keys to the icons used in the XVM Manager GUI.

Table 9-2 Key to XVM Manager GUI Icons







Icon	Entity
	XVM disk
	Unlabeled disk
	Foreign disk
	Slice
	Volume
	Subvolume
	Concat
	Mirror
	Stripe
	Slot
	Local filesystem
	Expanded tree

Table 9-2 Key to XVM Manager GUI Icons (continued)











Icon	Entity
	Collapsed tree
	Copy on write
	Repository

Table 9-3 shows keys to the states used in the XVM Manager GUI.

Table 9-3 Key to XVM Manager GUI States

Icon	State
	(Grey icon) Defined, offline, inactive or unknown
	Enabled for mount
	(Blue icon) Online, ready for use, up, or mounted without error
	(Green swatch) Open, in use
	(Orange arrow) Mirror reviving
	Disabled
	(Red icon) Error detected, down or mounted with error

Configuring the System

You configure XVM logical volumes with the XVM Manager GUI by performing tasks. Click **Tasks** in the menu bar to view all tasks, which are in submenus organized by category. Right-click in the **View** area to obtain a shorter tasks menu.

The **Tasks** menu contains the following:

- **Guided Configuration**, which consists of a group of tasks collected together to accomplish a larger goal. For example, **Set Up a New Volume** steps you through the process for configuring an XVM volume and allows you to launch the necessary individual tasks by clicking their titles.
- **Volume Elements**, which contains tasks to create, delete, modify, and administer XVM volume elements.
- **Disks**, which contains XVM disk administration tasks.
- **Filesystems**, which contains tasks to define and manage filesystems. For information on the filesystem tasks, see *CXFS 6 Administration Guide for SGI InfiniteStorage*.
- **Privileges**, which let you grant or revoke access to a specific task for one or more users. or information on the privileges tasks, see *CXFS 6 Administration Guide for SGI InfiniteStorage*.
- **Find Tasks**, which lets you use keywords to search for a specific task.

To perform an individual task, do the following:

1. Select the task name from the **Task** menu or click the right mouse button within the View area. For example:

Task > Volume Elements > Create a Concat

The task window appears. Figure 9-3 shows the task window for the **Create a Concat** task.

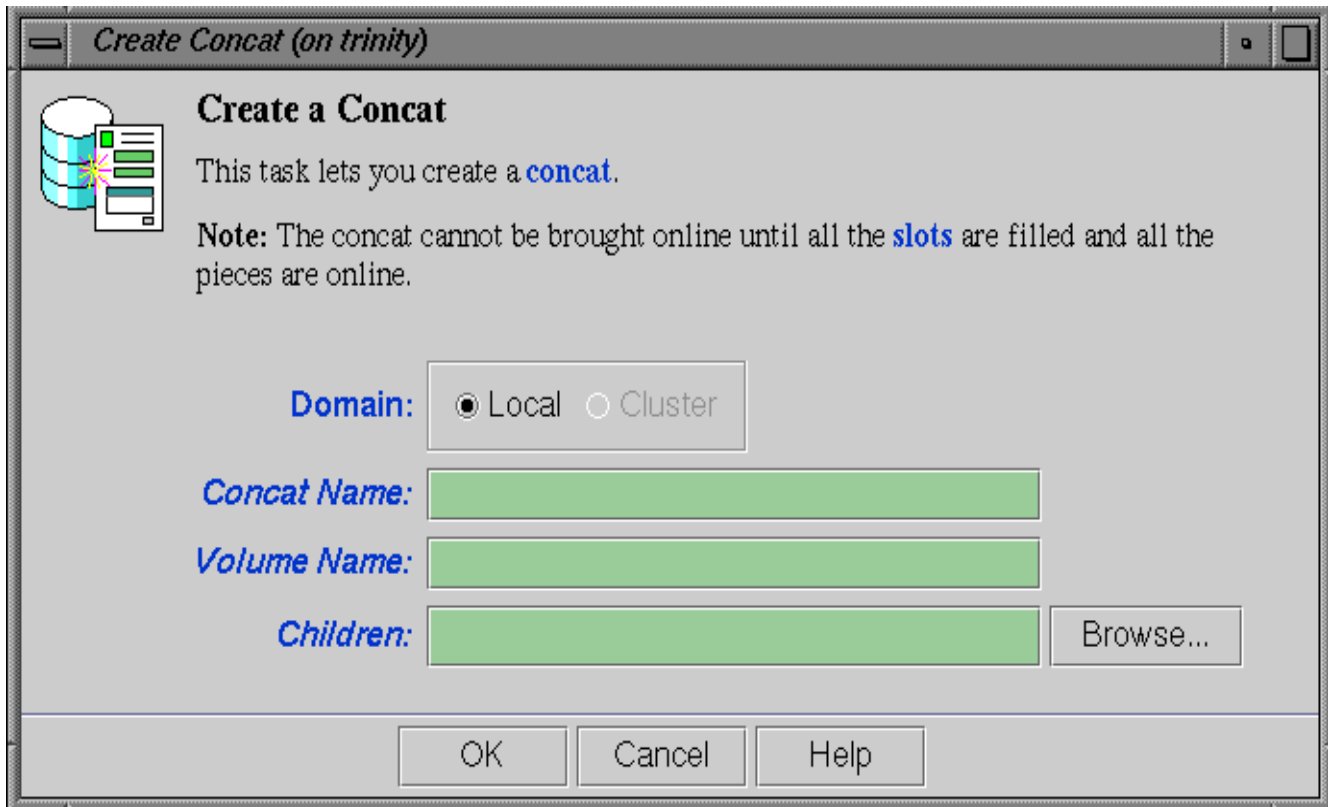


Figure 9-3 Create a Concat Task Window

Note: You can click any blue text to get more information about that concept or input field.

2. Enter information in the appropriate fields and click **OK** to complete the task.

Some tasks consist of more than one page; in these cases, click **Next** to go to the next page, complete the information there, and then click **OK**.

Some tasks include a **Browse** button, which you can click to view and choose task operands. For example, Figure 9-4 shows the task window for the **Label Disks** task.

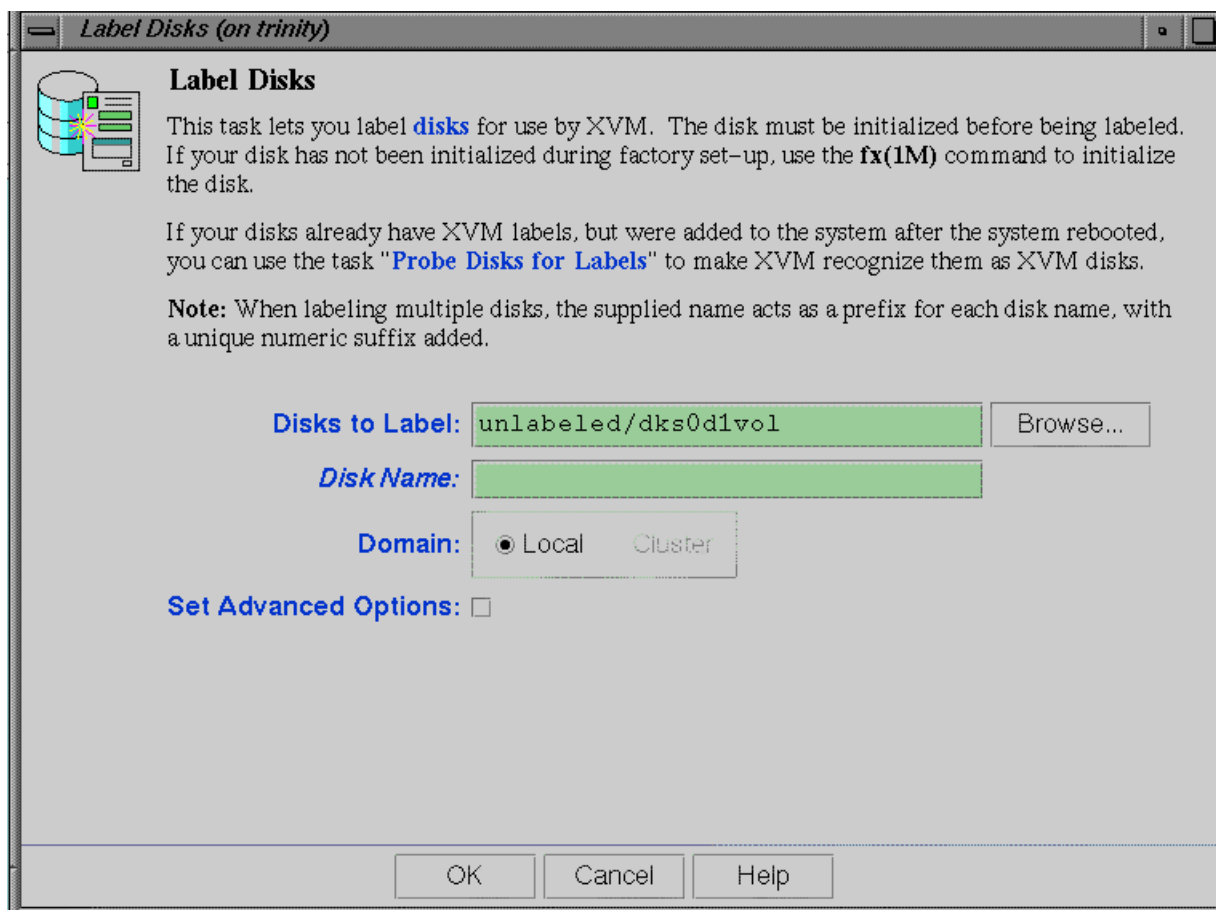


Figure 9-4 XVM Manager GUI Label Disks Task

Clicking on the **Browse** button displays a list of available disks to label, as shown in Figure 9-5. In this window, you can enter a text pattern to match.

