XVM Volume Manager Administrator Guide

007–4003–032

COPYRIGHT

LIMITED RIGHTS LEGEND

TRADEMARKS AND ATTRIBUTIONS

What’s New in This Guide

This revision contains the following changes:

- The new dynamic system tunable parameter pm_display_io_activity, which specifies whether or not recent I/O activity (the amount of data transferred and the number of I/O operations) is reported for a 1–second period in the output of the xvm show command on Linux systems. See "pm_display_io_activity" on page 239.
- Due to revised rules for udev in recent versions of Linux, device names of XVM volumes have changed. Previously, xvm devices had names such as /dev/lxvm/dmfusr1 or /dev/cxvm/dmfusr2. In this release, those paths still exist but are symbolic links to the actual device name (such as /dev/xvm-5). The output of other commands that show device names, such as mount(8) and df(1), may differ from previous releases. See "Understand the Use of Symbolic Links and Device Names in Output" on page 61.
- Additional clarifications and corrections

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 ISSP 2.4 release
027	November 2011 Supports the XVM product in the ISSP 2.5 release
028	May 2012 Supports the XVM product in the ISSP 2.6 release
029	April 2013 Supports the XVM product in the ISSP 3.0 release
030	November 2013 Supports the XVM product in the ISSP 3.1 release
031	May 2014 Supports the XVM product in the ISSP 3.2 release
032	October 2014 Supports the XVM product in the ISSP 3.3 release

Contents

About This Guide

This guide describes the configuration and administration of logical volumes using the XVM volume manager.

Related Publications

For information about this release, see the following release notes:

- SGI InfiniteStorage™ Software Platform (ISSP): `README.txt`
- CXFS™ general release note: `README_CXFS_GENERAL.txt`

The following documents contain additional information:

- *CXFS 7 Administrator Guide for SGI InfiniteStorage*
- *CXFS 7 Client-Only Guide for SGI InfiniteStorage*
- *Performance Co-Pilot Programmer's Guide*
- *Performance Co-Pilot for Linux User's and Administrator's Guide*

Obtaining Publications

You can obtain SGI documentation as follows:

- See the SGI Technical Publications Library at http://docs.sgi.com. Various formats are available. This library contains the most recent and most comprehensive set of online books, man pages, and other information.
- The `/docs` directory on the ISSP DVD or in the Supportfolio™ download directory contains the following:
  - The ISSP release note: `/docs/README.txt`
  - A complete list of the packages and their location on the media: `/docs/RPMS.txt`
  - The packages and their respective licenses: `/docs/PACKAGE_LICENSES.txt`

- The release notes and manuals are provided in the `noarch/sgi-isspdocs` RPM and will be installed on the system into the following location:

  `/usr/share/doc/packages/sgi-issp-`*ISSPVERSION*`/`*TITLE*

- You can view man pages by typing `man` *title* at a command line.

Note: The external websites referred to in this guide were correct at the time of publication, but are subject to change.

Conventions

This guide uses the following terminology abbreviations:

- *Linux* refers to the supported distribution of Linux defined in the release note
- *Windows* refers to the supported distribution of Windows defined in the CXFS general release note

The following conventions are used throughout this document:

Convention	Meaning
`command`	This fixed-space font denotes literal items such as commands, files, routines, path names, signals, messages, and programming language structures.
variable	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.)
[ ]	Brackets enclose optional portions of a command or directive line.
GUI element	This bold font denotes the names of graphical user interface (GUI) elements, such as windows, screens, dialog boxes, menus, toolbars, icons, buttons, boxes, and fields.

`<TAB>` Represents pressing the specified key in an interactive session

`server-admin#` In an example, this prompt indicates that the command is executed on a server-capable administration node

`client#` In an example, this prompt indicates that the command is executed on a client-only node

`MDS#` In an example, this prompt indicates that the command is executed on an active CXFS metadata server

`#` In an example, this prompt indicates that the command is executed on an any node

specificnode# In an example, this prompt indicates that the command is executed on a node named *specificnode* or of node type *specificnode*

Reader Comments

If you have comments about the technical accuracy, content, or organization of this publication, contact SGI. Be sure to include the title and document number of the publication with your comments. (Online, the document number is located in the front matter of the publication. In printed publications, the document number is located at the bottom of each page.)

You can contact SGI in either 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:

  http://www.sgi.com/support/supportcenters.html

SGI values your comments and will respond to them promptly.

Chapter 1

Introduction to the XVM Volume Manager

This chapter discusses the following:

- "XVM Features" on page 1
- "Overview of XVM Volume Composition" on page 5
- "Writing Data" on page 9
- "Volumes and Failover" on page 11
- "Installing XVM" on page 11
- "Path Manager" on page 12

XVM Features

The XVM volume manager combines underlying physical disk storage into a single logical unit, known as a *logical volume* (abbreviated to *volume*). Volumes behave like standard disk partitions and you can use them as arguments anywhere that you can specify a partition.

A volume allows a filesystem or raw device to consist of multiple physical disks. Using volumes can also increase disk I/O performance because a volume can be distributed (or *striped*) across multiple disks. Volumes can also be used to mirror data on different disks.

XVM provides the following features:

- "Management Tools" on page 2
- "Support for a Cluster Environment" on page 2
- "Flexible Volume Layering and Configuration" on page 3
- "Large Number of Slices" on page 3
- "Large Number of Volumes" on page 3
- "Mirror Performance Options" on page 3
- "Built-in Statistics Support" on page 3

- "Device Hot Plug" on page 4
- "Insertion and Removal" on page 4
- "Self-Identifying Volumes" on page 4
- "Multiple Storage Types" on page 4
- "Multiple Address Spaces" on page 4
- "Path Failover" on page 4
- "Online Configuration Changes" on page 5

Management Tools

The `xvm`(8) command-line interface and the **XVM Manager** graphical user interface (GUI) provides access to the tasks that help you set up and administer your volumes. The GUI provides icons representing status and structure. You can access the GUI via the `xvmgr`(1) command or via the web.

Support for a Cluster Environment

XVM supports a cluster environment in association with the CXFS clustered filesystem product, 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 (the *cluster domain*) or to individual nodes within the cluster (the *local domain*).

Note: If the appropriate services are not started, XVM cannot be set to the cluster domain. See "CXFS Service Requirements for Cluster Domain" on page 26.

See:

- "Local Domain and Cluster Domain" on page 26
- *CXFS 7 Administrator Guide for SGI InfiniteStorage*
- *CXFS 7 Client-Only Guide for SGI InfiniteStorage*

Flexible Volume Layering and Configuration

The elements that make up a volume can be layered in any configuration. For example, you can mirror disks at any level of the volume configuration.

Large Number of Slices

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

Large Number of Volumes

XVM supports thousands of volumes on a single disk and allows for the expansion of the label file as needed. There are no restrictions on *volume width*, which is the number of volume elements that make up the widest layer of the XVM topology tree for a volume.

Mirror Performance Options

XVM lets you specify the read policy for a mirror, 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 mirror leg is to be preferred for reading.

XVM also lets you specify that a mirror does not need to be synchronized at creation or that a particular mirror (such as a mirror of a scratch filesystem) will never need to be synchronized.

Built-in Statistics Support

XVM tracks statistics at every level of the topology 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 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 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.

Self-Identifying Volumes

Persistent configuration and attribute information for a volume is distributed among all disks that are part of the 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

Volumes support aggregate storage through concatenation and striping. Volumes also support redundant storage through mirroring.

Multiple Address Spaces

A volume can support multiple mutually exclusive address spaces in the form of *subvolumes*. Each subvolume within a volume has a different usage defined by the application accessing the data. XVM supports multiple subvolume types. See "Subvolume" on page 15.

Path Failover

When a host performs an I/O operation and it fails due to connection problems, XVM will automatically try all paths that the host has to the disk by switching (*failing over*) from an inaccessible path to the alternate paths in turn until a usable path is found, or until all paths are found to fail.

XVM supports both asymmetric logical unit access (ALUA) RAID, which simplifies the setup of automatic path failover, and non-ALUA RAID.

Online Configuration Changes

XVM lets you 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 mirror leg.

Overview of XVM Volume Composition

XVM volumes are composed of a hierarchy of logical storage objects, each of which is known as a *volume element* (referred to as a `ve` in some commands):

- Volumes are composed of subvolumes
- Subvolumes are composed of stripes, mirrors, concats (for *concatenated volume elements*), and slices, combined in whatever hierarchy suits your system needs
- Stripes, mirrors, and concats are ultimately made up of slices
- Slices define an area of physical storage

The concat, stripe, and mirror volume elements can be arranged and stacked arbitrarily. There is a limit of ten levels from the volume through the slice, inclusive. The hierarchy of elements that compose a volume is known as the *topology tree*.

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

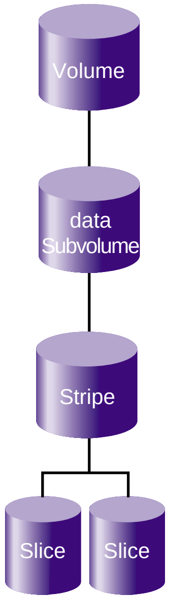


Figure 1-1 Basic Striped Volume

Figure 1-2 shows an XVM volume with two subvolumes (`data` and `log`) and a mirrored stripe in the `data` subvolume.

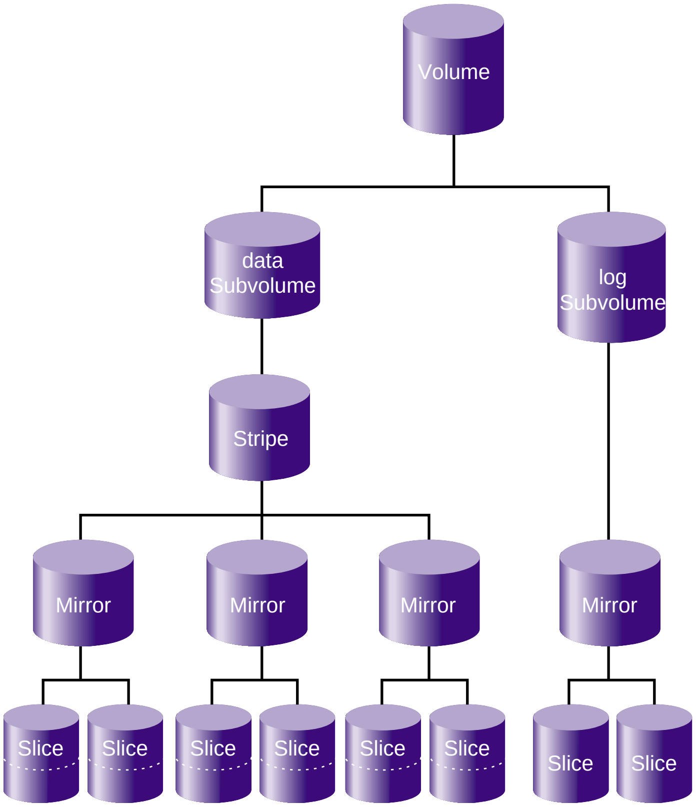


Figure 1-2 Volume with Mirrored Stripe and Two Subvolumes

Figure 1-3 shows the example illustrated in Figure 1-2 after the insertion of a concat. In Figure 1-3, 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.Two slices are on each physical disk in the `data` subvolume.

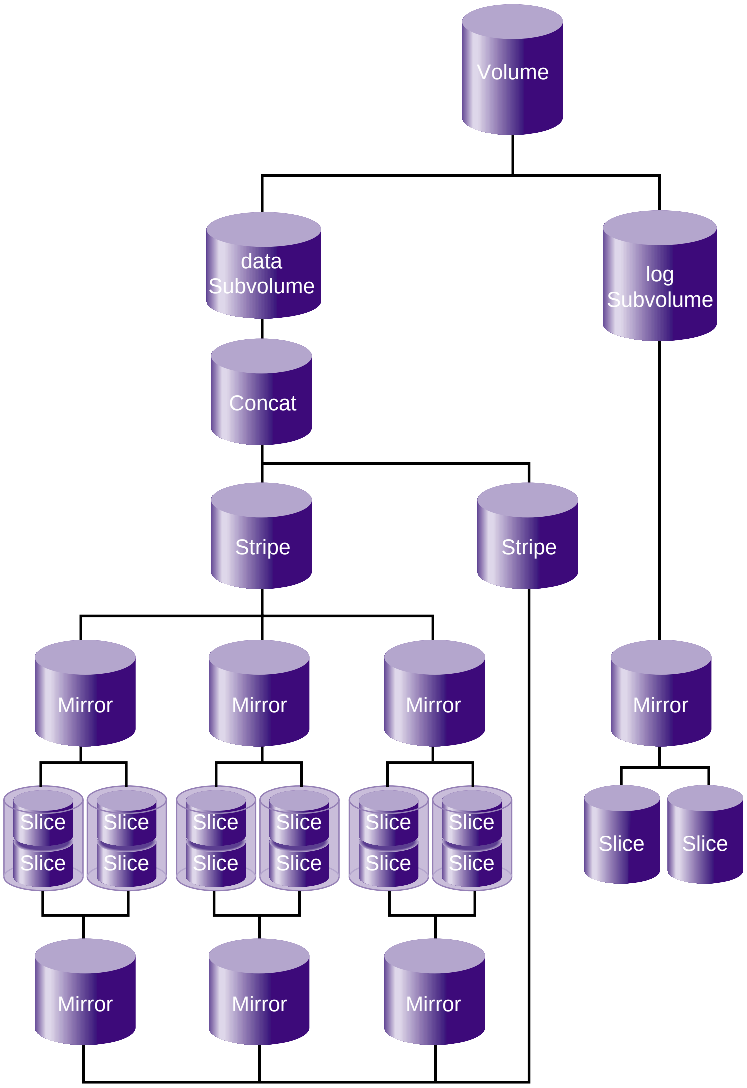


Figure 1-3 Volume after Insertion of Concat

For more details, see "Composition of XVM Volumes" on page 14.

Writing Data

A volume can include slices from several physical disk drives. If the volume is not striped, data is written to the first volume element until that element is full. Figure 1-4 shows the order in which data is written to a concat consisting of three slices. In this figure, each wedge represents a unit of data that is written to disk. To some degree, the data is written in parallel to the various slices.

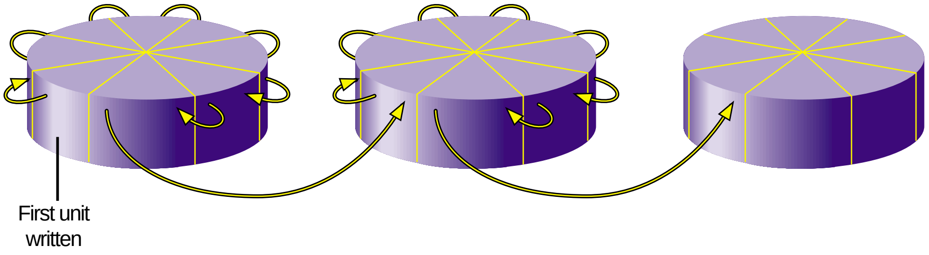


Figure 1-4 Writing Data to a Non-Striped Volume

If the volume is striped, an amount of data called the *stripe unit* is written to each underlying volume element in a round-robin fashion. Figure 1-5 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 each stripe component, with some degree of parallelism.