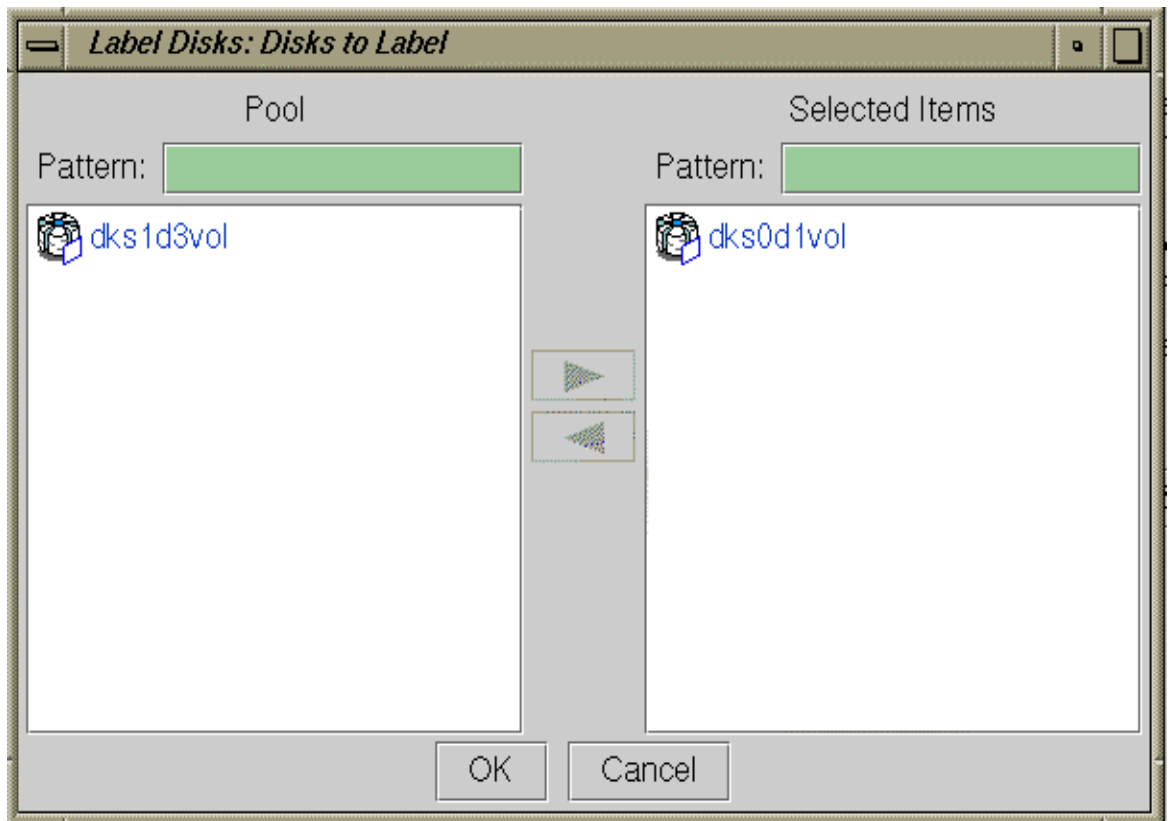


Figure 9-5 XVM Manager GUI Browse Window

Note: In every task, changes do not take effect until you click **OK**.

3. Continue launching tasks as needed.

Selecting Items to View or Modify

Use the following methods to select and deselect items in the **View** area:

- Click to select one item at a time

- Shift+click to select a block of items.
- Ctrl+click items to toggle the selection of any one item

Another way to select one or more items is to type a name into the **Find** text field and then press **Enter** or click the **Find** button.

Configuring the System Quickly

To perform tasks quickly, click the command buttons along the top of the XVM GUI Manager window. When you click a button, the corresponding task executes using default values, usually without displaying a task window. To override the defaults, launch the task from the **Tasks** menu.

Table 9-4 shows the available command buttons.

Table 9-4 XVM Manager GUI Command Buttons











Button	Task
	Labels selected unlabeled disks.
	Brings up the Slice Disk task with the selected disks as default inputs.
	Creates a concat with a temporary name.
	Creates a mirror with a temporary name.
	Creates a stripe with a temporary name.
	Creates a subvolume with a temporary name.

Table 9-4 XVM Manager GUI Command Buttons (**continued**)

Button	Task
	Creates a volume with a temporary name.
	Starts the Performance Co-Pilot XVM I/O monitor <code>pmgxvm</code> on the server, displaying via X Windows to your local administration station. For information on using Performance Co-Pilot with XVM, see “Analyzing I/O Performance” on page 183.
	Detaches the selected volume elements from their current parents.
	Deletes the selected non-slice volume elements or unlabels the selected disks directly, or brings up the appropriate Delete task window for the selected component.

Analyzing I/O Performance

If you want to use Performance Co-Pilot to run XVM statistics, install the default `pcp_eoe` subsystems and also select `pcp_eoe.sw.xvm`. This installs the Performance Co-Pilot PMDA (the agent to export XVM statistics) as an exit operation (`exitop`).

Click the Performance Co-Pilot command button to display a Performance Co-Pilot window that shows all volumes, with colored LEDs indicating read and write I/O activity. Position the cursor over any LED and press the spacebar to view a window showing the value-color legend for the LED and the current value of the read or write rate for the corresponding XVM volume or volume element. Middle-click any LED to get a menu from which you can launch additional tools to show XVM read and write I/O activity charts and a 3D graphical view of disk activity.

For more information on Performance Co-Pilot, see the *Performance Co-Pilot User's and Administrator's Guide*, the *Performance Co-Pilot Programmer's Guide*, and the `dkvis(1)`, `pmie(1)`, `pmieconf(1)`, and `pmlogger(1)` man pages.

Using Drag-and-Drop for XVM Configuration

The XVM Manager GUI allows you to use drag-and-drop to structure volume topologies and to administer XVM disks. The following sections describe drag-and-drop operation.

Caution: Always exercise care when restructuring volume elements, because data that resides on the volume element can be lost during restructuring (dragging and dropping). The GUI attempts to warn the user when it can predict there is a high likelihood of data loss, but, when a volume is not associated with a mounted filesystem, neither the CLI nor the GUI can determine whether that volume holds important data.

Structuring Volume Topologies

To reconfigure a logical volume, select a volume element icon, then drag the icon and drop it on another volume element icon. Icons turn blue as you drag to indicate when it is valid to drop upon them. When you drag, if the mouse cursor reaches the top or the bottom of the **View** area, the display will scroll automatically.

You can use drag-and-drop to operate on multiple volume elements of different types. For example, you can detach several types of volume elements by selecting items and dragging them to any **Unattached** heading, even if no selected item belongs to that category. You can select multiple items of different types and attach them to a parent. For example, you can select two concats and a stripe and use drag-and-drop to attach them to a parent concat.

You can rename volume elements by clicking a selected (highlighted) volume element and typing a new name into the resulting **Name** text field.

Configuring Disks

To label or unlabel disks using drag-and-drop, select **Disks** from the **View** menu. Select an unlabeled disk and then drag and drop it on the **Labeled Disks** heading, or select a labeled disk and then drag and drop it on the **Unlabeled Disks** heading.

You can rename a disk by clicking a selected (highlighted) disk and typing a name into the resulting **Name** text field.

Drag-and-Drop Restrictions

You cannot use drag-and-drop in the following circumstances:

- You cannot drag and drop between two XVM Manager windows.
- You cannot drag and drop items onto command buttons.

Viewing Log Messages

XVM Manager GUI log messages are contained in the following file:

```
/var/sysadm/salog
```

The XVM GUI log, containing all back-end commands and parameters executed by the XVM GUI.

You can also view the contents of these log files using the XVM Manager GUI. To view the messages that appear in the `salog` file as the GUI runs commands, select **File > Show SALog**. To view the messages that appear in the `SYSLOG` file select **File Show SYSLOG**.

Important GUI and CLI Differences

When volume elements other than volumes are created or detached, the system automatically creates a volume and a subvolume that are associated with the volume element. When you create a volume element, you can explicitly name this generated volume, in which case the volume name is stored in label space and persists across machine reboots.

The XVM Manager GUI does not display volumes and subvolumes that were not named explicitly. The GUI displays the children of these volumes and subvolumes as available for use or **Unattached**. In contrast, the command-line interface (CLI) shows all volumes and subvolumes.

The GUI displays filesystems that are on volumes that were not named explicitly, but lists the volumes as `None`. Volumes and subvolumes that the system generated automatically with temporary names are mentioned in the full paths of **Unattached** volume elements (for example, `/vol196/datav`), but the GUI ignores them otherwise.

To reduce the risk of data loss, it is recommended that you name volumes explicitly when using the GUI. If you have created volumes using the CLI that you did not name explicitly, you can use the `xvm` command-line tool to assign these volumes permanent names before proceeding. This can reduce the risk of data loss.

Disks Tasks

This section tells you how to perform XVM administrative tasks on disks using the CXFS Manager GUI or the XVM Manager GUI. When running the XVM Manager GUI as a standalone product, **Domain** is always set to `local` in these tasks.

Label Disks

In order to create XVM logical volumes on a physical disk, you must label the disk as an XVM disk. Labeling a disk writes out an XVM physical volume label on a disk and allows the XVM Volume Manager to control the partitioning on the disk. In a CXFS cluster, any XVM physical volumes that will be shared must be physically connected to all cells in the cluster.

When you label a disk as an XVM physical volume, the first four bytes of logical block one, when represented as ASCII characters, yield `xlab`. This enables you to determine whether a disk is an XVM physical volume even if you are not running the XVM Volume Manager.

If you do not specify a name for the XVM disk, the default name will be the base name of the unlabeled disk path (e.g., `dk0d1`). If you specify a name when assigning multiple disks to XVM, the supplied name acts as a prefix for each physical volume name, with a unique numeric suffix added. If you do not specify a name when assigning multiple disks, the unlabeled disk path is used as the prefix for each physical volume name.

You cannot label a disk as an XVM disk if the disk contains any partitions that are currently in use as mounted filesystems. On systems with many disks, these checks can be time-consuming. You can specify whether you want to override this restriction, and not check for in-use partitions. Use this feature with caution, as data corruption or system panics can result from labeling disks with partitions that are in use.

When you label an XVM disk, you can specify how much space to assign to the volume header; the default value is the number of blocks currently in the volume header of the disk being labeled. You can also specify how much space to assign to the XVM label area

in the volume header; the default is 1024 blocks (usually leaving 3072 blocks in the volume header that are not part of the XVM label area).

The usual default values for the volume header size and label area size support approximately 5000 XVM objects; this should be sufficient for most XVM logical volume configurations. If you will have more than that many objects on the XVM physvol that the label area needs to maintain, you may need to increase the XVM label area size. As a rule of thumb, one block is required for every seven objects. Note that a volume element and a name for a volume element count as two objects.

Although the default size for the XVM label area should be sufficient for most XVM logical volume configurations, you can increase the XVM label area size by shrinking the volume header from the default value and increasing the XVM label area correspondingly. For example, the default options will give you 1024 blocks for the XVM label area and, usually, 3072 blocks for the volume header. The user data then starts at block 4096. If you set the number of volume header blocks to 2048 you can set the number of XVM label blocks to 2048. This will double the XVM label area, shrinking the volume header area from the default and leaving the user data starting at block 4096.

If you add a new disk that has already been labeled as an XVM physical volume to a running system, you must manually probe the disk with the XVM Volume Manager in order for the system to recognize the disk as an XVM disk. You do not have to do this when you are labeling a new XVM disk on your system, however, since the XVM Volume Manager probes the disk as part of the label process. All disks are probed when the system is booted to determine which disks are XVM disks.

Note: When an unlabeled-disk is given to XVM to manage, it is repartitioned so that the raw disk may no longer be directly used for disk I/O. In general, only the volume header and volume partitions remain available. The original partitioning information is saved to a file in the volume header directory under the name `backvh`. This information is restored when the disk is unlabeled.

To label a disk as an XVM disk, do the following:

1. **Disks to Label:** Enter the disk or disks to label or click the **Browse** button to display a list of available disks to label. In the browse window, you can enter a text pattern to match.
2. **Disk Name:** (*Optional*) Enter the new name for the disk or disk to label. When labeling multiple disks, the name you supply will act as a prefix for each disk name, and a unique numeric suffix will be added to make the final name for each disk.
3. **Domain:** Select whether the disk will be defined for use only on the system running the GUI (*Local*) or for use on multiple nodes in a cluster (*Cluster*).
4. **Set Advanced Options:** Selecting this allows you to set the following options:
 - **Check for In-Use Partitions:** De-select this option if you want to label the disk or disks even if partitions are already present.
 - **Volume Header Size (blocks):** Enter the size of the volume header in the disks's label area if you do not want to accept the default size.
 - **Label Area Size (blocks):** Enter the size of the label area if you do not want to accept the default size.
5. Click **OK**.

Slice Disks

Slicing a disk creates a slice from a block range of an XVM physical volume. You can specify the starting block of a slice and you can specify the length of a slice. In addition, you can specify the following methods of creating slices:

- You can create a slice out of all of the blocks of a physical volume
- You can divide a specified address range into equal parts, with each part a different slice
- You can slice multiple physical volumes at once

Slices are named automatically and are persistent across machine reboots. You cannot rename slices.

If a slice length is not supplied, the address range will be from the indicated start block to the end of the free area containing the start block.

When volume elements other than volumes are created, the system automatically creates a volume and a subvolume that are associated with the volume element. When you

create a volume element, you can explicitly name this generated volume, in which case the volume name is stored in label space and persists across machine reboots.

To create slices on an XVM disk, do the following:

1. **Disks to Slice:** Enter the name of the XVM labeled disk on which to define a slice. If you select multiple disks, each disk will be sliced according to the given parameters. Alternately, you can click the **Browse** button to display a list of available disks to slice.
2. **Slice Into:** Fill in the number of equal-sized slices to create.
3. **Operate On:** Select whether you are creating slices out of all the remaining space on the XVM disk or whether you are creating slices out of a specific block range on the disk. If you are slicing a portion of the available space on the disk, enter the following:
 - **Start Block:** (*Optional*) The starting block (in 512-byte blocks) of the area of the disk you want to slice.
 - **Length in Blocks:** (*Optional*) The length in blocks of the area of the disk on which you are creating the slice or slices.
4. Click **OK**.

Rename a Disk

You can rename a physical volume. The name you give a physical volume is persistent across reboots.

To rename an XVM disk with the GUI, do the following:

1. **Disk to Rename:** Choose a disk to rename from the pull-down list.
2. **New Name:** Enter the new name to give to the selected disk.
3. Click **OK**.

Remove XVM Labels from Disks

To remove an XVM physical volume from a system, you remove the XVM label from the physical volume. After a disk is unlabeled, the original partitioning information is restored from a file saved in the volume header directory under the name `backvh`.

In a clustered environment, you cannot unlabel a disk that is not attached to the system you are working from.

You cannot unlabel disks containing slices unless you also delete the slices on those physical volumes. When you indicate that you are deleting all slices on the disk, the slices are deleted even if the slice is part of an open subvolume and its deletion will cause the subvolume state to go offline. If any of the attempts to delete a slice on a disk fails when you are unlabeling a disk, the unlabel will fail. If all deletes succeed, the physical volume will be unlabeled.

To remove XVM labels from one or more disks with the GUI, do the following:

1. **Disks to Unlabel:** Enter the disk or disks to unlabel or click the **Browse** button to display a list of labeled disks and select from that list.
2. Click **OK**.

Modify Statistics Collection on Disks

Statistics collection for an XVM physical volume may be set to on or off. You can change the state of statistics collection on a disk, and you can reset the current statistics to 0.

Statistics for physical volumes show the number of read and write operations as well as the number of 512-byte blocks read and written.

In a clustered environment, statistics are maintained for the local cell only.

To modify statistics collection on one or more disks with the GUI, do the following:

1. **Disks to Modify:** Enter the disk or disks on which to modify statistics collection or click the **Browse** button to display a list of labeled disks and select from that list.
2. **Statistics Collection:** Turn statistics collection on or off, or reset the current statistics to 0.
3. Click **OK**.

Give Away Ownership of Disks

You change the domain of an XVM physical volume by giving ownership of that physical volume to another machine or cluster.

You cannot give away ownership of a physical volume that has slices that are part of open subvolumes. For this reason, an attempt to give away a disk will fail while a mirror revive is active. In general, you must unmount filesystems on XVM logical volumes that contain the XVM physvol and wait for mirror revives to complete before giving away a physical volume.

Giving a disk away will result in all slices on the disk (and any empty parents that result) being deleted on the current host, as well as the physical volume. The configuration information will be retained on the disk. Subvolumes that span disks might go offline if giving a disk away will cause slices belonging to that physvol to be removed.

When you give a disk away, the new owning node or cluster must read the disk before the configuration is visible to the new owner. This happens in either of two ways: automatically on reboot, or when the new owner probes the new disk.