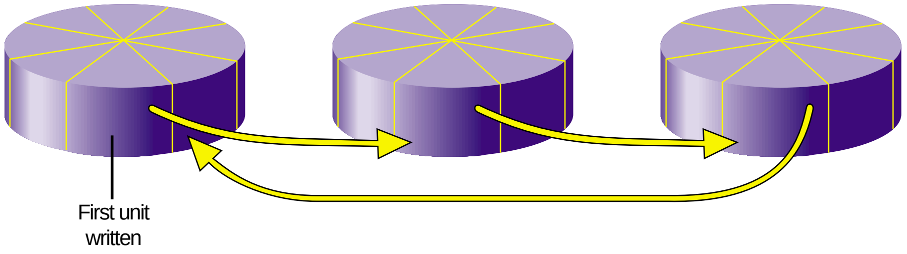


Figure 1-5 Writing Data to a Striped Volume

As an example of configuring stripes to improve performance, consider a situation where your typical I/O activity is 2 MB in size and you have four disks. If you simply concatenate the disks into one volume, the I/O will all go to one disk until it is full, as shown in Figure 1-6.

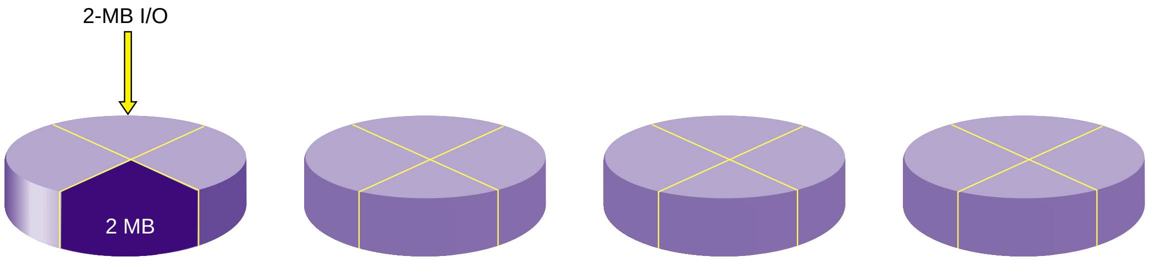


Figure 1-6 Four Disks, No Striping

However, if you stripe the four disks using a stripe unit of 512 KB, then a 2–MB I/O activity will use all four disks in parallel, each working with 512 KB of data, as shown in Figure 1-7.

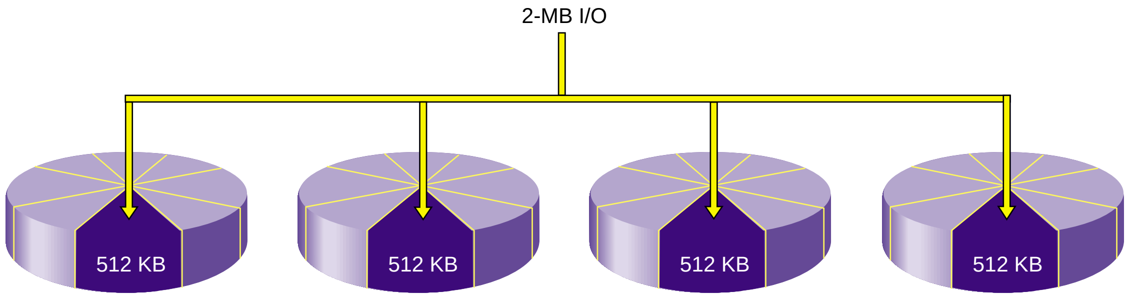


Figure 1-7 Four Striped Disks

For more information, see "Use an Appropriate Stripe Unit Size and Alignment" on page 52.

Volumes and Failover

For each I/O operation, XVM selects the I/O path that is the least heavily loaded at the moment and is connected to the proper RAID controller. Additional tuning may be done in the `/etc/failover2.conf` file:

- For ALUA RAID, the preferred RAID controller for each LUN is set in the RAID itself, and XVM reads that setting.
- For non-ALUA RAID, you must define the preferred RAID controller as well as the grouping of paths to each specific controller (to prevent unnecessary switching between RAID controllers, which can degrade performance considerably) by using the `preferred` and `affinity` keywords.

In either case, the likelyhood of an I/O operation using a particular path may be adjusted by using the `priority` keyword.

For details, see Chapter 6, "XVM Path Failover".

Installing XVM

XVM is released in the SGI InfiniteStorage Software Platform (ISSP) media kit and includes the following packages for SUSE® Linux Enterprise Server (SLES®) Red Hat® Enterprise Linux (RHEL):

- Kernel module:

  `kmod-xvm` *(RHEL)*
  `sgi-xvm-default` *(SLES)*

- Commands:

  ```
  sgi-pm-commands
  sgi-xvm-commands
  ```

- **XVM Manager** GUI:

  ```
  sgi-sysadm_xvm-client
  sgi-sysadm_xvm-server
  sgi-sysadm_xvm-web
  ```

For installation information, see the ISSP release note.

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 SLES servers/clients and RHEL 5 or later clients. It includes the `pathmgr` kernel module.

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
- Path failover functionality
- Dynamic load balancing across paths to a LUN

Note: The path manager feature does not require any additional actions on the part of a system administrator.

Chapter 2

XVM Concepts

Before configuring and administering XVM volumes, you should be familiar with the concepts that underlie the administration commands. This chapter describes the tasks that XVM performs on physical and logical disk resources. More complete descriptions of the `xvm` command-line interface (CLI) commands are provided in Chapter 5, "`xvm` Administration Commands". Also see Chapter 10, "XVM Manager GUI" on page 175.

This chapter discusses the following:

- "XVM Objects" on page 13
- "Composition of XVM Volumes" on page 14
- "Local Domain and Cluster Domain" on page 26
- "Physical Disk Administration" on page 30
- "Creating Logical Resources" on page 36
- "Managing Logical Resources" on page 45
- "Destroying Logical Resources" on page 48
- "Gathering Statistics for Physical Disks and Logical Resources" on page 49

XVM Objects

The following are XVM objects:

Unlabeled disk

Either of the following:

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

Note: 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.

Physvol

A disk (physical volume) that has been labeled for use by XVM and has been probed by the system. .

Foreign disk

A disk that has an XVM physvol label but cannot be administered by the current node because it is owned by a different node or a different cluster.

Volume element

A building block of an XVM topology. The following are all volume elements:

- Volume
- Subvolume
- Concat (for *concatenated volume element*)
- Mirror
- Stripe
- Slice

Note: In some command output, *volume element* is abbreviated to `ve`.

Composition of XVM Volumes

This section discusses the following:

- "Volume" on page 15
- "Subvolume" on page 15
- "Slice" on page 18
- "Concat" on page 18

- "Stripe" on page 19
- "Mirror" on page 20
- "Children" on page 24

Volume

A *volume* is the topmost XVM volume element. It is a collection of *subvolumes* (described in "Subvolume" on page 15) that 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.

Subvolume

A *subvolume* is the entry point for I/O. Each subvolume is a distinct address space. Subvolumes can be of the following types:

Type	**Description**
`data`	System-defined type that contains most data, including user files. The name of a subvolume of type `data` is always `data`. There can be only one `data` subvolume within a given volume.
`log`	System-defined type that contains an external log of XFS filesystem metadata. This log can be used to expedite system recovery after a crash. The name of a subvolume of type `log` is always `log`. The `log` subvolume is optional for a volume; if one is not present, the filesystem log is kept within the `data` subvolume. There can be only one `log` subvolume within a given volume.
`rt`	System-defined type that contains data appropriate for a real-time filesystem. The name of a subvolume of type `rt` is always `rt`. An `rt` subvolume is generally used for data applications such as video, where guaranteed response time is more important than data integrity. An `rt` subvolume is optional. There can be only one `rt` subvolume within a given volume. Volume elements that are part of a real-time subvolume

should not be on the same disk as volume elements used for `data` or `log` subvolumes.

Note: CXFS does not support real-time filesystems, therefore the `rt` subvolume effectively only applies to XFS filesystems in the XVM local domain.

`16-255` User-defined type that contains other user-defined information as configured by the site (the integers `0–15` are reserved for system-defined types).

You can specify any name for a subvolume with a user-defined type, other than the reserved words `data`, `log`, and `rt`.

You can have multiple user-defined subvolumes under a volume if each has a unique type number.

Subvolume types enforce separation of information. For example, user data in a `data` subvolume could not overwrite filesystem log data in a `log` subvolume or data in the user-defined type `16` and type `17` subvolumes.

Figure 2-1 shows a volume with system-defined subvolume types.

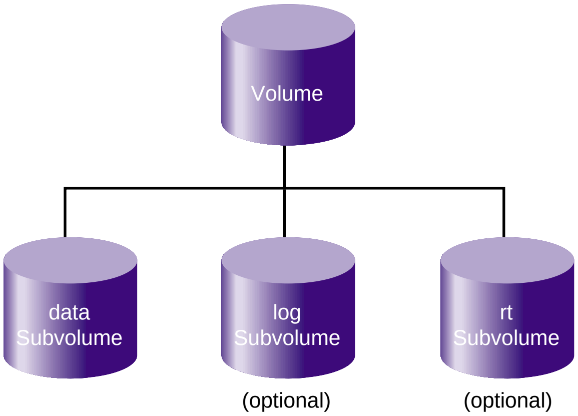


Figure 2-1 System-Defined Subvolume Types

Figure 2-2 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 XVM object names, see "Specifying Objects by Name" on page 72.

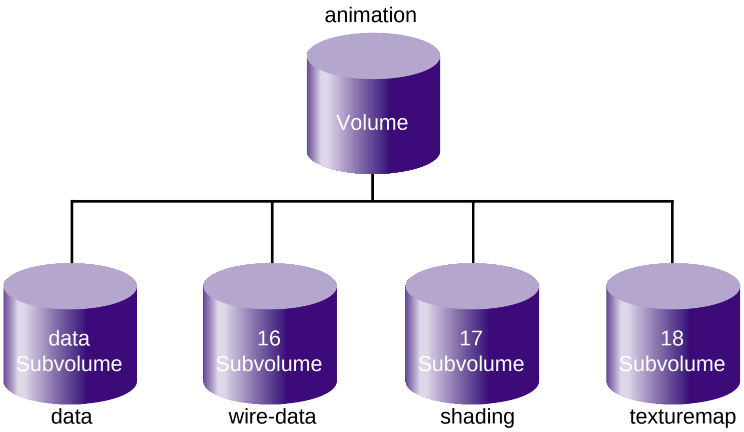


Figure 2-2 User-Defined and System-Defined Subvolume Types

Subvolumes let you meet goals for performance and reliability. For example, performance can be improved by putting subvolumes on different disk drives.

Each subvolume can be organized independently. For example, you could mirror the `log` subvolume for fault tolerance and stripe the `rt` subvolume across a large number of disks to give maximum throughput for video playback.

There can be only one volume element beneath a subvolume in an XVM topology. See "Children" on page 24.

Slice

A *slice* is the lowest level in the topology tree for a volume. A slice defines physical storage; it maps address space of a physical disk onto a volume element.

Concat

A *concat* 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, as shown in Figure 2-3.

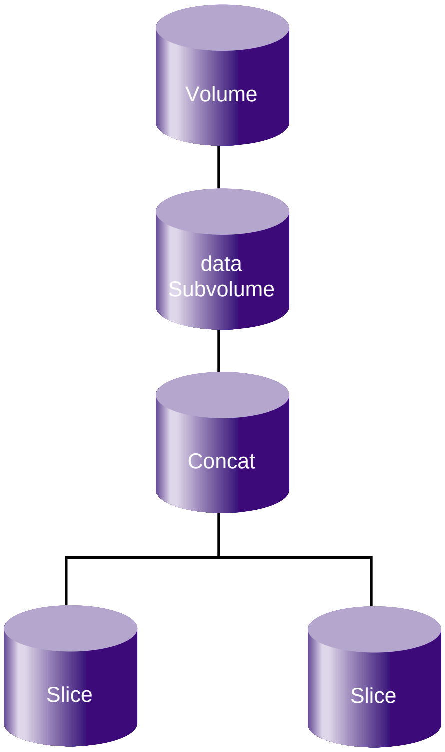


Figure 2-3 Concat Composed of Two Slices

Stripe

A *stripe* 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. You can use striping to distribute data among multiple disks. This provides a performance advantage by allowing parallel I/O activity.

Figure 2-4 shows the concept of a three-way stripe.

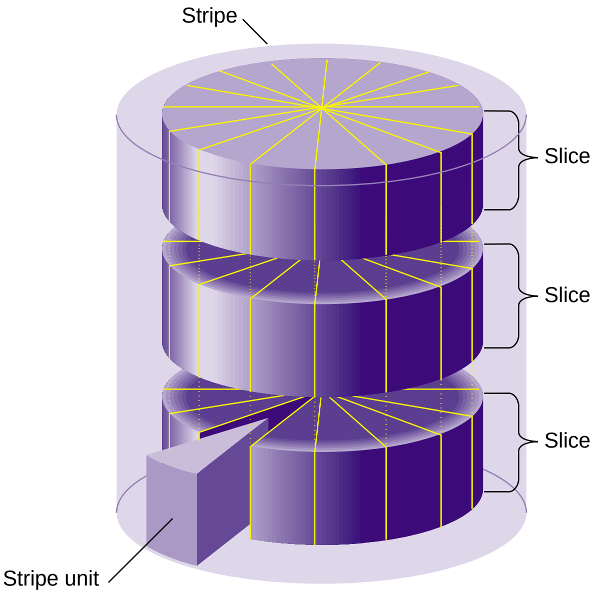


Figure 2-4 Three-Way Stripe

Mirror

A *mirror* 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.

The following figures show various types of mirror composition.