You can specify a physical volume to give away by either the name of the physical volume name or by the name of the disk itself.

To give away ownership of one or more disks to another host or cluster, do the following:

1. **Disks to Give Away:** Enter the disk or disks to give away or click the **Browse** button to display a list of disks and select from that list.
2. **Give Disks To:** Select the new host or new cluster for the disks.
3. Click **OK**.

This task is valid in a clustered environment only.

Steal a Foreign Disk

In some circumstances, the node or cluster that currently owns the physical volume may be unable to give a disk away. In these cases, you can steal a disk to change the domain of an XVM physical volume. Only disks which are foreign to the current node or cluster can be the targets of a steal.

Caution: Stealing a disk unconditionally resets the owner of an XVM physical volume to the current node or cluster. No attempt is made to inform the previous owner of the change in ownership. If another host or cluster has the XVM physvol instantiated, this could result in configuration corruption. You should steal a disk only when ownership

cannot be changed by giving the disk away. In a situation where you need to steal a disk to change the domain of an XVM physical volume, you may not know the name of the current owner of the physical volume.

You cannot use steal a physical volume that has slices that are part of open subvolumes. In general, you must unmount filesystems on XVM logical volumes that contain the XVM physvol and wait for mirror revives to complete before stealing the XVM physvol.

To take control of a foreign disk with the GUI, do the following:

1. **Foreign Disk to Steal:** Enter the disk to steal or click the **Browse** button to display a list of foreign disks and select from that list.
2. **Bring To:** Select whether you are bringing the foreign disk to the local domain or to the cluster domain.
3. Click **OK**.

This task is valid in a clustered environment only.

Probe Disks for Labels

If you add a new disk that has already been labeled as an XVM physical volume to a running system, you must manually probe the disk in order for the system to recognize the disk as an XVM disk. You do not have to do this when you are labeling a new XVM disk on your system, however, since the XVM Volume Manager probes the disk as part of the label process.

It is assumed that the disk to be probed is available in the hardware inventory (the controller that it is connected to has been probed outside of XVM).

If the disk being probed has not been previously labeled by XVM, an error is returned.

To probe one or more disks for XVM label information with the GUI, do the following:

1. **Disks to Probe:** Enter the disk to probe or click the **Browse** button to display a list of disks and select from that list.
2. Click **OK**.

Dump Volume Element or Physical Volume Structure to File

You can dump configuration information for an individual volume element, or you can dump the configuration information for all of the volume elements under the volume element you specify. You can also use dump configuration commands for a physical volume; you must explicitly dump the physical volume separately from a volume element tree. Dumping configuration information allows you to replace a disk in a running system and to regenerate the XVM configuration on the new disk without rebooting the system.

When you dump and regenerate a device, you do not regenerate the data on the disk you are replacing, but rather you regenerate the XVM configuration on the new disk.

When you dump a volume element, a new uuid is generated for the object being dumped in order to avoid any possible name collision issues when the object is later re-created.

To dump the XVM configuration commands of one or more volume elements and disks to a file, do the following:

1. **File Name:** Enter the name of the file to which you want to dump the configuration commands for the XVM object.
2. **Domain:** Set the domain of the volume element or physvol to dump; this selection determines what will appear when you click the **Browse** button.
3. **All disks and volume elements:** Indicate whether you want to dump all disks and volume elements. If you deselect this option, you can specify the following:
 - **Disks to Dump:** Enter the name of the disks to dump, or click the **Browse** button to display a list of disks and select from that list.
 - **Volume Elements to Dump:** Enter the name of the volume elements to dump, or click the **Browse** button to display a list of volume elements and select from that list.

If you dump selected volume elements, you can indicate whether to dump the descendents of the specified volume element as well.
4. Click **OK**.

Volume Element Tasks

This section tells you how to perform XVM administrative tasks on volume elements using the CXFS Manager GUI or the XVM Manager GUI. When running the XVM Manager GUI as a standalone product, **Domain** is always set to `local` in these tasks.

Create a Concat

A concat is a volume element that concatenates all of its child volume elements into one address space.

When volume elements other than volumes are created, the system automatically creates a volume and a subvolume that are associated with the volume element. When you create a volume element, you can explicitly name this generated volume, in which case the volume name is stored in label space and persists across machine reboots.

To create a concat, do the following:

1. **Domain:** Set the domain that will own the concat.
2. **Concat Name:** (*Optional*) Enter a name for the new concat. If you do not specify a name, a temporary name is generated. A temporary name is not guaranteed to be persistent across reboots.
3. **Volume Name:** (*Optional*) Enter a name for the volume to create. If you do not specify a name, a temporary name is generated. A temporary name is not guaranteed to be persistent across reboots.
4. **Children:** (*Optional*) Enter one or more volume elements to be children of the new concat, or click the **Browse** button to display a list of volume elements and select from that list.
5. Click **OK**.

Create a Mirror

A mirror is a volume element that mirrors all of its child volume elements. A mirror cannot have more than eight members.

When you create a mirror that has more than one piece, the mirror begins the reviving process; This means that the system is begins the process of mirroring the data, synchronizing the data in each of the legs.

For large mirror components, this revive process may take a long time. When you are creating a new mirror that does not need to be revived, you should consider specifying that the mirror does not need to revive on creation.

When you are creating a new mirror that you will use for scratch filesystems such as `/tmp` or `swap` that will never need to be revived, you should consider specifying that the mirror will never revive.

You cannot halt a mirror revive once it has begun except by detaching all but one of the pieces of the mirror.

The XVM Volume Manager allows you to specify one of the following read policies for a mirror:

- | | |
|-------------|---|
| round-robin | Balance the I/O load among the members of the mirror, blindly reading in a round-robin fashion. |
| sequential | Route sequential I/O operations to the same member of the round-robin |

Balance the I/O load among the members of the mirror, blindly reading in a round-robin fashion.

You can specify whether a particular leg of a mirror is to be preferred for reading by marking it as a primary leg.

The components of a mirror do not have to be identical in size, but if they are not there will be unused space in the larger components.

When volume elements other than volumes are created, the system automatically creates a volume and a subvolume that are associated with the volume element. When you create a volume element, you can explicitly name this generated volume, in which case the volume name is stored in label space and persists across machine reboots.

To create a mirror, do the following:

1. **Domain:** Set the domain that will own the mirror.
2. **Mirror Name:** (*Optional*) Enter a name for the new mirror. If you do not specify a name, a temporary name is generated. A temporary name is not guaranteed to be persistent across reboots.
3. **Volume Name:** (*Optional*) Enter a name for the volume to create. If you do not specify a name, a temporary name is generated. A temporary name is not guaranteed to be persistent across reboots.
4. **Read Policy:** Select the read policy for the mirror.
5. **Revive Option:** Select the revive option for the mirror.
6. **Children:** (*Optional*) Enter one or more volume elements to be children of the new mirror, or click the **Browse** button to display a list of volume elements and select from that list.
7. **Primary Leg:** (*Optional*) Select the volume element to be the primary leg of the mirror.
8. Click **OK**.

Create a Stripe

A stripe is a volume element that stripes a set of volume elements across an address space.

You can create a stripe that is made up of volume elements of unequal size, although this will leave unused space on the larger volume elements.

The actual size of the stripe volume element depends on the stripe unit size and the size of the volume elements that make up the stripe. In the simplest case, the volume elements are all the same size and are an even multiple of the stripe unit size. For example, if the stripe unit is 128 512-byte blocks (the default stripe unit size), and you create a stripe consisting of two slices that are each 256,000 blocks, all the space of each of the slices is used. The stripe size is the full 512,000 blocks of the two slices.

On the other hand, if two slices that make up a stripe are each 250,000 blocks and the stripe unit is 128 blocks, then only 249,984 of the blocks on each slice can be used for the stripe and the size of the stripe will be 499,968 blocks. This situation may arise when you create the slices on a disk by dividing the disk equally, or use the entire disk as a slice, and do not coordinate the resulting stripe size with the stripe unit size.

Even if one of the two slices that make up the two-slice stripe in the second example is 256,000 blocks (while the other is 250,000 blocks), the stripe size will be 499,968 blocks, since the same amount of space in each volume element that makes up the slice is used.

The general formula for determining what the stripe size will be is the following, where *stripe_width* is the number of volume elements that make up the stripe:

$$\text{stripe_size} = (\text{smallest_stripe_member} / \text{stripe_unit}) * \text{stripe_unit} * \text{stripe_width}$$

Note that this formula uses integer arithmetic.

If your XVM configuration requires that you spread I/O across controllers, you must have a complete `failover.conf` file configured. This is necessary to ensure that I/O is restricted to a chosen primary path. For example, if you want a striped volume to span two host bus adapters, you must configure a `failover.conf` file to specify a primary path.

For information on configuration of failover for storage devices, see your SGI support provider. Information on the `failover.conf` file can also be found on the `failover(7M)` man page and in the `/etc/failover.conf` file itself.

When volume elements other than volumes are created, the system automatically creates a volume and a subvolume that are associated with the volume element. When you create a volume element, you can explicitly name this generated volume, in which case the volume name is stored in label space and persists across machine reboots.

To create a stripe, do the following:

1. **Domain:** Set the domain that will own the stripe.
2. **Stripe Name:** (*Optional*) Enter a name for the new stripe. If you do not specify a name, a temporary name is generated. A temporary name is not guaranteed to be persistent across reboots.
3. **Volume Name:** (*Optional*) Enter a name for the volume to create. If you do not specify a name, a temporary name is generated. A temporary name is not guaranteed to be persistent across reboots.
4. **Stripe Unit Size:** Enter a stripe unit size for the stripe as a number of 512-byte blocks).
5. **Children:** (*Optional*) Enter one or more volume elements to be children of the new stripe, or click the **Browse** button to display a list of volume elements and select from that list.
6. Click **OK**.

Delete Volume Elements

When you delete volume elements, parents of deleted volume elements remain and have open slots.

If a volume element you delete is part of an open subvolume, its deletion cannot cause the subvolume state to go offline.

If a volume element contains any attached children, it cannot be deleted. However, the **Delete a Volume Elements** task provides two options that override this restriction: You delete the volume element and all volume elements below it, or you delete a volume element and all non-slice volume elements below it, detaching and keeping the slices.

If a slice is mapped to a partition, the underlying partition will be removed from the partition table.

To delete one or more volume elements, do the following:

1. **Domain:** Set the domain of the volume elements to delete; this selection determines what will appear when you click the **Browse** button.
2. **Volume Elements to Delete:** Enter the volume elements to delete or click the **Browse** button to display a list of volume elements and select from that list.

3. **Also Delete Descendants:** Select whether to delete the selected volume elements only, the selected volume elements and all their descendants except slices, or the selected volume elements and all their descendants including slices.
4. Click **OK**.

Rename a Volume Element

You can rename a volume element. The name you give an object when you explicitly rename it remains persistent across reboots. You cannot change the name of a slice.

To rename a volume element, do the following:

1. **Domain:** Set the domain of the volume element to delete. This selection determines what will appear in the list of available volume elements to rename.
2. **Volume Element to Rename:** Enter the volume element to rename or choose an element from the pull-down menu.
3. **New Name:** Enter the new name for the volume element.
4. Click **OK**.

Insert Mirrors or Concats above Volume Elements

You can insert a mirror or a concat volume element above another volume element. You cannot insert a volume element above a volume or a subvolume.

You can grow a volume element on a running system by inserting a concat and you can add mirroring on a running system by inserting a mirror. The volume element you are growing or mirroring can be part of an open subvolume and can have active I/O occurring.

To insert a new mirror or concat above an existing volume element, do the following:

1. **Domain:** Set the domain for the new volume element to create.
2. **Volume Elements to Insert Above:** Enter the volume elements above which to insert the mirror or concat or click the **Browse** button to display a list of volume elements and select from that list. A mirror or concat will be created for each volume element you select.

3. **Insert:** Choose to insert either mirrors or concats above the selected volume elements.
4. Click **OK**.

Remove Unneeded Mirrors and Concats

You can remove an unneeded mirror or concat from an XVM tree if the mirror or concat has only one child. This task links the child to the parent of the mirror or concat that you remove.

You can remove a mirror or concat in an open subvolume. Generally, you perform this task during configuration to reverse a previous insert operation.

To remove mirrors and concats with only one child, do the following:

1. **Domain:** Set the domain for the mirrors and concats to remove. This selection determines what will appear when you click the **Browse** button.
2. **Mirrors and Concats to Remove:** Enter the mirrors and concats to remove or click the **Browse** button to display a list of volume elements and select from that list.
3. Click **OK**.

Enable Volume Elements

When a volume element is disabled no I/O will be issued to it until you explicitly enable it.

To enable one or more volume elements that have been disabled, do the following:

1. **Domain:** Set the domain for the volume elements to enable. This selection determines what will appear when you click the **Browse** button.
2. **Volume Elements to Enable:** Enter the volume elements to enable or click the **Browse** button to display a list of volume elements and select from that list.
3. Click **OK**.

Disable Volume Elements

When a volume element is disabled no I/O will be issued to it until explicitly enable the volume element. An object remains disabled across machine reboots.

To disable one or more volume elements, do the following:

1. **Domain:** Set the domain for the volume elements to disable. This selection determines what will appear when you click the **Browse** button.
2. **Volume Elements to Disable:** Enter the volume elements to disable or click the **Browse** button to display a list of volume elements and select from that list.
3. Click **OK**.

Bring Volume Elements Online

A volume element is online when it is properly configured. It is able to be opened, or it is already open.

The system kernel may disable a volume element and take that element offline. This could happen, for example, when a mirror member shows an I/O error.

To bring a set of volume elements online, do the following:

1. **Domain:** Set the domain for the volume elements to bring online. This selection determines what will appear when you click the **Browse** button.
2. **Volume Elements to Bring Online:** Enter the volume elements to bring online or click the **Browse** button to display a list of volume elements and select from that list.
3. Click **OK**.

Remake a Volume Element

When you remake a volume element, you collapse holes in the volume element. The child volume elements remain in their current order.

To remake a volume element, do the following:

1. **Domain:** Set the domain. This selection determines what will appear in the list of available volume elements to remake.
2. **Volume Element:** Enter the volume element to collapse or select a volume element from the pull-down menu.
3. Click **OK**.

Detach Volume Elements

You can detach volume elements from their parents.

If a volume element you detach is part of an open subvolume, its detachment cannot cause the subvolume state to go offline. Any command that would reduce the address space of an open subvolume, such as detaching a slice that is not mirrored, will cause it to go offline. You cannot detach the last valid piece of an open mirror from that mirror, since this will cause the mirror to go offline.

To detach volume elements, do the following:

1. **Domain:** Set the domain for the volume elements to detach. This selection determines what will appear when you click the **Browse** button.
2. **Volume Elements to Detach:** Enter the volume elements to detach or click the **Browse** button to display a list of volume elements and select from that list.
3. Click **OK**.

Create a Volume

When you create a volume, you can specify subvolumes to attach to the volume after it is created.

A volume cannot have more than one system-defined subvolume of a given type. The system-defined subvolumes are data subvolumes and log subvolumes.

To create a volume, do the following:

1. **Volume Name:** (*Optional*) Enter a name for the new volume. If you do not specify a name, a temporary name is generated. A temporary name is not guaranteed to be persistent across reboots.
2. **Domain:** Set the domain that will own the volume.
3. **Volume Children:** Select one of three options:
 - **Add no children:** Create a volume with no attached subvolumes.
 - **Add child subvolume:** Create a volume and attach an existing subvolume that you specify.

- **Add child to a new data subvolume:** Create a volume and attach an existing volume element to the new data subvolume that is created.
4. Click **OK**.

Create a Subvolume

When you create a subvolume you can optionally attach a specified volume element to the subvolume. The volume element attached to the subvolume cannot be a volume or another subvolume.

You can create a subvolume of a system-defined type of data, log, or you can create a subvolume of a user-defined type.

There cannot be more than one subvolume having the same system-defined type under a volume. For example, there can be only one data subvolume under a volume. There is no such restriction for the user-defined subvolume types.

A user-defined subvolume type is in the range 16 through 255 (0 through 15 are reserved for system-defined types). User-defined subvolume types are used to associate an application-dependent type with a subvolume. Subvolume types in the user-defined range are not interpreted in any way by the system. You can specify more than one subvolume of a specific user-defined type under a volume.

When you create a subvolume, you specify the user-id (uid) of the subvolume owner (default 0), the group-id of the subvolume owner (default 0), and the mode of the subvolume (default is 0644).

If you do not specify a child volume element, an empty subvolume is created, which can be attached to later.

When volume elements other than volumes are created, the system automatically creates a volume and a subvolume that are associated with the volume element. When you create a volume element, you can explicitly name this generated volume, in which case the volume name is stored in label space and persists across machine reboots.

To create a subvolume, do the following:

1. **Domain:** Set the domain that will own the subvolume.
2. **Subvolume Type:** Specify whether the subvolume is a data, log, or user-defined volume type.

3. Click **Next** to move to the next page.
4. **Attach to Volume:** (*Optional*) Select an existing volume as a parent for the new subvolume.
5. **Child Volume Element:** (*Optional*) Select an existing volume element to be the child of the subvolume.
6. **User ID:** Enter the ID of the user that owns the block and character special files corresponding to the subvolume.
7. **Group ID:** Enter the ID of the group that owns the block and character special files corresponding to the subvolume.
8. **Mode:** Enter the file mode of the block and character special files corresponding to the subvolume.
9. Click **OK**.

Modify Subvolumes

When you create a subvolume, you specify the user-id (uid) of the subvolume owner (default 0), the group-id of the subvolume owner (default 0), and the mode of the subvolume (default is 0644). You can change these parameters when you modify a subvolume.