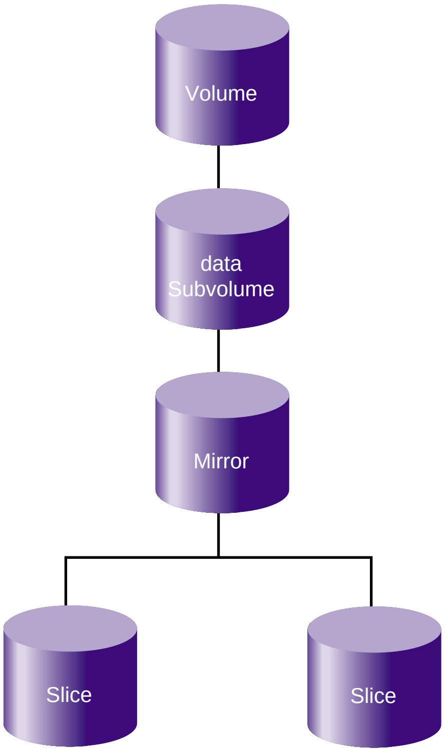


Figure 2-5 Mirror Composed of Two Slices

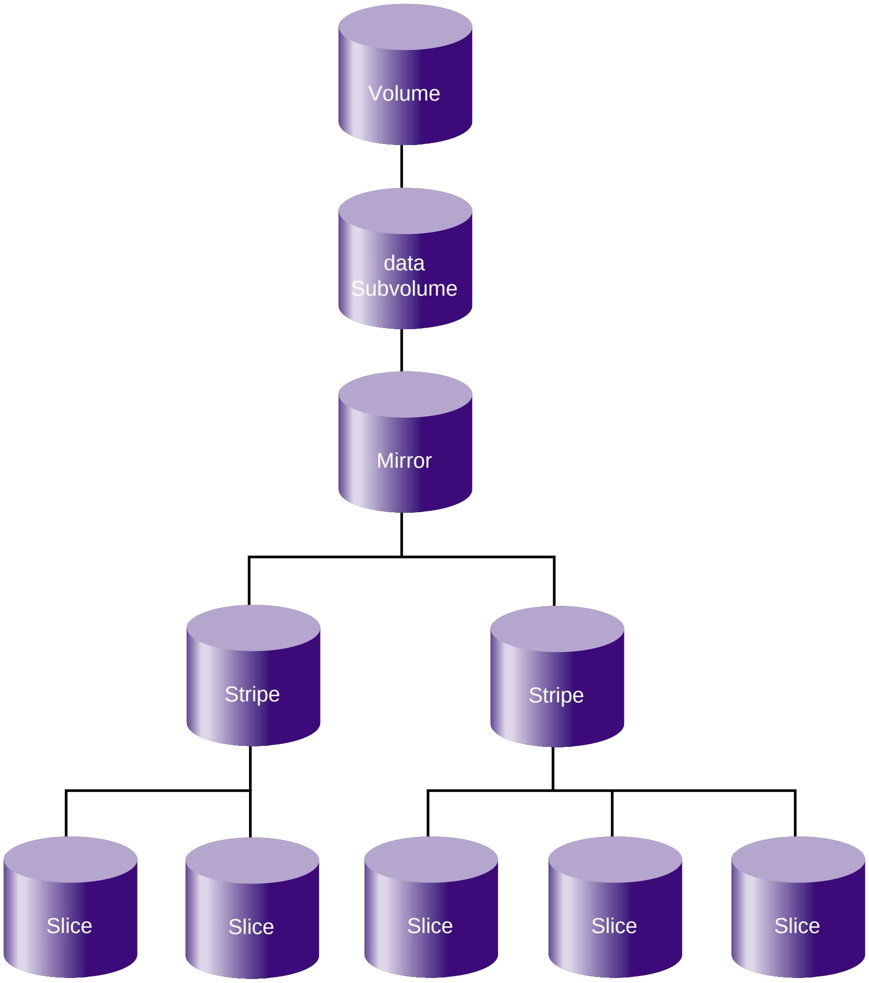


Figure 2-6 Mirror Composed of Two Stripes

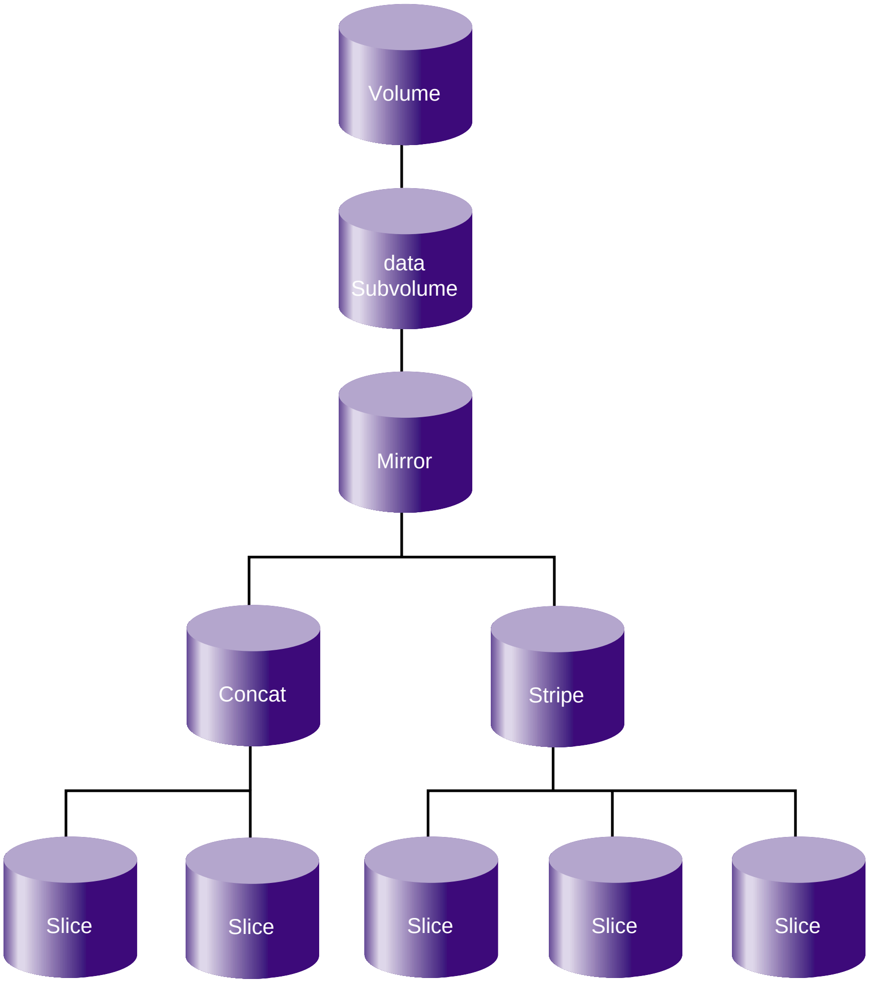


Figure 2-7 Mirror Composed of a Concat and a Stripe

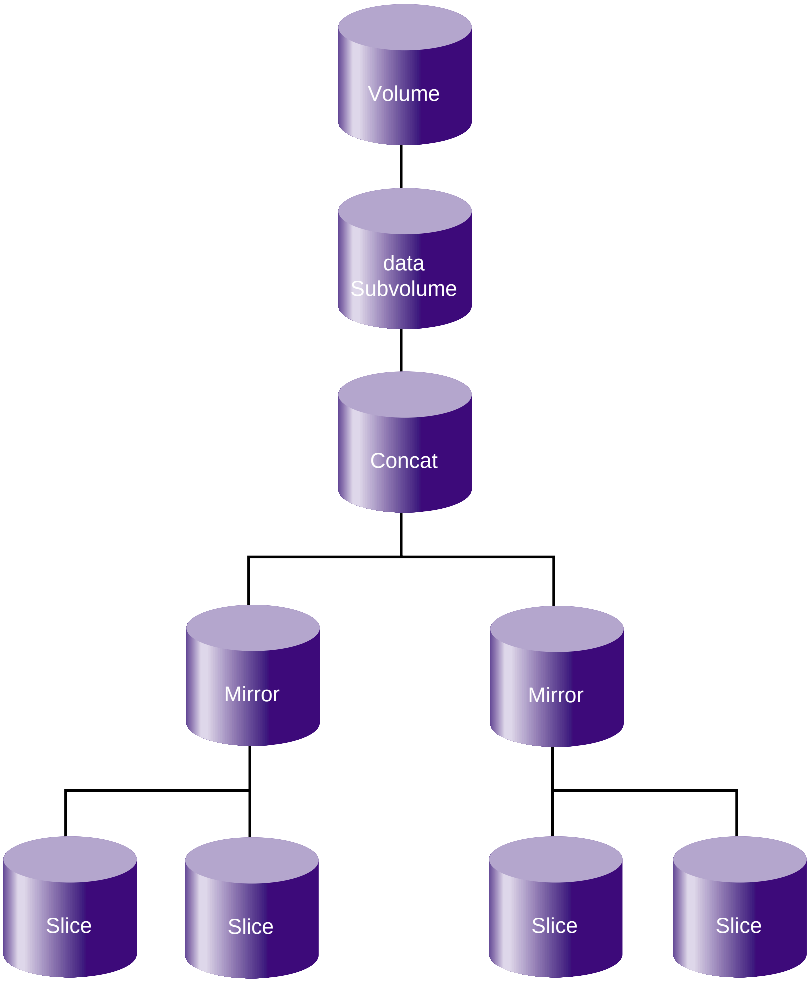


Figure 2-8 Concat Composed of Two Mirrors

Children

A volume element beneath another volume element in the hierarchy is known as a *child* 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 2-9 displays the maximum number of children for each volume element in the XVM topology tree.

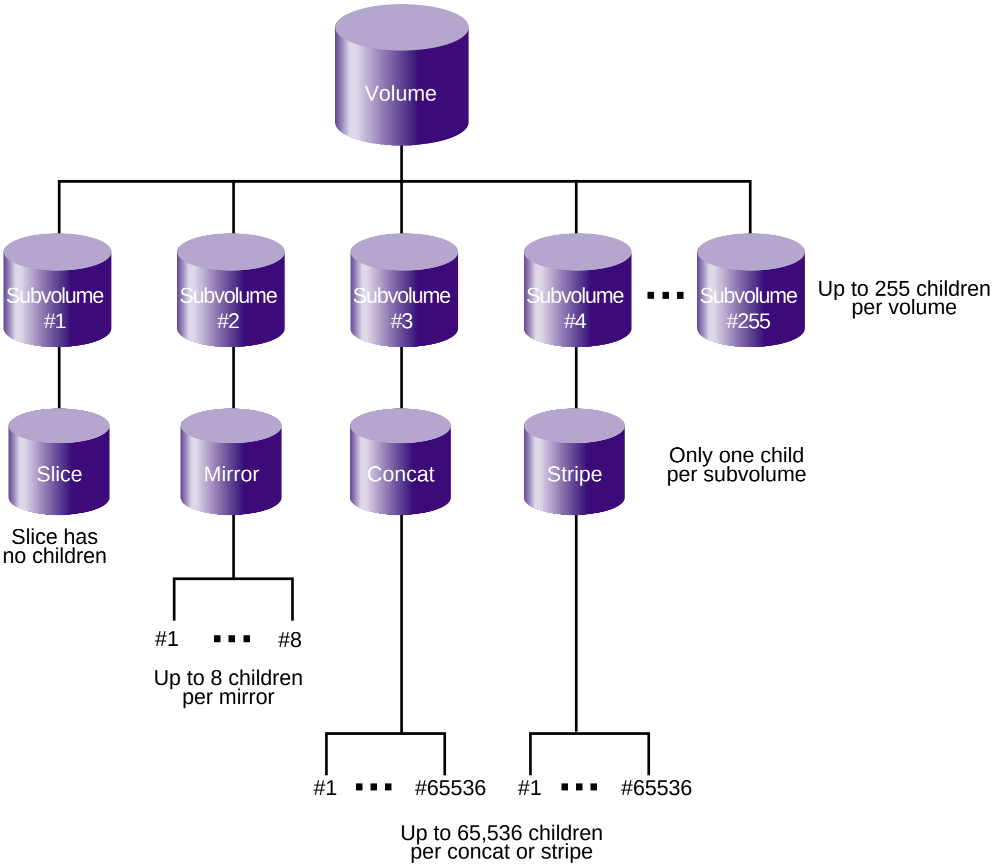


Figure 2-9 Children Maximums

Local Domain and Cluster Domain

This section discusses the following:

- "Overview of Domains" on page 26
- "CXFS Service Requirements for Cluster Domain" on page 26
- "Domain Examples" on page 26
- "xvm CLI and Domains" on page 28
- "XVM Manager GUI and Domains" on page 29

Overview of Domains

XVM has the following *domains*:

- *Cluster domain*, in which an XVM physical volume (physvol) is owned by a CXFS cluster and can be controlled by any of the nodes in that cluster.
- *Local domain*, in which an XVM physvol is owned by a single node and can be controlled only by that node

Only the owner of an XVM physvol can modify the configuration on that physvol. There may be an XVM physvol that is seen by a host but owned by another host or another cluster. XVM recognizes the disk and marks it as foreign. A disk without an XVM label is shown as unlabeled.

CXFS Service Requirements for Cluster Domain

Cluster domain on a server-capable administration node requires that both the cxfs_cluster service and the cxfs service are started; on a client-only node, cluster domain requires that the cxfs_client services is started.

Domain Examples

Figure 2-10 illustrates a physvol that is controlled by a local owner. In this example, the physvol lucy (reported by the xvm command as phys/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.

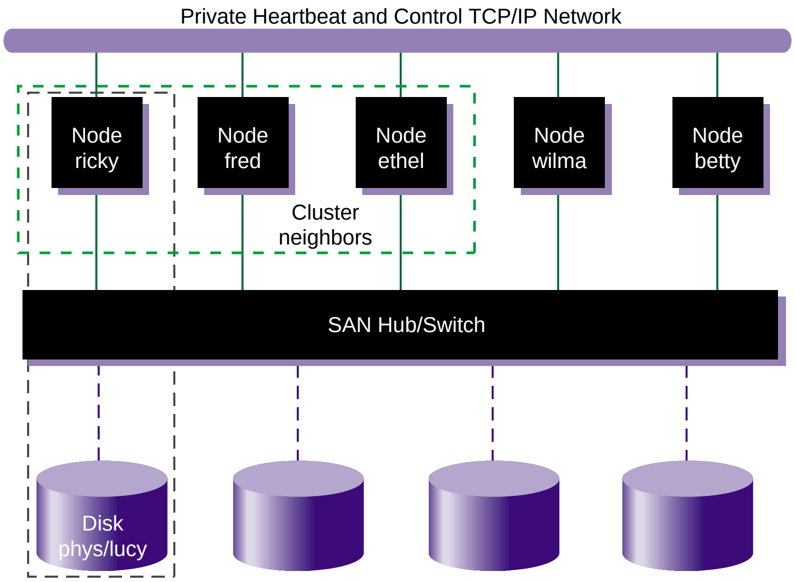


Figure 2-10 XVM Physvol in Local Domain

In the configuration illustrated in Figure 2-10, 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 84.)

Figure 2-11 illustrates a physvol 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 `ethyl`.

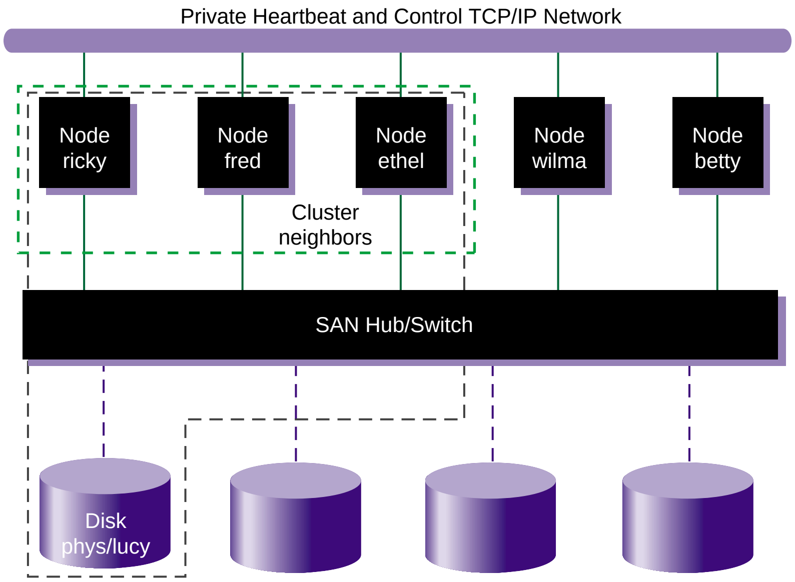


Figure 2-11 XVM Physvol in Cluster Domain

In the configuration illustrated in Figure 2-11, 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.