To change the parameters of one or more subvolumes, do the following:

1. **Domain:** Set the domain that will own the subvolumes.
2. **Subvolumes to Modify:** Select one or more subvolumes to modify.
3. **User ID:** Enter the new ID of the user that owns the block and character special files corresponding to the subvolume.
4. **Group ID:** Enter the new ID of the group that owns the block and character special files corresponding to the subvolume.
5. **Mode:** Enter the new file mode of the block and character special files corresponding to the subvolume.
6. Click **OK**.

Modify Statistics Collection on Volume Elements

Statistics collection for a subvolume, stripe, concat, mirror, or slice may be set to on or off. You can change the state of statistics collection on a volume, and you can reset the current statistics on a volume element to 0. Statistics are collected only for the volume element layer you specify. If you want to collect statistics for more than one layer of an XVM logical volume, you must specify each layer explicitly.

Statistics for volume elements show the number of read and write operations as well as the number of 512-byte blocks read and written.

In a clustered environment, statistics are maintained for the local cell only.

To modify statistics collection on one or more volume elements, do the following:

1. **Domain:** Set the domain for the subvolumes to modify. This selection determines what will appear when you click the **Browse** button.
2. **Volume Elements to Modify:** Select one or more volume elements to modify.
3. **Statistics Collection:** Turn statistics collection on or off, or reset the current statistics to 0.
4. Click **OK**.

Filesystem Tasks

This section tells you how to perform XVM administrative tasks on filesystems using the XVM Manager GUI.

Make Filesystems

This task lets you create a filesystem on a volume that is online but not open. To create filesystems on multiple volume elements, use the **Browse** button.

Caution: Clicking **OK** will erase all data that exists on the target volume.

To make a filesystem, do the following:

1. Enter the following information:
 - **Domain:** Select the domain that will own the volume element to be created. Choose **Local** if the volume element or disk is defined for use only on the node to which the GUI is connected, or choose **Cluster** if it is defined for use on multiple nodes in the cluster.
 - **Volume Element:** Select the volumes on which to create the filesystem or select the volume elements whose parent volumes will be used for the filesystems. The menu lists only those volume elements that are available. (When volume elements other than volumes are created or detached, the system automatically creates a volume and a subvolume that are associated with the volume element. If you did not explicitly name an automatically generated volume, the GUI will display its children only.)
 - **Specify Sizes:** Check this box to modify the default options for the filesystem, including data region size, and log size.

By default, the filesystem will be created with the data region size equal to the size of the data subvolume. If the volume contains a log subvolume, the log size will be set to the size of the log subvolume.

2. If you checked the **Specify Sizes** box, click **Next** to move to page 2. On page 2, enter the following information. For more information about these fields, see the `mkfs_xfs(1M)` man page.
 - **Block Size:** Select the fundamental block size of the filesystem in bytes.
 - **Directory Block Size:** Select the size of the naming (directory) area of the filesystem in bytes.
 - **Inode Size:** Enter the number of blocks to be used for inode allocation, in bytes. The inode size cannot exceed one half of the **Block Size** value.
 - **Maximum Inode Space:** Enter the maximum percentage of space in the filesystem that can be allocated to inodes. The default is 25%. (Setting the value to 0 means that the entire filesystem can become inode blocks.)
 - **Flag Unwritten Extents:** Check this box to flag unwritten extents. If unwritten extents are flagged, filesystem write performance will be negatively affected for preallocated file extents because extra filesystem transactions are required to convert extent flags for the range of the file.

You should disable this feature (by unchecking the box) if the filesystem must be used on operating system versions that do not support the flagging capability.

- **Data Region Size:** Enter the size of the data region of the filesystem as a number of 512-byte blocks. This number is usually equal to the size of the data subvolume. You should specify a size other than 0 only if the filesystem should occupy less space than the size of the data subvolume.
 - **Use Log Subvolume for Log:** Check this box to specify that the log section of the filesystem should be written to the log subvolume of the XVM logical volume. If the volume does not contain a log subvolume, the log section will be a piece of the data section on the data subvolume.
 - **Log Size:** Enter the size of the log section of the filesystem as a number of 512-byte blocks. You should specify a size other than 0 only if the log should occupy less space than the size of the log subvolume.
3. Click **OK**.

Grow a Filesystem

This task lets you grow a mounted filesystem.

Note: Before you can grow a filesystem, you must first increase the size of the logical volume on which the filesystem is mounted. You can launch the **Insert Mirrors or Concats above Volume Elements** task to add a concat, or you can use the drag-and-drop mechanism to attach a slice to an existing concat. You cannot add a slice to an existing volume element if this changes the way that the data is laid out in that volume element or in any ancestor of that volume element.

To grow a filesystem, do the following:

1. Enter the following information:
 - **Filesystem:** Select the name of the filesystem you want to grow. The list of available filesystems is determined by looking for block devices containing XFS superblocks.
 - **Specify Sizes:** Check this option to modify the default options for the filesystem, including data region size and (if already present for the filesystem) log size.

By default, the filesystem will be created with the data region size equal to the size of the data subvolume. If the volume contains a log subvolume, the log size will be set to the size of the log subvolume.

2. If you checked the **Specify Sizes** box, click **Next** to move to page 2. For more information about these fields, see the `mkfs_xfs(1M)` man page.
 - **Data Region Size:** Enter the size of the data region of the filesystem as a number of 512-byte blocks. This number is usually equal to the size of the data subvolume. You should specify a size other than 0 only if the filesystem should occupy less space than the size of the data subvolume.
 - **Log Size:** Enter the size of the log section of the filesystem as a number of 512-byte blocks. You should specify a size other than 0 only if the log should occupy less space than the size of the log subvolume. This option only appears if the filesystem has a log subvolume.
3. Click **OK**.

Mount a Filesystem Locally

This task lets you mount a filesystem only on the node to which the GUI is connected (the local node).

To mount a filesystem locally, do the following:

1. **Filesystem to Mount:** Select the filesystem you wish to mount. The list of available filesystems is determined by looking for block devices containing XFS superblocks.
2. **Mount Point:** Specify the directory on which the selected filesystem will be mounted.
3. (Optional) **Mount Options:** Specify the options that should be passed to the `mount(1M)` command. For more information about available options, see the `fstab(4)` man page.
4. By default, the filesystem will remount every time the system starts. However, if you uncheck the box, the mount will take place only when you explicitly use this task.
5. Click **OK**.

For more information, see the `mount(1M)` man page.

Unmount a Locally Mounted Filesystem

To unmount a filesystem from the local node, do the following:

1. **Filesystem to Unmount:** Choose the filesystem to be unmounted.
2. **Remove Mount Information:** Click the check box to remove the mount point from the `/etc/fstab` file, which will ensure that the filesystem will remain unmounted after the next reboot. This item is available only if the mount point is currently saved in `/etc/fstab`.
3. Click **OK**.

Remove Filesystem Mount Information

This task lets you delete a local filesystem's mount information in `/etc/fstab`.

Note: The filesystem will still be present on the volume.

Do the following:

1. **Filesystem Name:** Select the filesystem for which you want to remove mount information. The list of available filesystems is determined by looking for block devices containing XFS superblocks.
2. Click **OK**.

Privileges Tasks

The privileges tasks let you grant specific users the ability to perform specific tasks, and to revoke those privileges.

Note: You cannot grant or revoke tasks for users with a user ID of 0.

Grant Task Access to a User or Users

You can grant access to a specific task to one or more users at a time.

Do the following:

1. Select the user or users for whom you want to grant access. You can use the following methods to select users:
 - Click to select one user at a time
 - Shift+click to select a block of users
 - Ctrl+click to toggle the selection of any one user, which allows you to select multiple users that are not contiguous
 - Click **Select All** to select all users

Click **Next** to move to the next page.

2. Select the task or tasks to grant access to, using the above selection methods. Click **Next** to move to the next page.
3. Confirm your choices by clicking **OK**.

Note: If more tasks than you selected are shown, then the selected tasks run the same underlying privileged commands as other tasks, such that access to the tasks you specified cannot be granted without also granting access to these additional tasks.

To see which tasks a specific user can access, select **View: Users**. Select a specific user to see details about the tasks available to that user.

To see which users can access a specific task, select **View: Task Privileges**. Select a specific task to see details about the users who can access it and the privileged commands it requires.

Granting Access to a Few Tasks

Suppose you wanted to grant user `diag` permission to make, mount, and unmount a filesystem. You would do the following:

1. Select `diag` and click **Next** to move to the next page.
2. Select the tasks you want `diag` to be able to execute:
 - a. Ctrl+click **Make Filesystems**
 - b. Ctrl+click **Mount a Filesystem Locally**
 - c. Ctrl+click **UnMount a Locally Mounted Filesystem**

Click **Next** to move to the next page.

3. Confirm your choices by clicking **OK**.

Granting Access to Most Tasks

Suppose you wanted to give user `sys` access to all tasks **except** stealing a foreign disk. The easiest way to do this is to select all of the tasks and then deselect the task (or tasks) you want to restrict. You would do the following:

1. Select `sys` and click **Next** to move to the next page.
2. Select the tasks you want `sys` to be able to execute:
 - a. Click **Select All** to highlight all tasks.
 - b. Deselect the task to which you want to restrict access. `Ctrl+click` **Steal a Foreign Disk**.

Click **Next** to move to the next page.

3. Confirm your choices by clicking **OK**.

Revoke Task Access from a User or Users

You can revoke task access from one or more users at a time.

Do the following:

1. Select the user or users from whom you want to revoke task access. You can use the following methods to select users:
 - Click to select one user at a time
 - `Shift+click` to select a block of users
 - `Ctrl+click` to toggle the selection of any one user, which allows you to select multiple users that are not contiguous
 - Click **Select All** to select all users

Click **Next** to move to the next page.

2. Select the task or tasks to revoke access to, using the above selection methods. Click **Next** to move to the next page.
3. Confirm your choices by clicking **OK**.

Note: If more tasks than you selected are shown, then the selected tasks run the same underlying privileged commands as other tasks, such that access to the tasks you specified cannot be revoked without also revoking access to these additional tasks.

To see which tasks a specific user can access, select **View: Users**. Select a specific user to see details about the tasks available to that user.

To see which users can access a specific task, select **View: Task Privileges**. Select a specific task to see details about the users who can access it.

XVM and XLV Logical Volumes

This chapter includes sections on the following topics:

- “XVM and XLV Logical Volume Creation Comparison”
- “Upgrading from XLV to XVM Logical Volumes” on page 215
- “Converting XLV Mirrored Stripes to XVM Striped Mirrors” on page 216

XVM and XLV Logical Volume Creation Comparison

Table A-1 summarizes the creation of a simple XVM logical volume and a simple XLV logical volume. This example is the same example provided in “Creating a Logical Volume with a Three-Way Stripe” on page 108. Refer to that section for a step-by-step explanation of the XVM logical volume creation.

The commands provided in this example create a logical volume named `stripedvol` in which the data is to be striped across three disks. Although the commands are presented side-by-side in the table, they do not necessarily correspond to the same action.

Table A-1 XVM and XLV Logical Volume Creation

XVM Logical Volume	XLV Logical Volume
<p>Label the disks as XVM physical volumes:</p> <pre>xvm:cluster> label -name disk0 \ dks2d70vol xvm:cluster> label -name disk1 \ dks2d71vol xvm:cluster> label -name disk2 \ dks2d72vol</pre>	<p>Create the logical volume:</p> <pre>xlvmake> vol stripedvol</pre>
	<p>Create the data subvolume:</p> <pre>xlvmake> data</pre>
<p>Create XVM slices out of each disk:</p> <pre>xvm:cluster> slice -all disk*</pre>	<p>Create the plex:</p> <pre>xlvmake> plex</pre>
<p>Create the striped volume element and attach it to the volume <code>stripedvol</code> (creating the data subvolume in the process):</p> <pre>xvm:cluster> stripe -volname \ stripedvol slice/disk0s0 \ slice/disk1s0 slice/disk2s0</pre>	<p>Create the striped volume element to be striped across the three disks:</p> <pre>xlvmake> ve -stripe dks0d2s7 \ dks0d3s7 dks5d4s7</pre>
<p>Exit the XVM Volume Manager:</p> <pre>xvm:cluster> quit</pre>	<p>Exit the XLV Volume Manager. Since the <code>xlvmake -A</code> option was not used, <code>xlvm assemble</code> is automatically called to configure the volume into the running kernel:</p> <pre>xlvmake> end xlvmake> exit</pre>
<p>Execute the <code>mkfs</code> command on the filesystem:</p> <pre># mkfs /dev/cxvm/stripedvol</pre>	<p>Execute the <code>mkfs</code> command on the filesystem:</p> <pre># mkfs /dev/xlv/stripedvol</pre>
<p>Mount the filesystem. For a shared filesystem in a CXFS cluster, you mount the filesystem with the CXFS GUI or the <code>cmgr(1M)</code> command. For a local filesystem, you can use the <code>mount</code> command.</p>	<p>Mount the filesystem:</p> <pre># mount /dev/xlv/stripedvol /mnt</pre>

Upgrading from XLV to XVM Logical Volumes

Use the following procedure to convert an XLV logical volume configuration to an XVM logical volume configuration:

1. Save the old XLV configuration and disk partitions:

```
# xlv_mgr -c "script -write xlvconfig all"
# prtvtoc /dev/rdisk/*vh > diskparts
```

2. Back up existing filesystems.
3. Delete the XLV volumes you are converting.

To delete all volumes, unmount all XLV filesystems then execute the following commands:

```
# xlv_shutdown
# xlv_mgr -x -c "delete all_labels"
```

Do not delete all XLV volumes if you have an XLV root volume, because you cannot use an XVM volume as a root volume.

4. Give the XLV disks to XVM to manage by labeling them as XVM disks:

```
# xvm label -name xvmdisk1 dks0d1vh
```

5. Create slices to match the old partitions used by XLV.

For the XVM `-start` argument, subtract the size of the volume header from the `prtvtoc` partition start block. In this case, if slice 7 starts at block 4096 and is adjacent to the volume header, use `-start 0`.

The `-length` argument is the `prtvtoc` partition size.

```
# xvm slice -start 0 -length 17779016 xvmdisk1
```

6. Create concatenated or striped volumes on top of the slices. Name the volumes to match previous XLV volume names to minimize the need to change the contents of `/etc/fstab`.
7. Mount the filesystems and run `xfstest` on the filesystems. If there is a problem mounting the filesystems or if there is a problem that `xfstest` reveals, check the following:
 - Compare the slices created on each disk with the saved `prtvtoc` information.
 - Compare the organization of the stripes and concatenated volumes with the XLV information.

If you find you need to restore your XLV configuration, you can use the following procedure as long as you have not built a new configuration using the XVM Volume Manager that you cannot replicate in XLV (for example, you cannot stripe concatenated volume elements in XLV).

1. Unmount the filesystems:

```
# umount -a
```

2. Remove the XVM volume header from all the physical volumes and restore the original partitions:

```
# xvm unlabel -force phys/*
```

3. Rebuild the XLV configuration:

```
# xlv_make xlvconfig
```

Converting XLV Mirrored Stripes to XVM Striped Mirrors

The way that mirrors are configured in XLV logical volumes increases the likelihood that two disk failures will cause the entire mirror to fail. This is because an XLV logical volume could include a mirrored stripe, but not a striped mirror. The XVM Volume Manager, on the other hand, does allow you to stripe mirrors.

The following procedure provides a safe way to convert from XLV mirrored stripes to XVM striped mirrors. In this procedure, the logical volume is named `alpha` and the disk names are `dks18*` and `dks5*`.

1. Unmount the filesystem you are converting.
2. Create a script to regenerate the XLV configuration for `alpha` in case you want to recreate it later:

```
# xlv_mgr -c "script -write alpha object alpha"
```

3. Detach plex 0 from the XLV volume:

```
# xlv_mgr -c "detach plex alpha.data.0 alphaplex"
```

4. Delete the object. If other XLV volumes exist on this disk, generate a script to regenerate the volumes as they will also need to be converted:

```
# xlv_mgr -c "delete object alphaplex"
```

5. Call up the XVM Volume Manager and label the plex0 disks:

```
xvm:local> label dks18*
```

6. In this example, the filesystem has a single slice per disk. Create an XVM slice on each of the disks, consisting of the entire disk:

```
xvm:local> slice -all dks18*
```

7. Create a stripe to match the XLV configuration:

```
xvm:local> stripe -volname alpha slice/dks18*
```

8. Leave the XVM Volume Manager and verify that the filesystem mounts correctly:

```
# mount /dev/lxvm/alpha /mountpoint
```

9. The data is now available through the XVM Volume Manager, but is not mirrored. Note the topology of the volume alpha:

```
xvm:local> show -t alpha
vol/alpha
  subvol/alpha/data          106664448 online,open
    stripe/stripe0          106664448 online,tempname,open
      slice/dks18d1s0        17777424 online,open
      slice/dks18d2s0        17777424 online,open
      slice/dks18d3s0        17777424 online,open
      slice/dks18d4s0        17777424 online,open
      slice/dks18d5s0        17777424 online,open
      slice/dks18d6s0        17777424 online,open
```

Insert mirrors above each of the slices in the volume alpha:

```
xvm:local> insert mirror slice/dks18*
```

This yields the following configuration:

```
xvm:local> show -t alpha
vol/alpha
  subvol/alpha/data          106664448 online,open
    stripe/stripe0          106664448 online,tempname,open
      mirror/mirror0        17777424 online,tempname
        slice/dks18d1s0      17777424 online,open
      mirror/mirror1        17777424 online,tempname
        slice/dks18d2s0      17777424 online,open
      mirror/mirror2        17777424 online,tempname
        slice/dks18d3s0      17777424 online,open
      mirror/mirror3        17777424 online,tempname
        slice/dks18d4s0      17777424 online,open
      mirror/mirror4        17777424 online,tempname
        slice/dks18d5s0      17777424 online,open
      mirror/mirror5        17777424 online,tempname
        slice/dks18d6s0      17777424 online,open
```