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

`xvm` CLI and Domains

You can bring up the `xvm` CLI in either local or cluster domain to perform administrative tasks for the volumes in that domain:

- If the required CXFS services have been started, `xvm` is brought up in cluster domain by default. The `xvm:cluster>` prompt. indicates that all XVM physvols you create are in the cluster domain. See "CXFS Service Requirements for Cluster Domain" on page 26

- When the required services are not started, xvm is brought up in local mode (with no warning message about stopped services). The xvm:local> prompt indicates that all XVM physvols you create are in the local domain.

  Note: If the required services have not been started, you cannot set the domain to cluster.

You can explicitly specify the domain by invoking xvm with the -domain option, as described in "Invoking the xvm CLI" on page 63 or within an xvm session by using the set domain command, as described in "Changing the Current Domain with the set Command" on page 82.

When you are running XVM in the cluster domain, by default you can see and modify only the XVM physvols that are also in that 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 must 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 XVM 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 "Invoking the xvm CLI" on page 63.

You can change the owner of an existing physvol by using the give command to give that physvol to a different owner, whether that owner is a single node or a cluster. If the node or cluster that currently owns the physvol is unable to execute the give command, use the steal command to change the domain of an XVM physvol. For information on the give and steal commands, see:

- "Gracefully Transferring Ownership of a Disk or Physvol with the give Command" on page 90
- "Forcibly Transferring Ownership of a Foreign Disk with the steal Command" on page 90

XVM Manager GUI and Domains

The **XVM Manager** graphical user interface (GUI) also operates in either domain. See Chapter 10, "XVM Manager GUI" on page 175.

Physical Disk Administration

This section discusses the following:

- "Formatting Physical Disks" on page 30
- "Creating Physvols" on page 34
- "Managing Physvols" on page 34
- "Destroying Physvols" on page 36

See also "Gathering Statistics for Physical Disks and Logical Resources" on page 49.

Formatting Physical Disks

This section discusses the following:

- "Partition Layout with GPT Disk Format" on page 30
- "Preparing to Configure XVM Volumes in the Local Domain" on page 31

Partition Layout with GPT Disk Format

XVM is used with globally unique identifier (GUID) partition table (GPT) disks. The GPT label puts header data in sector 1 of a logical unit (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 volumes, you must use the Linux `parted`(8) command to format the disk with one partition that must start before block 64. The partition is used for both the XVM metadata and the user's data and it typically includes all of the space on the disk that is not used by the GPT label.

Figure 2-12 shows a GPT disk that is formatted for XVM.

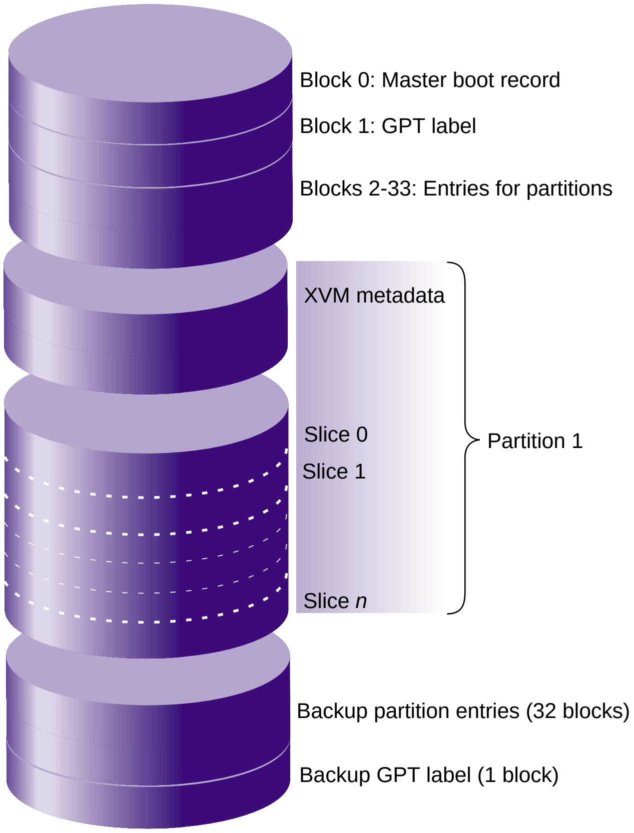

Block 0: Master boot record
Block 1: GPT label
Blocks 2-33: Entries for partitions
XVM metadata
Slice 0
Slice 1
Slice *n*
Partition 1
Backup partition entries (32 blocks)
Backup GPT label (1 block)

Figure 2-12 GPT Disk Layout for XVM

For more information, see "Making a GPT Label" on page 164.

Preparing to Configure XVM Volumes in the Local Domain

Note: These instructions only apply to XVM in the local domain. For information on using XVM with CXFS, see *CXFS 7 Administrator Guide for SGI InfiniteStorage* and *CXFS 7 Client-Only Guide for SGI InfiniteStorage*.

Procedure 2-1 Preparing to Configure XVM Volumes in the Local Domain

Do the following:

1. Install the `SGI XVM Standalone` pattern according to the instructions in the ISSP release note.

2. Confirm that the correct modules are loaded. You can use the `lsmod` command, as in the following example:

   ```
   # lsmod | egrep 'xvm|pm'

      sgi_xvm               513174  0
      sgi_pm                 97247  19
      sgi_os_lib            128203  4 sgi_xvm,sgi_pm
   ```

3. Verify that a LUN chosen by XVM is not in use by another subsystem:

   a. View all of the paths to disks and partitions on the system by executing the following:

      ```
      # ls /dev/disk/by-path
      ```

      XVM will select the LUNs that could potentially be used as XVM LUNs.

   b. View the potential XVM LUNs by executing the following:

      ```
      # xvm show -v unlabeled
      ```

      For example:

```
# xvm show -v unlabeled
Unlabeled disk unlabeled/dev/pm/SGI-TP9700--lun6-600a0b8000269d1e0000c9b14d31a849
================================
using paths:

/dev/disk/by-path/pci-0000:08:03.0-fc-0x22000011c61dd97e-lun-0 <sdaa 65:160> affinity=none ws

/dev/disk/by-path/pci-0000:08:03.0-fc-0x21000011c61dd97e-lun-0 <sdab 65:176> affinity=none ws

/dev/disk/by-path/pci-0000:08:03.1-fc-0x21000011c61dd97e-lun-0 <sdbn 68:16> affinity=none ws

/dev/disk/by-path/pci-0000:08:03.1-fc-0x22000011c61dd97e-lun-0 <dbm 68:0> affinity=none ws
```

```
Unlabeled disk unlabeled/dev/pm/SGI-TP9700--lun7-600a0b8000269d1e0000c9b14d31a849
================================
using paths:

/dev/disk/by-path/pci-0000:08:03.0-fc-0x22000011c61dd850-lun-0 <sdao 66:128> affinity=none ws

/dev/disk/by-path/pci-0000:08:03.0-fc-0x21000011c61dd850-lun-0 <sdaq 66:160> affinity=none ws

/dev/disk/by-path/pci-0000:08:03.1-fc-0x22000011c61dd850-lun-0 <sdca 68:224> affinity=none ws

/dev/disk/by-path/pci-0000:08:03.1-fc-0x21000011c61dd850-lun-0 <sdcb 68:240> affinity=none ws

Unlabeled disk unlabeled/dev/pm/SGI-TP9700--lun8-600a0b8000269d1e0000c9b14d31a849
================================
using paths:

/dev/disk/by-path/pci-0000:08:03.0-fc-0x22000011c61e1a46-lun-0 <sdt 65:48> affinity=none ws

/dev/disk/by-path/pci-0000:08:03.1-fc-0x22000011c61e1a46-lun-0 <sday 67:32> affinity=none ws

/dev/disk/by-path/pci-0000:08:03.1-fc-0x21000011c61e1a46-lun-0 <sdbd 67:112> affinity=none ws

/dev/disk/by-path/pci-0000:08:03.0-fc-0x21000011c61e1a46-lun-0 <sdu 65:64> affinity=none ws
```

Note: This output groups together all the paths to a single LUN. Do not use a given disk device as an XVM volume if you are already using it to mount a filesystem outside of XVM. XVM cannot always detect that a LUN is already in use by some other subsystem, so verify that the LUN is not in use before proceeding.

4. Format each disk you will use for the XVM volume as a GPT disk. See "Partition Layout with GPT Disk Format" on page 30.

5. *(Optional)* Set up Command Tagged Queuing (CTQ) .

6. *(Optional)* Enable write-caching.

Creating Physvols

In order to create XVM logical volumes on a formatted physical disk, you must use the `label` command to write an XVM physvol label on the disk, which allows XVM to control layout of its data in the GPT partition. See "Assigning Disks to XVM with the `label` Command" on page 82.

Note: In a CXFS cluster, any XVM physvols that will be shared must be physically connected to all nodes in the cluster.

When you label a new XVM disk, XVM automatically probes the disk as part of the label process. All disks are also probed when the system is booted to determine which disks are XVM disks.

Note: However, if you add a previously labeled XVM physvol to a running system, you must explicitly use the `probe` command to probe the disk in order for the system to recognize the disk. See "Probing a Physical Volume with the `probe` Command" on page 88.

You cannot label a disk as an XVM disk if the disk contains any partitions that are currently in use as mounted filesystems.

Managing Physvols

This section describes the following:

- "Displaying Physvols" on page 34
- "Changing the Owner of a Physvol" on page 35
- "Adding a Physvol to Running System" on page 35
- "Replacing a Physvol" on page 35
- "Renaming a Physvol" on page 35

Displaying Physvols

To display information about physvols, both labeled and unlabeled, use the `show` command. You can also use the `show` command to display information about disks that are foreign to the current node, as described in "Displaying Physical Volumes with the show Command" on page 84.

Changing the Owner of a Physvol

To change the owner of an existing XVM physvol, giving that physvol to a different local or cluster owner, use the give command. If the node or cluster that currently owns the physvol is unable to execute the give command, use the steal command to change the domain of a physvol. See:

- "Gracefully Transferring Ownership of a Disk or Physvol with the give Command" on page 90
- "Forcibly Transferring Ownership of a Foreign Disk with the steal Command" on page 90

Adding a Physvol to Running System

When the system boots, 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 so that the kernel recognizes the disk as an XVM disk. See "Probing a Physical Volume with the probe Command" on page 88.

Replacing a Physvol

XVM lets you 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. To dump the commands to a file that you can use to regenerate a physvol label, use the dump command.

Note: When you dump the commands to regenerate a physvol label, you must separately and explicitly dump the commands to regenerate the volume element tree that leads to the physvol, as described in "Reconstructing Volume Elements with the dump Command" on page 104.

Renaming a Physvol

You can rename a physvol with the name option of the change command. See "Modifying Volume Elements with the change Command" on page 88

Destroying Physvols

To remove a physvol from the system, use the `unlabel` command to remove the physvol label from an XVM disk and restore the original partitioning scheme. The `unlabel -force` command deletes each slice that currently exists on the physvol unless the slice is part of an open subvolume (and therefore its deletion will cause the subvolume state to go offline). See "Removing Disks from XVM with the `unlabel` Command" on page 89.

Creating Logical Resources

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

This section discusses the following:

- "Creating an XVM Topology Tree" on page 37
- "Automatic Creation of a Volume and Subvolume" on page 37
- "Naming Volume Elements" on page 37
- "Attaching Volume Elements" on page 38
- "Detaching Volume Elements" on page 39
- "Creating Slices" on page 40
- "Creating Concats" on page 41
- "Creating Stripes" on page 41
- "Creating Mirrors" on page 41
- "Creating Volumes" on page 44
- "Creating Subvolumes" on page 44
- "Reorganizing Volumes" on page 44

Creating an XVM Topology Tree

You can create an XVM topology tree starting from the top or the bottom. Only those trees that end in a slice will have labels written to disk and will therefore be persistent across boots.

While you are building the topology tree, you may find it useful to display the existing hierarchy by using the `-topology` option of the `show` command.

Note: When you create stripes, mirrors, concats, subvolumes, and volumes, you have the option of not specifying which children 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.

Automatic Creation of a Volume and Subvolume

When you create volume elements other than volumes, 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. Unless you specify the subvolume type, a subvolume of type `data` is automatically generated for the volume. 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 name automatically, a different name might be generated when the system reboots. Slices, however, are a special case; when the system generates a volume name for a slice, that volume name is permanent and persistents across reboots.

You can make a temporary volume name persistent across reboots by using the `change` command to rename the volume; see "`change`" on page 98.

Naming Volume Elements

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

- As described in "Automatic Creation of a Volume and Subvolume" on page 37, 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.

- You can name a subvolume explicitly only if it is of a user-defined type. See "Subvolume" on page 15.
- You can name concats, mirrors, or stripes explicitly when you create them. In this case, 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 XVM with the `label` Command" on page 82.

  If you do not name concats, mirrors, or stripes explicitly, you must specify that a temporary name should be generated.
- When XVM generates a temporary name, it uses the following format for most objects, where *type* is the type of object and *N* is a counter that increments with each object of that type created since the `xvm` module, usually at system boot time:

  typeN

  There is a separate counter for each object type.
- Slices are named automatically when you create them.

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

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 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 71.

Attaching Volume Elements

To explicitly attach a lower-level volume element to a higher-level volume element, use the `attach` command. You can also attach elements together by creating volume elements.

XVM enforces the following rules and restrictions:

- You attach a lower-level item in the topology tree (the *source* of the attach) to a higher-level item (the *target* of the attach):
  - The source of an attach must be a subvolume, concat, mirror, stripe, or slice
  - A subvolumes can be attached only to a volume
  - A volume cannot be the source of an attach because it is the highest element in the hierarchy
- Subvolumes can have only one child.
- A volume cannot have more than one subvolume of a given type. See "Subvolume" on page 15.
- A mirror cannot have more than eight legs.
- If you specify a topology position (slot) 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.
- You can add a volume element only to the end (rightmost position) of a concat and you can add mirror legs; you cannot insert a volume element in the middle of a concat or change the size of a stripe.

  When you attach a volume element to a mirror, this initiates a mirror synchronization known as a *revive*. A message is written to the system log when this process is complete. You cannot halt a mirror revive after it has begun except by detaching all but one of the legs 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

To detach a volume element from its parent, use the `detach` command. When you detach a volume element, a new volume 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 `data` subvolume is also automatically generated for the volume element you are detaching unless the volume element you are detaching is itself a subvolume of a different type.

An element of an open subvolume can only be detached if its detachment will not cause the subvolume to go offline. The only element that can be detached from an open subvolume is a mirror leg that is not the last leg of that mirror. You cannot detach the last valid leg of an open mirror from that mirror, because this will cause the mirror to go offline.

Note: If a subvolume is not open, you must use the `-force` option to detach the last leg of a mirror or a mirror leg that is being revived. The `detach -safe` command imposes this restriction even if the subvolume is not open. For more information, see "`detach`" on page 99.

Creating Slices

To create a slice from a block range of an XVM physvol, use the `slice` command. 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:

- Create a slice out of all of the blocks of a physvol.
- Divide a specified address range into equal parts, with each part a different slice.
- Slice multiple physvols at once.
- Specify that a slice must start on a sector that is an even multiple of some number, relative to the start of the logical unit (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 redundant array of independent disks (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

To create a volume element that concatenates all of its children into one address space (known as a *concat*), use the `concat` command.