10. Convert the other half of the XLV plex to an XVM volume and attach to the mirrors:

```
# xlv_mgr -c "delete object alpha"

xvm:local> label dks5*
xvm:local> slice -all dks5*
xvm:local> attach slice/dks5d1s0 mirror0
xvm:local> attach slice/dks5d2s0 mirror1
xvm:local> attach slice/dks5d3s0 mirror2
xvm:local> attach slice/dks5d4s0 mirror3
xvm:local> attach slice/dks5d5s0 mirror4
xvm:local> attach slice/dks5d6s0 mirror5
```

Alternately, you could use the following syntax for the attach commands:

```
xvm:local> attach slice/dks5d1s0 stripe0/0
xvm:local> attach slice/dks5d2s0 stripe0/1
xvm:local> attach slice/dks5d3s0 stripe0/2
xvm:local> attach slice/dks5d4s0 stripe0/3
xvm:local> attach slice/dks5d5s0 stripe0/4
xvm:local> attach slice/dks5d6s0 stripe0/5
```

Show the topology of volume alpha:

```
xvm:local> show -t alpha
vol/alpha          0 online
  subvol/alpha/data 106664448 online,open
    stripe/stripe0 106664448 online,tempname,open
      mirror/mirror0 17777424 online,tempname,reviving:28%,open
        slice/dks18d1s0 17777424 online,open
        slice/dks5d1s0 17777424 online,open
      mirror/mirror1 17777424 online,tempname,reviving:queued,open
        slice/dks18d2s0 17777424 online,open
        slice/dks5d2s0 17777424 online,open
      mirror/mirror2 17777424 online,tempname,reviving:queued,open
        slice/dks18d3s0 17777424 online,open
        slice/dks5d3s0 17777424 online,open
      mirror/mirror3 17777424 online,tempname,reviving:queued,open
        slice/dks18d4s0 17777424 online,open
        slice/dks5d4s0 17777424 online,open
      mirror/mirror4 17777424 online,tempname,reviving:queued,open
        slice/dks18d5s0 17777424 online,open
        slice/dks5d5s0 17777424 online,open
      mirror/mirror5 17777424 online,tempname,reviving:queued,open
        slice/dks18d6s0 17777424 online,open
        slice/dks5d6s0 17777424 online,open
```

11. Update the `/etc/fstab` file entry from `/dev/xlv/alpha` to `/dev/lxvm/alpha`.

Index

A

apivers subsystem parameter, 168
attach command, 42, 49, 80

C

change command
 stat option, 157
change command, 40, 44, 73, 79
child volume element
 definition, 16
clean
 state, 51
clear
 option of mirror command, 48
cluster domain, 34, 55, 68
cluster initialized subsystem parameter, 168
clustered parameter, 168
collapse command, 52, 83
concat
 creation, 45
 definition, 23
 statistics, 160
concat command, 45, 75
config gen subsystem parameter, 168
CXFS filesystems, 31

D

data subvolume
 definition, 22
delete command, 53, 86
detach command, 44, 49, 81
device hot plug, 4
direct connection, 172
disabled state, 51
domain
 cluster, 31, 34, 68
 local, 31, 34, 68
drag-and-drop, with XVM Manager GUI, 184
dump command, 39, 52, 73, 86

F

failover, 91
 version 1, 93
 version 2, 94
failover2.conf file, 95
failover.conf file, 93
foconfig command, 98
-force option
 delete command, 53
 detach, 44
 general, 65
 unlabel command, 40
foreign disk, 34, 61, 70
foswitch command, 99

- G**
- GPT label, 144
 - growing logical volumes, 135
- H**
- help command, 57
 - hot plug, 4
- I**
- "Illegal request" message, 94
 - incomplete state, 51
 - inconsistent state, 50
 - insert command, 52, 82
 - installing XVM Volume Manager, 32
- K**
- keywords
 - xvm command, 58
- L**
- label command, 37, 68, 113
 - Linux
 - configuring XVM volumes, 107
 - device directories, 64
 - local connection, 172
 - local domain, 34, 68
 - log messages, XVM Manager GUI, 185
 - log subvolume
 - definition, 22
 - logical volume
 - creation, 134
 - creation example, 112
 - destruction, 53
 - display, 50
 - growing, 135
 - management, 49
 - mirroring, 137
 - offline state, 52, 53
 - online modification, 133
 - online state, 52
 - reorganization, 49
 - writing data, 29
- M**
- Making a GPT label, 144
 - Making an XVM volume using a GPT label, 143
 - mediaerr state, 50
 - miniroot, 126
 - mirror
 - attaching, 43
 - creation, 45
 - definition, 27
 - detach, 43, 81
 - mirroring stripes, 142
 - no synchronize at creation, 48
 - primary leg, 47
 - read policies, 46
 - removal, 141
 - revive, 43, 46, 76, 81, 164, 165
 - revive resources, 165
 - statistics, 161
 - mirror command
 - clean option, 48
 - norevive option, 48
 - mirror command, 45, 75

N

- name option, label command, 68
- naming
 - volume elements, 41
 - volumes, 41
- norevive
 - option of mirror command, 48

O

- object names
 - regular expressions, 63
- object type, 61
- offline state, 50, 52
 - logical volume, 53
- online state, 50, 52
- open state, 51
- option disk, 5

P

- partition layout, 4-8
- path specification, 61
- Performance Co-Pilot, use with XVM GUI, 183
- physical volume
 - adding to a running system, 39
 - creation, 37
 - definition, 34
 - destruction, 40
 - display, 39
 - management, 38
 - replacing, 39
 - statistics, 158
- physvol object type
 - definition, 61
 - pathname, 61

- physvol. *See* physical volume
- piece
 - definition, 16
- pieceoffline state, 51
- primary leg, mirror, 47
- privileged subsystem parameter, 168
- probe command, 73
- probing a disk. *See* probe command
- proxy connection, 172

R

- regular expressions, object names, 63
- remake command, 49, 81
- remote shell connection, 172
- reprobe command, 87
- reviving state, 51
- root privileges, 56
- round-robin mirror read policy, 46

S

- safe commands, 65
- safe option
 - detach, 44
 - general, 43, 65
- SAN disk paths, 61
- sequential mirror read policy, 46
- set command, 68
- show command, 50, 69, 85, 113, 168
- slice
 - creation, 44
 - definition, 23
 - statistics, 162
- slice command, 44, 74, 113

- state
 - clean, 51
 - disabled, 51
 - incomplete, 51
 - inconsistent, 50
 - mediaerr, 50
 - offline, 50
 - online, 50
 - open, 51
 - pieceoffline, 51
 - reviving, 51
 - tempname, 51
 - volume element, 50
- statistics, 44, 157
 - concat, 160
 - mirror, 161
 - physical volume, 158
 - slice, 162
 - stripe, 158
 - subvolume, 158
- steal command, 70
- stripe
 - creation, 45
 - definition, 25
 - size, 77
 - statistics, 158
 - unit, 77
- stripe command, 45, 77, 114
- subsystem option, show command, 168
- subvolume
 - creation, 48
 - data, 22, 49
 - definition, 22
 - limitations, 20
 - log, 22, 49
 - names, 59
 - statistics, 158
 - type, 49
 - user-defined, 23, 49
- subvolume command, 48, 78, 79

- system disk, 6

T

- tempname state, 51
- topology. *See* XVM topology
- tunable parameters, 165
- tuning XVM, 155

U

- unlabel command, 40, 73
- unlabeled disk, 61
 - definition, 33
- upgrading from XLV to XVM, 215
- user-defined subvolume, 23
- usr filesystem, 5

V

- volume
 - creation, 48
 - definition, 20
- volume command, 48
- volume element
 - attaching, 42
 - automatic creation, 41
 - creation, 40
 - definition, 16, 34
 - detaching, 43
 - display, 50
 - empty volume element, 44
 - growing online, 52
 - limitations, 16
 - naming, 41
 - piece syntax, 61

volume header, 20
volume name, temporary, 41

W

wildcards, regular expressions, 63

X

XLV

- logical volume, 2
- upgrading to XVM, 215

xvm command

- abbreviations, 58
- help, 57
- keywords, 58
- output, 63
- redirection, 63
- syntax, 58

XVM GUI

See XVM Manager Graphical User Interface (GUI)

XVM logical volume

- definition, 20
- limitations, 16, 20
- logical levels, 16
- regenerating, 52
- saving, 52

XVM Manager Graphical User Interface (GUI), 169

- command buttons, 182
- drag-and-drop, 184
- icons, 176
- installing, 170
- log messages, 185
- starting, 171
- states, 177
- Web-based version, 170
- window, 172

XVM objects

- names, 59
- regular expressions, 63
- types, 60

XVM system disk, 6

XVM topology

- creation, 41
- display, 41

XVM volume

- limitations, 42

xvm_max_revive_rsc parameter, 165

xvm_mr_daemon_max variable, 155