During concat creation, XVM enforces the rules described in "Attaching Volume Elements" on page 38.

Creating Stripes

To create a volume element (known as a *stripe*) composed of multiple chunks that alternate across the address space, thereby allowing parallel I/O operations for higher performance, use the `stripe` command.

Note: It is possible to create a stripe that is made up of volume elements of unequal size, but this will result in unused space on the larger volume elements.

During stripe creation, XVM enforces the rules of attachment described in "Attaching Volume Elements" on page 38.

For information on configuring stripes that span two host bus adapters (HBAs), see "Volumes and Failover" on page 11.

Creating Mirrors

A *mirror* is a volume element that provides data redundancy in that each *mirror leg* (child) is an exact duplicate of every other mirror leg. If any leg fails, all data written to the mirror will still be available on all other legs, so that access to the mirror can continue without interruption. A mirror is useful when you require data redundancy and availability. To create a mirror, use the `mirror` command.

When you create a mirror that has more than one leg, a message indicating that the mirror is reviving (or beginning the process of mirroring the data) is written to the system log. Another message is written to the system log 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 system log.

You cannot halt a mirror revive after it has begun except by detaching all but one of the legs of the mirror. For more information, see "Mirror Revives" on page 173.

During mirror creation, XVM enforces the rules of attachment described in "Attaching Volume Elements" on page 38.

When you create a mirror, you can optionally set the following characteristics:

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

For large mirror components, the revive process may take a long time. Therefore, you should consider using the `-clean` and `-norevive` options to limit revives as appropriate.

The following sections describe each of these options:

- "Read Policies" on page 42
- "Primary Leg" on page 43
- "Mirror Revive at Creation" on page 43
- "Mirror Revive at Reboot" on page 44

Read Policies

XVM lets you specify one of the following read policies for a mirror:

Policy	Description
`rrobin`	Balances the I/O load among the legs of the mirror, blindly reading in a round-robin fashion.
`sequential`	Routes sequential I/O operations to the same leg of the mirror.

Figure 2-13 illustrates how data is read from the legs of a mirror with an `rrobin` read policy. The wedges represent units of data that you are reading. The first operation reads from the first leg, the second operation reads from the second leg, the third operation reads from the third leg, and the fourth operation reads from the first leg again.

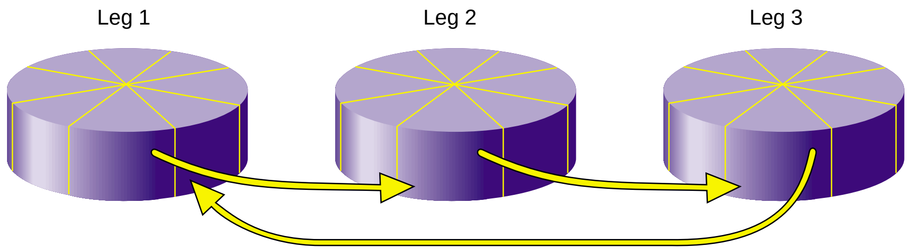


Figure 2-13 Reading Data from a Mirror with a Round-Robin Read Policy

Figure 2-14 illustrates how data is read from the legs of a mirror with a `sequential` read policy, showing that the different mirror legs are not accessed for a single sequential I/O operation.

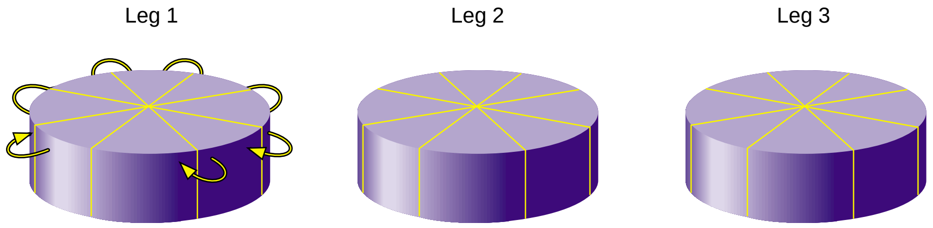


Figure 2-14 Reading Data from a Mirror with a `sequential` Read Policy

You can modify the read policy with the `change` command; see "`change`" on page 98.

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*. To redefine the primary leg, use the `change` command. See "`change`" on page 98.

Mirror Revive at Creation

When you create a mirror, you can use the `mirror -clean` option to specify that the legs of the mirror do not need to be revived when they are created. This option is useful when the legs of the mirror are already in synchronization or when the mirror is new and all data will be written before being read. See "`mirror`" on page 92.

Mirror Revive at Reboot

When you create a mirror, you can use the `mirror -norevive` option 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`..

If you are creating a mirror for a temporary filesystem that is in active use, you can specify `-norevive` (but not `-clean`). If you are creating a mirror for a temporary filesystem that is not in active use, then you can specify both `-norevive` and `-clean` because 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 any leg) then specifying `-clean` by itself eliminates the initial revive. See "`mirror`" on page 92.

Creating Volumes

To create a volume explicitly, use the `volume` command; see "`volume`" on page 97. Volumes may also be created automatically when you create a volume element, as described in "Automatic Creation of a Volume and Subvolume" on page 37.

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, XVM enforces the rules described in "Attaching Volume Elements" on page 38.

Creating Subvolumes

To create a subvolume explicitly, use the `subvolume` command; see "`subvolume`" on page 96. Subvolumes of type `data` may also be created automatically when you create a volume element, as described in "Automatic Creation of a Volume and Subvolume" on page 37.

When you use the `subvolume` command, you can specify its child (a stripe, concat, mirror, or slice). If you do not specify a child, an empty subvolume is created.

For more information, see "Subvolume" on page 15.

Reorganizing Volumes

As you create an XVM topology, you can use the `attach` and `detach` commands to reorganize its elements; see "`attach`" on page 97, and "`detach`" on page 99.

You can also reorganize the volume elements by using the remake command; see "remake" on page 99. The remake command collapses holes in the topology and rearranges children under a given 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

This section discusses the following:

- "Displaying the State of Volume Elements and Physvols" on page 45
- "Disabling Volume Elements" on page 47
- "Bringing a Volume Element Online" on page 47
- "Making Online Changes" on page 48
- "Saving and Regenerating an XVM Configuration" on page 48

See also "Gathering Statistics for Physical Disks and Logical Resources" on page 49.

Displaying the State of Volume Elements and Physvols

To display information about volume elements and physvols, use the show command.

A volume element or physvol can have one or more of the following states:

State	Description
accessible	The underlying devices can be reached.
clean	The mirror leg has been created with -clean option of the mirror command to specify that the leg does not need to be revived on creation; it will be revived on subsequent boots.

Note: After 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.

`disabled`
The volume element has been disabled with the `change disable` command. You must explicitly enable the volume element with `change enable` before it can be brought online.

`incomplete`
The volume element is missing one or more children. For all volume elements other than mirrors, the missing children must be attached or the volume element must be remade with the `remake` command before the volume element can be brought back online.

`inconsistent`
One or more children of the volume element may have changed since the last failure: a child may have been attached or detached, a missing child may have come back into place, or the state may have changed. See "Volume Element in `inconsistent` State" on page 225.

`mediaerr`
The volume element has encountered at least one media error.

`offline`
Volume element: offline. No I/O can be done to the volume element. When a volume element is in this state, you must examine the volume element's topology and note the state each child. There should always be at least one other state displayed that can help determine why the volume element is offline.

Physvol: the system does not know where the physvol resides. This can occur in a cluster where the system obtains information from another node but does not have access to the disk.

`online`
Volume element: online and properly configured. It is able to be opened, or it is already open.

Physvol: the system has successfully discovered the disk. After the system has discovered the disk, it will not move back to an offline state.

`open` The volume element is part of an open subvolume.

`pieceoffline` The volume element has a child (piece) that is offline. For volume elements other than mirrors, the offline children must be brought back online before the volume element can be brought online.

`reviving` The mirror leg is in the process of synchronizing. The data on this mirror leg is not valid until the revive completes.

`reviving:queued` The mirror is targeted for a revive, but the revive has not started yet.

`reviving:`*XX*`%` The system is in the process of synchronizing this mirror and is *XX*% complete.

`tempname` The volume element has a name that may not be persistent across reboots.

`valid` The volume element is up-to-date; the data is readable.

Disabling Volume Elements

To manually disable a volume element, use the `change` command. 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. To bring the volume element back online, use the `change` command.

Making Online Changes

To insert a mirror or a concat above another volume element, use the insert command. You can use this command 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.

To remove a layer from a tree, use the collapse command. Generally, you use a collapse command to reverse a previous insert operation. See "collapse" on page 101.

Saving and Regenerating an XVM Configuration

To save a volume configuration, use the dump command to output the configuration commands to a file. You can then use this file to regenerate the XVM configuration.

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: When you dump the commands to regenerate a volume topology tree, you must also separately and explicitly dump the commands to regenerate the physvols that the tree leads to, as described in "Reconstructing Volume Elements with the dump Command" on page 104.

For example, the following shell command line will dump all xvm volume elements and physvols to the file /var/xvm_config:

```
# xvm dump -topology -f /var/xvm_config phys/’*’ vol/’*’
```

Destroying Logical Resources

This section discusses the following:

- "Deleting Volume Elements" on page 49
- "Removing Configuration Information for Inaccessible Disks" on page 49

Deleting Volume Elements

To delete a volume element, use the `delete` command. 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 use the following options to override this restriction:

- `-all` specifies that all of the children will be deleted
- `-nonslice` specifies all of the children other than slices will be deleted (the slices are detached and a volume and `data` subvolume are automatically generated for the slices)

An element of an open subvolume can only be deleted if its deletion will not cause the subvolume to go offline. The only element that can be deleted without putting the subvolume offline is a mirror leg that is not the last leg of that mirror.

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 volumes that include that disk. To recover from this situation, use the `reprobe` command to remove previous configuration information from the kernel.

For information on using the `reprobe` command, see "Removing Configuration Information with the `reprobe` Command" on page 105.

Gathering Statistics for Physical Disks and Logical Resources

XVM can maintain statistics for physvols, subvolumes, stripes, concats, mirrors, and slices. To turn statistics on and off and to reset the statistics, use the `stat` option of the `change` command. For more, see Chapter 8, "Statistics" on page 169.

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

Chapter 3

XVM Best Practices

This section discusses the following:

- "Configuration Best Practices" on page 51
- "Administrative Best Practices" on page 57

Configuration Best Practices

This section discusses the following:

- "Use XVM Configuration Tools Appropriately" on page 51
- "Use One Slice Per LUN" on page 52
- "Use the Default Data Subvolume" on page 52
- "Explicitly Name Volume Elements" on page 52
- "Use an Appropriate Stripe Unit Size and Alignment" on page 52
- "Use Mirrors Efficiently" on page 53
- "Restripe Efficiently" on page 55
- "Categorize Portions of a Complex Volume" on page 55
- "Use Automatic Probing Wisely" on page 55
- "Do Not Create Slices Within the RAID Exclusion Zone" on page 56
- "XVM Configuration for Mixing SSD and HDD Media" on page 56

Use XVM Configuration Tools Appropriately

Do not attempt to make simultaneous configuration changes using the xvm(8) command-line interface (CLI) and the **XVM Manager** graphical user interface (GUI). Use one tool at a time.

The GUI provides a convenient display of XVM components. If you are using XVM in a cluster environment, you should use the **XVM Manager** or the **CXFS Manager**

GUI to see your progress and to avoid adding or removing CXFS nodes too quickly. After defining a CXFS node, you should wait for it to appear in the view area before adding another node. After defining a cluster, you should wait for it to appear before you add nodes to it. If you make changes too quickly, errors can occur. For more information, see Chapter 10, "XVM Manager GUI" on page 175.

Use One Slice Per LUN

Use LUNs of equal size and one slice for each LUN. You should assemble stripes for any desired performance characteristics out of the slices or mirrors. If more size is needed, use a concat at the top level to allow an arbitrary number of volume elements to be connected.

Use the Default Data Subvolume

For most configurations, the default `data` subvolume is the only subvolume required.

Explicitly Name Volume Elements

In order to create a name that will persist across reboots, SGI recommends that you explicitly name a volume when you create a volume element or an empty volume. This will reduce the risk of data loss.

Note: 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.

If you have already created volumes that you did not name explicitly, you can use the `change` command to assign these volumes permanent names. See "Modifying Volume Elements with the `change` Command" on page 88.

Use an Appropriate Stripe Unit Size and Alignment

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 of your XVM LUNs to transfer in parallel, but all of the disks in the RAID will be accessed in parallel, in units of the same size. By default, a stripe unit must be a multiple of 32 512-byte blocks.

To get the best performance, make the XVM stripe unit be the same as the RAID stripe width, so that the RAID gets an optimally sized chunk of data to store.

Use the single-partition method for GTP label creation. For example: (line breaks added for readability):

```
# parted /dev/disk/by-path/pci-0000:03:00.0-fc-0x20360080e5232098:0x0000000000000000 "unit s mklabel gpt"
# parted /dev/disk/by-path/pci-0000:03:00.0-fc-0x20360080e5232098:0x0000000000000000 \
"unit s mkpart primary xfs 34 -8192"
```

For LUNs with a power-of–2 number of data drives, XVM will align by default. For non-power-of–2, use the following formula to calculate the alignment:

RAID_segment_size_in_KiB `*` *number_of_data_drives* `* 2`

This value will be in 512–byte basic blocks. For example, using a 6+1 RAID with a 64KiB-segment LUN:

```
64KiB * 6 * 2 = 768 blocks
```

The `slice` command would be:

```
# xvm slice -all -align 786 phys/is5500_lun0
```

Use Mirrors Efficiently

This section discusses the following:

- "Use Mirrors with Identical Components" on page 53
- "Place Mirrors at the Bottom of the XVM Topology Tree" on page 54
- "Avoid Unnecessary Revives " on page 54
- "Set Mirror Revive Resources Properly" on page 54

Use Mirrors with Identical Components

To make the best use of space, create mirrors with components of identical size. If the components are not identical, there will be unused space in the larger components.

Place Mirrors at the Bottom of the XVM Topology Tree

Place mirrors at the lowest possible level (below any stripes) to maximize independence and minimize synchronization times during revive operations. This provides the redundancy of a mirror with improved performance.

Avoid Unnecessary Revives

For large mirror components, the revive process may take a long time. Consider the following:

- For a new mirror that does not need mirroring at creation, use the `-clean` option to specify that the mirror will revive at reboot but not creation. An example is creating a new filesystem; because everything will be written before it will be read, there is no need for a revive beforehand.
- For new mirror that you will use for scratch filesystems (such as `/tmp`) that will never need to be synchronized, use the `-norevive` option to specify that the mirror will never revive.

Set Mirror Revive Resources Properly

You should set the `xvm_max_revive_rsc` and `xvm_max_revive_threads` XVM system-tunable kernel parameters appropriately for your site’s mirror revive performance requirements. Increasing `xvm_max_revive_rsc` will increase the data throughput per thread, and increasing `xvm_max_revive_threads` will increase the number of parallel I/O processes used in reviving. Decreasing the resources causes less interference with an open filesystem at the cost of increasing the total time to revive the data.

As a general guideline:

- Increase the revive resource tunable values if you want to revive as quickly as possible and do not mind the performance impact on normal I/O processes
- Decrease the revive resource tunable values if you want to have a smaller impact on a particular filesystem

See Appendix A, "XVM System Tunable Kernel Parameters" on page 233.

Restripe Efficiently

To restripe an existing volume without deleting the slices, do the following:

- Use the following option to delete the XVM structure **other than** the slices:

  ```
  delete -nonslice
  ```

- Use the `stripe` command as needed to create new stripes

Categorize Portions of a Complex Volume

Sometimes it may be useful to categorize volume elements by name. For example, you may want to name a portion of a volume `fast` so that you can search for volumes that have fast stripe objects. For example:

```
xvm:cluster> stripe -volname myvol -vename fast0 -unit 128 slice/lucys2 slice/rickys0 slice/ethyls0 slice/freds0
</dev/cxvm/myvol> stripe/fast0
xvm:cluster> stripe -volname myvol -vename fast1 -unit 128 slice/lucys3 slice/rickys1 slice/ethyls1 slice/freds1
</dev/cxvm/myvol> stripe/fast1
```

When you name the stripes as in the preceding example, you can use a wildcard to show both `fast0` and `fast1` stripes:

```
xvm:cluster> show -topology stripe/fast*
stripe/fast0        23705088 online
     slice/lucys2              5926340 online
     slice/rickys0             5926340 online
     slice/ethyls0             5926340 online
     slice/freds0              5926340 online
stripe/fast1        23705088 online
     slice/lucys3              5926340 online
     slice/rickys1             5926340 online
     slice/ethyls1             5926340 online
     slice/freds1              5926340 online
```

Use Automatic Probing Wisely

After you label a device, XVM will automatically probe it and any unlabeled disks in order to locate alternate paths. Disks are also probed when the system is booted and when you explicitly execute an XVM probe command. In most cases, this default behavior is appropriate.

However, a probe can be slow, and it is necessary to probe a newly-labeled device only once. For example, if you are executing a series of individual label commands, you might wish to disable automatic probing using one of the methods described in "Controlling Automatic Probing with the label and set Commands" on page 83.

Do Not Create Slices Within the RAID Exclusion Zone

Due to some performance issues in XVM, SGI strongly recommends that the XVM slice be completely outside the RAID exclusion zone. By default, the xvm CLI label command places the user data area entirely outside of the exclusion zones, so you do not need to consider the exclusion zones in allocating slices. See "Making an XVM Volume Using a GPT Label" on page 163.

Note: If it is necessary to allows the user space to overlap the RAID exclusion zones, you can use the following command to override the default layout behavior:

```
xvm:cluster> label -use-exclusion-zones unlabeled_disk
```

XVM Configuration for Mixing SSD and HDD Media

Different types of media are appropriate for different uses:

- Solid-state drive (SSD) media is appropriate for small latency-sensitive operations
- Rotating hard-disk drive (HDD) media is appropriate for larger bandwidth- and capacity-intensive operations

The ibound mount option specifies where the filesystem places the inodes, which lets you use SSD media for a filesystem's inodes and HDD media for the file data. In this case, you should create an XVM volume that concatenates a slice of SSD media with HDD media and then use the ibound mount option to restrict filesystem inode allocation to the fast SSD media at the beginning of the XVM volume.

Note: The ibound mount option implies inode32 behavior and is therefore incompatible with the inode64 mount option. Behavior of the inode32 mount option is not affected.

For more information, see the chapter about enhanced NFS extensions in *XFS for Linux Administration*.

Administrative Best Practices

This section discusses the following:

- "Save the XVM Configuration Before Making Changes" on page 57
- "Do Not Use a Given Disk for both XVM and Non-XVM" on page 57
- "Specify Path Failover for Non-ALUA RAID" on page 58
- "Do Not Override ALUA RAID Settings via `/etc/failover2.conf`" on page 59
- "Do Not Run `fsck` on Filesystems that Use XVM Devices" on page 60
- "Give Rather than Steal Ownership" on page 60
- "Unmount Filesystems Before Making XVM Topology Changes" on page 60
- "Unmount a CXFS Filesystem Before Growing It" on page 60
- "Do Not Use an XVM Volume as a Dump Device" on page 61
- "Use `xvm` Commands Carefully in Scripts" on page 61
- "Do Not DIsplay I/O Activity in `show` Commands" on page 61
- "Understand the Use of Symbolic Links and Device Names in Output" on page 61

Save the XVM Configuration Before Making Changes

It is possible for XVM labels to become corrupted. Therefore, you should use the XVM `dump` command to make a copy of the configuration before making a change so that you can recover from potential problems introduced by the change. You should save the `dump` output into a filesystem other than the one being dumped.

Do Not Use a Given Disk for both XVM and Non-XVM

Do not use a given disk device as an XVM volume if you are already using it to mount a filesystem outside of XVM. XVM cannot always detect that a LUN is already in use by some other subsystem, so verify that a LUN is available before creating XVM physvols on it.

Specify Path Failover for Non-ALUA RAID

This section discusses the following for RAIDs that do not use the asymmetrical logical unit access (ALUA) feature:

- "Define the `/etc/failover2.conf` File on Every Node" on page 58
- "Do Not Use `affinity` or `preferred` Keywords for ALUA RAID" on page 58
- "Set Nonzero `affinity` Values" on page 58
- "Periodically Check for Unassigned Paths" on page 59
- "Change Affinity Consistently Across the Cluster" on page 59

Define the `/etc/failover2.conf` File on Every Node

If you use non-ALUA RAID and if you spread I/O across controllers, you should define the `/etc/failover2.conf` file to ensure that I/O is done efficiently and is directed to the path that you prefer; unnecessary switching between RAID controllers can degrade performance considerably. See Chapter 6, "XVM Path Failover" on page 107.

In a cluster configuration, be sure that the `/etc/failover2.conf` file is correct and consistent on every node in the cluster.

Do Not Use `affinity` or `preferred` Keywords for ALUA RAID

For RAID that supports the ALUA feature, you should not use the `affinity` or `preferred` keywords for normal operation. Those keywords can be used in a `failover2.conf` file to override the settings read from the RAID in order to work around a problem.

Set Nonzero `affinity` Values

You may find it useful to specify affinity values starting with `affinity=1` and specify a nonzero value for all paths. This makes it easy to detect those paths that have not yet been configured because they are assigned the default of `affinity=0`. See "Set Appropriate `affinity` Values for Non-ALUA LUNs" on page 115.

Periodically Check for Unassigned Paths

If you used the method recommended in "Set Nonzero affinity Values" on page 58, you should periodically examine the show -v output for new LUNs to find any that are unassigned. You may wish to write a script to perform this function and execute it via a cron(8) job.

Change Affinity Consistently Across the Cluster

If you change the affinity setting for a path in the cluster domain, you should include the -cluster option so that the setting is consistent across all nodes in the cluster. For example:

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

If you change the preferred path, you should include the -cluster option if the switch will move to a different affinity group. For example, suppose the following:

```
pathA affinity=1 preferred
pathB affinity=1
pathC affinity=2
pathD affinity=2
```

You could switch the preferred path to pathB for a single node in the cluster because it has the same affinity setting as pathA:

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

However, if you want to use pathC as the preferred path, you should include -cluster because it is part of a different affinity group:

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

Do Not Override ALUA RAID Settings via /etc/failover2.conf

Normally, you should not use the /etc/failover2.conf file to override the path settings provided automatically by a RAID that has the ALUA feature. If changes are required, you should make them via the ALUA RAID software.

Do Not Run fsck on Filesystems that Use XVM Devices

It is possible that XVM might not discover all devices associated with XVM volumes by the time that the filesystems listed in /etc/fstab are mounted, meaning that some volumes may not yet be complete at that point. If an fsck command is run on an XFS filesystem when XVM devices are undiscovered, the system may suspend the system boot sequence and require input from the administrator.

Therefore, for XFS filesystems listed in /etc/fstab that use XVM devices, you should set the fsck flag to 0. XVM includes a helper service that mounts all filesystems listed in /etc/fstab that use XVM devices at the time that XVM is started during the boot sequence.

Give Rather than Steal Ownership

You should only use the steal command when ownership cannot be changed by using the give command.

Unmount Filesystems Before Making XVM Topology Changes

You should unmount a filesystem before making XVM topology changes.

A child of an open volume element can only be detached if this will not cause the volume element to go offline. The only child that can be detached without putting the volume element offline is a mirror leg that is not the last leg of that mirror.

In particular, normally you should not execute the following xvm commands on an open volume element:

```
change disable
```

Unmount a CXFS Filesystem Before Growing It

You should unmount a filesystem from the CXFS cluster and mount it locally before growing it. For more information, see the section about growing filesystems in the *CXFS 7 Administrator Guide for SGI InfiniteStorage*.

Do Not Use an XVM Volume as a Dump Device

You should not use an XVM volume as a dump device.

Use `xvm` Commands Carefully in Scripts

If you write scripts that use `xvm`(8) configuration commands, be aware that running multiple commands in quick sequence can cause the commands to fail. An XVM device newly created by one command is held open for an interval by Linux utility programs; subsequent `xvm` commands in the script cannot use the device and therefore fail. Following is common error in this situation:

```
error creating  item:  operation will cause the ve’s
subvolume to go offline
```

Do Not DIsplay I/O Activity in `show` Commands

For best performance in a production system, do not display of I/O activity in the output of the `xvm show` command (that is is, you should use the default setting of `0` for the `pm_display_io_activity` system tunable parameter, which disables this feature).

You should enable `pm_display_io_activity` only while you are performing configuration. See "`pm_display_io_activity`" on page 239

Understand the Use of Symbolic Links and Device Names in Output

XVM uses symbolic links (in the format `/dev/lxvm/`*localXFSname* or `/dev/cxvm/`*CXFSname*) to the actual device names (in the format `/dev/xvm-`*n*). For example, the default `ls`(1) output will show the symbolic link:

```
# ls /dev/lxvm/align-tests0
/dev/lxvm/align-tests0
```

To view the link plus the actual device name, use the `-l` option:

```
# ls -l /dev/lxvm/align-tests0
lrwxrwxrwx 1 root root 8 Sep 30 13:41 /dev/lxvm/align-tests0 -> ../xvm-5
```

Commands such as mount(8) and df(1) show the actual device names. For example, to mount (using the symbolic link) and then show the mounted xfs filesystems:

```
# mount /dev/lxvm/align-tests0 mnt
# mount -t xfs
/dev/sda2 on / type xfs (rw)
/dev/xvm-5 on /root/mnt type xfs (rw)
```

To report on filesystem usage:

```
# df -k
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/sda2             20970644   6269112  14701532  30% /
udev                   4027496      1252   4026244   1% /dev
tmpfs                  4027496         0   4027496   0% /dev/shm
/dev/xvm-5            35802032     32928  35769104   1% /root/mnt
```

To get information by starting with a /dev/xvm-*n* device name, you can display information from the sysfs filesystem as follows:

```
# ls /dev/xvm-*
/dev/xvm-10  /dev/xvm-5  /dev/xvm-6  /dev/xvm-8
# cat /sys/block/xvm-5/vol-info/xvm-domain
lxvm
# cat /sys/block/xvm-5/vol-info/xvm-volname
align-tests0
# cat /sys/block/xvm-5/vol-info/xvm-svol-type
data
```

Chapter 4

Overview of the xvm CLI

This chapter discusses the following:

- "Invoking the xvm CLI" on page 63
- "Summary of xvm CLI Commands" on page 66
- "Online Help for xvm CLI Commands" on page 68
- "xvm CLI Syntax" on page 70
- "Object Names in XVM" on page 71
- "Device Directories and Pathnames" on page 78
- "Command Output and Redirection" on page 78
- "Safe Versus Unsafe Commands" on page 79

Invoking the xvm CLI

This section discusses the following:

- "root Requirement" on page 63
- "Interactive Use" on page 64
- "Shell Use" on page 65
- "Redirection" on page 66

root Requirement

To configure XVM volumes, you must be logged in as root.

However, you can display configuration information even if you do not have root privileges.

Interactive Use

To use the `xvm` CLI interactively, enter the following:

```
# /sbin/xvm
```

If the required CXFS services (see "CXFS Service Requirements for Cluster Domain" on page 26) have been started when you enter this command, you should see the following prompt:

```
xvm:cluster>
```

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

Note: You must start the required services before you can see and access XVM cluster configuration objects.

However, if the services have not been enabled when you enter the `xvm` command, the prompt will appear as follows:

```
xvm:local>
```

To explicitly specify the XVM domain, use the `-domain` option:

```
# /sbin/xvm [-domain] domaintype
```

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

Note: When you are running XVM in the cluster domain, you can see and modify only the XVM physical volumes (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, either change to the local domain with the `set domain` command or use the `local:` prefix when specifying a physvol name. Similarly, when you are running XVM in the local domain, you must change to the cluster domain or include the `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 physvols in the local domain, enter the following:

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

Similarly, if you are running in the local domain but wish to see the XVM physvols in the cluster of which you are a member, enter the following:

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

For more information on XVM domains, see "Local Domain and Cluster Domain" on page 26.

After the command prompt displays, you can enter commands to configure and manage volumes. These commands are executed interactively, as you supply them.

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.)

Shell Use

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

```
# /sbin/xvm [-domain local|cluster][xvmcommand ...]
```

For example, to display the brief syntax for the `concat` command:

```
# xvm help concat
concat -tempname [-pieces N] [-vename concatname] [ve ...]
concat -volname name [-pieces N] [-vename concatname] [ve ...]
```

For more information, see the `xvm`(8) man page.

Redirection

You can redirect a file of xvm CLI commands into the xvm CLI just as you would redirect input into any standard Linux tool. For example, to redirect the contents of the file myscript:

```
# 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 78.

Summary of xvm CLI Commands

Table 4-1 provides a summary of the xvm CLI commands according to function.

Table 4-1 Summary of xvm CLI Commands

Function	Command	Description
Physvol and unlabeled-disk management	change	Changes the attributes of a physvol
	label	Labels a disk for XVM use
	probe	Reads the disk label and determines if the disk is an XVM physvol that must be instantiated
	reprobe	Reads the disk labels and removes previous configuration information from the kernel if it does not match the current disk information or if the disk is inaccessible
	unlabel	Removes an XVM label from a physvol and restores the original disk partitions.
Volume-element creation	concat	Concatenates volume elements into a single address space

Function	Command	Description
	`mirror`	Creates a volume element that maintains identical data images on its children
	`slice`	Create a slice from a specified block range of a physvol
	`stripe`	Creates a stripe group from multiple volume elements
	`subvolume`	Creates a subvolume
	`volume`	Creates a volume and optionally attaches subvolumes
Volume-element manipulation	`attach`	Attaches a set of lower-level volume elements to a higher-level volume element in the XVM topology tree
	`change`	Changes the attributes of a volume element
	`collapse`	Removes a volume element from a topology tree without taking the subvolume offline
	`delete`	Deletes a volume element
	`detach`	Detaches a volume element from its parent
	`insert`	Inserts a concat or mirror into a topology tree without taking the subvolume offline
	`remake`	Remakes a volume element, collapsing holes and optionally reordering the children
Display	`dump`	Dumps out the `xvm` CLI commands required to reconstruct a volume element or physvol
	`show`	Shows information about a volume element or physvol
Ownership	`give`	Transfers ownership of an XVM disk gracefully to another local host or cluster
	`steal`	Transfers ownership of a foreign XVM disk forcibly to the local host or cluster
	⚠	**Caution:** If another host or cluster has the physvol instantiated, configuration corruption can occur. You should use `steal` only when the ownership cannot be changed by using the `give` command.
Path failover	`foconfig`	Process the XVM failover configuration file

Function	Command	Description
	foswitch	Changes the path used to access a physical disk
General	bye (also aliases exit and quit)	Exits the xvm CLI
	help (also alias ?)	Displays the synopsis for a command or (with the -verbose option) the full help text for the command
	set	Sets the domain (to either cluster or local) or enables/disables the autoprobe feature **Note:** If the required services have not been started, you cannot set the domain to cluster. See "CXFS Service Requirements for Cluster Domain" on page 26.

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 commands. You can use a question mark (?) as an alias for the help command. This section discusses the following:

- "Viewing a List of Supported Commands" on page 68
- "Brief Synopsis" on page 69
- "Full Help Text" on page 69

Viewing a List of Supported Commands

To see a list of supported commands, enter the following:

```
xvm:cluster> help
```

Brief Synopsis

To see a brief synopsis of a command, enter the following:

```
xvm:cluster> help keyword
```

The *keyword* is any of the commands listed in Table 4-1 on page 66 plus the general help topics described in Table 4-2.

For example, the following command displays the synopsis for the `slice` command:

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

Table 4-2 General Help Topics

Topic	Description
`grouping-subvolumes`	Displays information about creating a volume with multiple subvolumes
`names`	Displays information on XVM object names
`regexp`	Displays information on the regular expressions that you can use when specifying XVM object names

Full Help Text

To see the full help text for a command, include the `-verbose` option (which you can abbreviate to `-v`):

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

For example, the following command displays the full help for the `slice` command:

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

xvm CLI Syntax

This section discusses the following:

- "Command Abbreviation" on page 70
- "Syntax Rules" on page 70

Command Abbreviation

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 that command. Commands and options are not case sensitive, but object names are case sensitive.

For example, are examples of full commands and their abbreviations:

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

```
xvm:local> show -verbose slice/freds0
xvm:local> sh -v slice/freds0
```

Syntax Rules

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

- Blank lines and lines beginning with the pound sign (#) are interpreted as comments
- Strings between < > characters are interpreted as comments (even if the strings are xvm CLI commands)
- A backslash (\) at the end of a line acts as a continuation character
- An exclamation point (!) at the beginning of a command passes the command to the shell

Object Names in XVM

This section discusses the following:

- "Explicit and Temporary Names" on page 71
- "Naming Rules" on page 72
- "Specifying Objects by Name" on page 72
- "Specifying Objects by Piece Number" on page 74
- "Object Name Examples" on page 76
- "Regular Expressions" on page 77

Explicit and Temporary Names

You can explicitly name the following XVM objects:

- Physvol
- Volume
- Concat
- Mirror
- Stripe

Note: Slice names are always generated by XVM.

If you do not supply a name when you create the object, XVM will generate a temporary name that refers to the object type and generation counter (starting from `0`) within the XVM topology tree, such as the following:

- `vol0` for the first volume generated
- `concat2` for the third concat generated

The generated name may not persist across reboots. To have a permanent name, you must either specify the name at creation time or use the `change` command. See "Modifying Volume Elements with the `change` Command" on page 88.

Naming Rules

The following naming rules apply:

- Names are limited to 32 characters in length
- Names consist of alphanumeric characters and the period (.), underscore (_), and hyphen (-) characters
- Names cannot begin with a digit
- Names are case sensitive
- The following keywords are reserved may not be used to name objects:

  ```
  concat
  mirror
  phys
  raid
  slice
  stripe
  subvol
  unlabeled
  vol
  ```

Specifying Objects by Name

With the exception of subvolumes, objects are specified using the object name, using one of the following forms:

[*domain*:][*prefix*/]*name*
[*domain*:][*prefix*/]*path*

where:

- *domain* is either `local` or `cluster`. You must include *domain* when you specify an XVM object that is not in the current domain, as described in "Invoking the `xvm` CLI" on page 63.
- *prefix* is the type of object, such as `phys` for a physvol. See Table 4-3.
- *name* is the name of the object, such as `fred`.

- *path* is the path to the volume element in the XVM topology tree, either the explicit path from the top of the tree or the relative path from one volume element to another.

  Specifying a path component of two dots (`..`) for a volume element indicates the parent of the volume element. For example, the following command displays the parent of slice `foos0`:

  ```
  xvm:cluster> show slice/foos0/..
  ```

Note: 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.

Because user-defined names are allowed, it is possible to have ambiguities in the XVM object namespace. For example, if there are two objects of different object types named `fred`, specifying `fred` alone is not sufficient to identify the object.

You can also use wildcards; see "Regular Expressions" on page 77.

To remove ambiguity in an object name, include the type prefix followed by a slash, such as `concat/concat1`. Specifying an object type can also make name resolution faster by providing information about the type of object.

Note: The following 3-tuple construction is always unambiguous:

`subvol/`*volname*/*subvol_name*

If you specify an ambiguous name to an `xvm` CLI command without a wildcard, the command will generally produce an error message.

Table 4-3 lists the object prefixes.

Table 4-3 Prefixes Specifying Object Type

Prefix	Object
`concat`	Concat
`foreign`	Foreign disk
`mirror`	Mirror

Prefix	Object
`phys`	Physvol
`slice`	Slice
`stripe`	Stripe
`subvol`	Subvolume
`unlabeled`	Unlabeled disk
`vol`	Volume

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.

The path portion of an unlabeled disk is the filesystem path to the volume partition. For example:

- Explicit path:

  `unlabeled/hw/rdisk/`*diskname*

- Relative path:

  `unlabeled/`*diskname*

The format of the path portion of a foreign disk is the same as the path portion of an unlabeled disk.

For more information, see "XVM Objects" on page 13.

Specifying Objects by Piece Number

XVM volume elements can also be specified using a combination of component names and *piece numbers*, which reflect position of a component in the XVM topology tree. For example, `concat0/0` refers to the zero piece (child) of the volume element `concat0`. The piece number syntax is helpful when you want to target a volume element without knowing its name.

Figure 4-1 shows an XVM topology tree that uses system-generated names for the volume and all of its children, and their corresponding piece numbers.

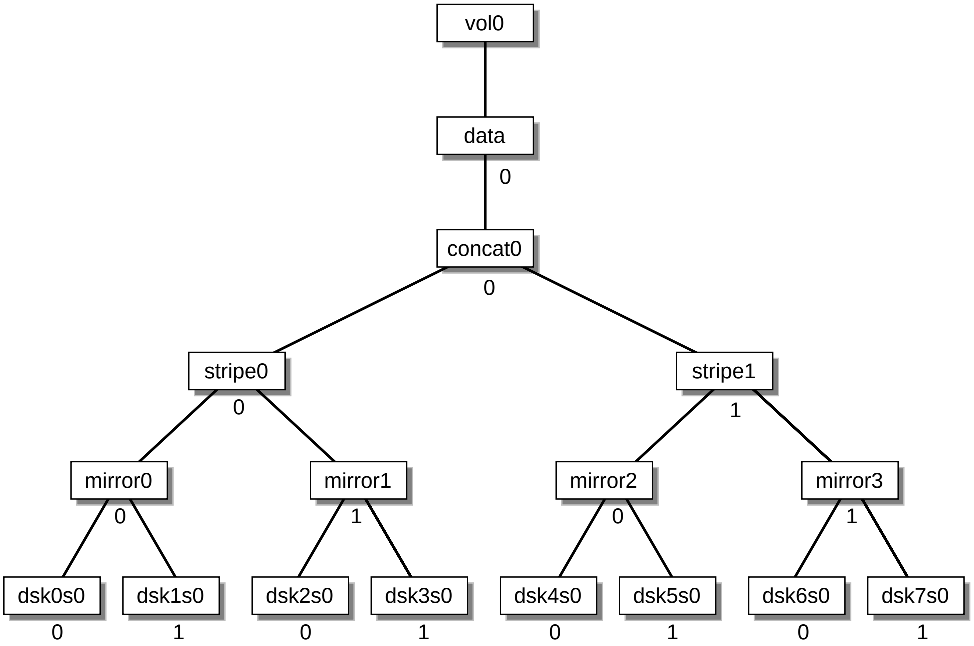


Figure 4-1 XVM Volume with System-Generated Names and Piece Numbers

The first child of a given volume element is considered the *leftmost child* (piece `0`), and the last child is considered the *rightmost child*.

Table 4-4 shows examples of how you can use piece syntax to specify individual volume elements illustrated in Figure 4-1.

Table 4-4 Specifying Volume Elements Using Example Piece Syntax

XVM Object	Example Object Specifications
`concat/concat0`	`vol0/data/0` `vol0/0/0`
`stripe/stripe0`	`vol0/data/0/0` `vol0/0/0/0`
`stripe/stripe1`	`vol0/data/0/1` `vol0/0/0/1`

`mirror/mirror0`	`vol0/data/concat0/stripe0/0`
	`vol0/data/0/0/0`
	`vol0/0/0/0/0`

`mirror/mirror2`	`vol0/data/concat0/stripe1/0`
	`vol0/data/0/1/0`
	`vol0/0/0/1/0`

`slice/dsk6s0`	`vol0/data/concat0/1/mirror3/0` `vol0/0/0/1/1/0`

Object Name Examples

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

Example	**Description**
`vol0`	The object named `vol0`
`unlabeled/sda`	The unlabeled disk named `sda` (`sda` indicates the first SCSI disk discovered, `sdb` indicates the second, and so on)
`mirror/mirror6`	The mirror volume element named `mirror6`
`concat0/0`	The leftmost child of `concat0`
`vol0/data/0`	The child of the `data` subvolume of volume `vol0`

stripe0/freds0	The volume element named freds0 under the stripe stripe0

Regular Expressions

You can use regular expressions when specifying object names. The following characters are recognized and have their standard regular expression meanings as supported through the fnmatch(3) function:

```
*
?
[ ]
```

Regular expressions are limited to the rightmost component of an XVM object path. For example:

Example	**Description**
*	Matches all objects **Note:** At the shell command line, you must protect the * character by enclosing it within quotation marks so that it is interpreted correctly by the shell. For example: # **xvm show '*'** The protection is not required when you are using the xvm CLI interactively.
vol/*	Matches all volumes
slice/*s0	Matches all slices ending in s0
concat0/*	Matches all children of concat0
subvol/log*	Matches all log subvolumes
fred*	Matches all objects beginning with fred
unlabeled/*	Matches all unlabeled disks

For example, to show all objects starting with I:

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

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

A volume element path that consists of an object type keyword is equivalent to specifying the keyword followed by the /* character sequence. For example, the following commands are equivalent:

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

Device Directories and Pathnames

Block devices for XVM volumes used for a host's local volume are contained in the /dev/lxvm directory.

For information on names of objects within XVM volumes, see "Object Names in XVM" on page 71.

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 concat 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 csh 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.

Commands that create or modify volume elements also display the subvolume block-special name that the target volume element belongs to inside of < > symbols. For example,

```
xvm:cluster> slice -all phys1
</dev/cxvm/phys1s0> slice/phys1s0
```

The above indicates that `slice/phys1s0` is the name of the slice created and `/dev/cxvm/phys1s0` is a path to the device that can be opened to gain access to the slice. Strings that appear inside of `< >` characters are treated as comments by the `xvm` CLI and will therefore be ignored if you feed the output of one `xvm` CLI command into another.

Safe Versus Unsafe Commands

The `xvm` CLI commands can be categorized as follows:

- A *safe command* does not affect the address space of the subvolume, such as detaching or deleting all but the last child of a mirror (detaching or deleting the last child is unsafe).

  Safe commands can always be issued regardless of the open state of the subvolume. Mounted subvolumes are always open; however, a subvolume may also be open without being mounted, such as if an application is accessing the raw subvolume.

  Certain commands have a `-safe` option, which will enforce the safe checks even if the subvolume is not open.

- An *unsafe command* is one that will affect the address space of the subvolume that the volume element is under, such as detaching or deleting a child of a concat.

  Unsafe commands can be issued only if the subvolume is not open.

Chapter 5

xvm Administration Commands

This chapter summarizes the xvm command-line interface (CLI) and provides examples of each command. For a full description of each command, see the help text, as described in "Online Help for xvm CLI Commands" on page 68:

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

This chapter discusses the following

- "Physical Volume Commands" on page 81
- "Logical Volume Commands" on page 91

See also Chapter 4, "Overview of the xvm CLI" on page 63.

Physical Volume Commands

This section discusses the following:

- "Changing the Current Domain with the set Command" on page 82
- "Assigning Disks to XVM with the label Command" on page 82
- "Controlling Automatic Probing with the label and set Commands" on page 83
- "Displaying Physical Volumes with the show Command" on page 84
- "Modifying Volume Elements with the change Command" on page 88
- "Probing a Physical Volume with the probe Command" on page 88
- "Regenerating Physvols using the dump Command" on page 89
- "Removing Disks from XVM with the unlabel Command" on page 89
- "Gracefully Transferring Ownership of a Disk or Physvol with the give Command" on page 90
- "Forcibly Transferring Ownership of a Foreign Disk with the steal Command" on page 90

Changing the Current Domain with the set Command

The set command changes the current XVM domain, which can be either cluster or local.

Note: You can set the domain to cluster only if the appropriate services are started; see "CXFS Service Requirements for Cluster Domain" on page 26.

If the current domain is local, the XVM objects you are creating belong to the local node you on which you are executing the xvm command. If the current domain is cluster, the XVM objects you are creating belong to the cluster that the local node belongs to.

The current domain is displayed as part of the xvm prompt, which appears as one of the following:

- Cluster prompt:

  ```
  xvm:cluster>
  ```

- Local prompt:

  ```
  xvm:local>
  ```

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

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

You can also display the current domain from the shell command line. For example:

```
# /sbin/xvm set domain
cluster
```

For more information, see "Local Domain and Cluster Domain" on page 26.

Assigning Disks to XVM with the label Command

The label command assigns a disk to XVM. The label command writes or modifies a physvol label on a disk. In a clustered environment, you can label only the disks that are attached to the local node.

The -name option assigns a name to the physvol. When assigning multiple disks to XVM, the supplied name acts as a prefix for each physvol name, with a unique numeric suffix added.

For example, to label disk 600a0b8000269d1e00004ce048bfb304 as the XVM physvol fred:

```
xvm:cluster> label -name fred /dev/pm/600a0b8000269d1e00004ce048bfb304
```

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.

Caution: Data corruption or system panics can result from labeling disks with partitions that are in use.

For more information and examples, see the full help text.

Controlling Automatic Probing with the `label` and `set` Commands

If you want to disable the default automatic probing of devices under appropriate circumstances, use one of the following methods:

- Use the `-noprobe` option of the `label` command when you label a disk to temporarily disable automatic probing for the current action:

  ```
  xvm:cluster> label -name physvolname -noprobe unlabeled-disk
  ```

- Use the `set` command to disable autoprobing entirely:

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

  Note: See "Use Automatic Probing Wisely" on page 55.

  To reenable autoprobing, use the following command:

  ```
  xvm:cluster> set autoprobe enabled
  ```

Displaying Physical Volumes with the show Command

The show command displays information about XVM objects:

- "All Physvols" on page 84
- "Unlabeled Disks" on page 86
- "Foreign Disks" on page 86
- "Disks with No Physical Connection" on page 87

All Physvols

The following example using the show -extend option displays all the existing physvols and their device paths:

```
xvm:cluster> show -extend phys
phys/_mtx    2339536896 online,cluster,accessible (/dev/disk/by-path/pci-0000:08:03.1-fc-0x201400a0b8269d1e-lun-6)
phys/first   2339536896 online,cluster,accessible (/dev/disk/by-path/pci-0000:08:03.1-fc-0x201400a0b8269d1e-lun-5)
phys/phenix  2339536896 online,cluster,accessible (/dev/disk/by-path/pci-0000:08:03.1-fc-0x201400a0b8269d1e-lun-7)
```

The following example shows all information for the physvol first:

```
xvm:cluster> show -v phys/first
XVM physvol phys/first
=========================
size: 2339536896 blocks  sectorsize: 512 bytes  state: online,cluster,accessible
uuid: 30ac6373-75f7-4518-b4f0-826fd99a0aa6
system physvol:  no
path manager device:  /dev/pm/SGI-TP9700--lun6-600a0b8000269d1e0000c9b14d31a849 on host cxfsxe18
using paths:
/dev/disk/by-path/pci-0000:08:03.1-fc-0x201400a0b8269d1e-lun-5 <sdah 66:16> affinity=none ws
/dev/disk/by-path/pci-0000:08:03.1-fc-0x203400a0b8269d1e-lun-5 <sdy 65:128> affinity=none
/dev/disk/by-path/pci-0000:08:03.1-fc-0x201500a0b8269d1e-lun-5 <sdp 8:240> affinity=none
/dev/disk/by-path/pci-0000:08:03.1-fc-0x203500a0b8269d1e-lun-5 <sdg 8:96> affinity=none [
 GPT label. magic:EFI PART
   GPT Label detected
Disk has the following XVM label:
  Clusterid:  0
  Host Name:  clustard
  Disk Name:  first
  Magic:  0x786c6162 (balx)     Version 2
  Uuid: 30ac6373-75f7-4518-b4f0-826fd99a0aa6
  last update:  Tue Nov 27 17:19:53 2012
  state: 0xa1<online,cluster,accessible> flags: 0x0<idle>
  secbytes:  512
  label area:  8157 blocks starting at disk block 35 (10 used)
  user area:   2339536896 blocks starting at disk block 8192

Physvol Usage:
Start        Length       Name
---------------------------------------------------
0            233953664    slice/firsts0
233953664    233953664    slice/firsts1
467907328    70186080     slice/firsts10
538093408    70186080     slice/firsts11
608279488    70186080     slice/firsts12
678465568    70186080     slice/firsts13
748651648    70186080     slice/firsts14
818837728    70186080     slice/firsts15
889023808    70186080     slice/firsts16
959209888    70186080     slice/firsts17
1029395968   70186080     slice/firsts18
1099582048   70186080     slice/firsts19
1169768128   70186080     slice/firsts20
1239954208   70186080     slice/firsts21
1310140288   70186080     slice/firsts22
1380326368   70186080     slice/firsts23
1450512448   70186080     slice/firsts24
1520698528   70186080     slice/firsts25
1590884608   70186080     slice/firsts26
1661070688   70186080     slice/firsts27
1731256768   70186080     slice/firsts28
1801442848   70186080     slice/firsts29
1871628928   384          (unused)
1871629312   233953664    slice/firsts8
2105582976   233953664    slice/firsts9
2339536640   256          (unused)

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

Unlabeled Disks

The following example displays information about unlabeled disks:

```
xvm:cluster> show -v unlabeled
Unlabeled disk unlabeled/dev/pm/SGI-TP9700--lun6-600a0b8000269d1e0000c9b14d31a849
================================
using paths:
/dev/disk/by-path/pci-0002:00:02.1-fc-0x21000011c61dd97e:0x0000000000000000 <sdbr 68:80> affinity=none ws  ]
/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 <sdy 65:128> affinity=none
```

Foreign Disks

The show command can also display information about foreign disks, which can be useful if you must 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, because it appears as a foreign disk to you. The output of the show command will show the current owner as the Host Name of the physvol.

For example, the following shows that the foreign disk is owned by the node named clux2:

```
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 <sdbs 68:96> affinity=none ws
/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: 47a1faf5-7a9e-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
```

For information on foreign disks, see "Local Domain and Cluster Domain" on page 26. For information on the `steal` command, see "Forcibly Transferring Ownership of a Foreign Disk with the `steal` Command" on page 90.

Note: The `steal` command should be used only when ownership cannot be changed using the `give` command. See "Gracefully Transferring Ownership of a Disk or Physvol with the `give` Command" on page 90.

Disks with No Physical Connection

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 `lcmtst` has no physical connection on this system.

```
xvm:cluster> show vol
vol/c1                            0 online
vol/con1                          0 offline,no physical connection
vol/lmc1                          0 online,no physical connection
vol/lmctst3s2                     0 online
vol/lmctst3s3                     0 online
vol/lmctst3s4                     0 online

xvm:cluster>  show -topology vol/con1
 vol/con1                          0 offline,no physical connection
    subvol/con1/data             400000 offline,pieceoffline
        concat/concat9               400000 offline,tempname,incomplete
            slice/lmctsts1               200000 online
            (empty)                           * *

xvm:cluster> show -topology vol/lmc1
vol/lmc1                          0 online,no physical connection
    subvol/lmc1/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 with the show Command" on page 102.

Modifying Volume Elements with the change Command

You can make the following modifications by using the change command:

- The name of the following objects:
  - Physvol
  - Volume
  - User-defined subvolume
  - Concat
  - Mirror
  - Stripe

  Note: You cannot modify slice names.

- The online/offline or disabled/enabled state of a volume element
- The primary leg of a mirror
- The user ID, group ID, and permission modes of a subvolume
- Whether statistics are gathered or reset for the local node

For example, to enable statistics for physvol pvol0 and the data subvolume of volume fred:

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

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.

For example:

- To probe the unlabeled drives:

```
xvm:cluster> probe unlabeled/*
```

- To probe the physvol named fred:

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

Regenerating Physvols using the dump Command

The dump command outputs the xvm CLI commands that will generate a volume element; you can save the output to a file and use it to regenerate a physvol. For examples of the dump command, see "Reconstructing Volume Elements with the dump Command" on page 104.

Removing Disks from XVM with the unlabel Command

The unlabel command removes 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.

Note: In a clustered environment, you can unlabel only those disks that are attached to the local node.

For example:

- To remove the XVM label from the physvol named phys1:

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

- To forcibly unlabel phys1, first deleting any slices that may exist:

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

Gracefully Transferring Ownership of a Disk or Physvol with the give Command

To gracefully give the ownership of a disk or physvol to another node or another cluster, use the give command.

Note: You cannot give away a disk or physvol that has slices that are part of open subvolumes. For this reason, an attempt to give away a disk will fail during a mirror revive. In general, you must unmount filesystems on XVM volumes that contain the physvol and wait for mirror revives to complete before giving away a physvol.

Giving a disk or physvol away from a domain will result in all slices (and any empty parents that result) being deleted in the nodes of that domain. The configuration information will be retained on the disk. Subvolumes that span disks might go offline if giving a disk away causes slices belonging to that physvol to be removed. The new owner must probe the disk for the configuration information to become visible, either as part of a reboot or if you explicitly execute the probe command on the new owner.

For example, to relinquish ownership of the physvol named fred to the cluster named mycluster:

```
xvm:cluster> give -cluster mycluster fred
```

Note: You can specify any host or cluster name as the new owner of the disk, even if it is not valid when you execute the command.

Forcibly Transferring Ownership of a Foreign Disk with the steal Command

In situations where you cannot use the give command to gracefully change the ownership, such as when the owning host/cluster is unavailable or when disks are moved from one machine to another, it may be necessary to use the steal command to transfer the ownership of a foreign disk to the local node or cluster.

⚠ **Caution:** The steal command will unconditionally reset the owner of a foreign disk. No attempt is made to inform the previous owner that the ownership has changed. If another host or cluster has the physvol instantiated, configuration corruption or data corruption can occur. You should only use steal if you cannot use give.

For example:

- To change the ownership of the specified disk from the cluster `yourcluster` to the current cluster:

```
xvm:cluster> steal -cluster yourcluster foreign//dev/pm/600a0b8000269d1e00004ce048bfb304
```

- To change the ownership of the specified disk from the node `yournode` to the current cluster:

```
xvm:cluster> steal -cluster yourcluster foreign//dev/pm/600a0b8000269d1e00004ce048bfb304
```

- To change the ownership of the specified disk from the node `yournode` to the local node:

```
xvm:cluster> set domain local
xvm:local> steal yournode foreign//dev/pm/600a0b8000269d1e00004ce048bfb304
```

Logical Volume Commands

This section discusses the following:

- "Creating Volume Elements" on page 91
- "Modifying Volume Elements" on page 97
- "Modifying Volume Elements on a Running System" on page 100
- "Displaying Volume Elements with the `show` Command" on page 102
- "Reconstructing Volume Elements with the `dump` Command" on page 104
- "Deleting Volume Elements with the `delete` Command" on page 104
- "Removing Configuration Information with the `reprobe` Command" on page 105

Creating Volume Elements

This section discusses the following commands:

- "`concat`" on page 92
- "`mirror`" on page 92
- "`slice`" on page 94

- "stripe" on page 94
- "subvolume" on page 96
- "volume" on page 97

concat

The concat command creates a volume element that concatenates all of its children 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. If you explicitly name this generated volume, the name is stored in label space and persists across reboots.

The following example concatenates the slices freds0 and wilmas0 into a larger address space. The created concat 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
```

mirror

The mirror command creates a volume element that maintains identical data images on its underlying 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. If you explicitly name this generated volume, the name is stored in label space and persists across reboots.

When you create a mirror that has more than one leg, a message is written to the system log, 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 system log when this process is complete. For large mirror components, this may take a long time. You cannot halt a mirror revive after it has begun except by detaching all but one of the legs of the mirror.

If the revive fails, a message will be written to the system console as well as to the system log. For more information, see "Mirror Revives" on page 173.

When you create a mirror, you can define a read policy and a primary leg for the mirror. You can also specify that the mirror does not need to be revived when it is created. Alternately, you can specify that the mirror will never need to be revived; this is an option that is useful when you are mirroring a scratch filesystem. These features are described in "Creating Mirrors" on page 41.

Note: 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.

For example:

- To create a mirror whose legs are the slices `freds0` and `wilmas0` and associate the mirror with a volume named `mirvol`:

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

- To create a mirror with legs `slice/freds0` and `slice/wilmas0` and volume name `newmirvol`:

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

  In this example, a revive will not be initiated when the mirror is created.

- To create an empty mirror with a sequential read policy:

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

  This command creates a mirror with a system-generated name that is contained in a volume with a system-generated name. To make this mirror usable, you must explicitly add legs using the `attach` command.

- To create a two-leg mirror with a primary leg named `freds0`:

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

  All reads will be directed to `freds0`, with writes going to both `freds0` and `wilmas0`. This command creates a mirror with a system-generated name that is contained in a volume with a system-generated name.

slice

The slice command creates slices from specified block ranges of physvols.

For example:

- To create one slice covering the whole usable space of the physvol phys1:

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

- To create four equal-sized slices covering the physvol phys1:

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

- To create a slice starting with block 5,000 with a length of 100,000 blocks:

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

- To divide the 100,000-block chunk beginning at block 5,000 into four equal-sized slices:

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

stripe

The stripe command creates a volume element that distributes 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. If you explicitly name this generated volume, the name is stored in label space and persists across reboots.

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

By default, a stripe unit must be a multiple of 32 512-byte blocks. You can remove this restriction with the -noalign flag.

The actual size of the stripe depends on the size of the stripe unit size and 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 because the same amount of space in each volume element that makes up the slice is used.

Following is the general formula for determining the stripe size, where *stripe_width* is the number of volume elements that make up the stripe (using integer arithmetic):

stripe_size = (*smallest_stripe_member* / *stripe_unit*) * *stripe_unit* * *stripe_width*

You can view the stripe unit of an existing stripe using either of the following commands (where *stripename* is the name of the existing stripe):

```
xvm:cluster> show -extend stripename
xvm:cluster> show -v stripename
```

If your non-ALUA RAID configuration requires that you spread I/O across controllers, you must have a complete `/etc/failover2.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 `/etc/failover2.conf` file to specify a primary path. For information on configuration of failover for storage devices, see your SGI support provider. Information on the `/etc/failover2.conf` file can also be found in the `/etc/failover2.conf` file itself. See Chapter 6, "XVM Path Failover" on page 107.

For information on configuring stripes that span two host bus adapters, see Chapter 6, "XVM Path Failover" on page 107.

For example:

- To stripe the slices `freds0` and `wilmas0` and associate the volume `stripedvol` with the slices:

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

- To stripe the `mirror0` and `mirror1` using a stripe-unit size of 512 512–byte blocks:

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

- To create an empty stripe with room for four slices:

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

Note: Four volume elements must be attached to the stripe before it will come online.

subvolume

The subvolume command creates a subvolume and optionally attaches a specified concat, mirror, stripe, or slice 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. If you explicitly name this generated volume, the name is stored in label space and persists across reboots.

You can create a subvolume of a system-defined type (data, log, or rt) or you can create a subvolume of a user-defined type. There cannot be more than one subvolume with the same type under a volume. For example, there can be only one data subvolume under a volume. See "Subvolume" on page 15.

For example:

- To create a log subvolume, attach concat0 to it, and associate the volume myvol with the subvolume:

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

- To create a subvolume and attach concat0 to it, setting the user ID and permission mode of the block and character special files corresponding to the subvolume:

```
xvm:cluster> subvolume -tempname -uid 1823 -mode 0664 concat0
```

- To create a subvolume with a user-defined type of 100:

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

volume

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

For example, to create an empty volume named fred:

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

Modifying Volume Elements

Caution: You should unmount a filesystem before making changes to it via XVM if those changes will affect address space. See "Unmount Filesystems Before Making XVM Topology Changes" on page 60.

This section discusses the following commands:

- "attach" on page 97
- "change" on page 98
- "detach" on page 99
- "remake" on page 99

attach

The attach command attaches an existing volume element to another existing volume element. For information on the restrictions that XVM imposes on attachments, see "Attaching Volume Elements" on page 38.

You can explicitly specify where to attach a volume element; if you do not, the source volume element will be attached to the first (leftmost) hole in the XVM topology tree for the target volume element. If there are no holes in the tree, the source volume element will be appended to the end (that is, to the rightmost position).

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
```

change

The `change` command modifies the attributes of a physvol or volume element that you have previously defined. You can change a variety of attributes, 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 reenable a volume element.

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

For example:

- To enable statistics for physvol `pvol0` and the `data` subvolume of `vol/fred`:

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

- To reset statistics for all objects that have statistics enabled:

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

- To change the name of volume `fo1s0` to `myvol`:

  ```
  xvm:cluster> xvm change name myvol vol/fo1s0
  ```

detach

The detach command detaches a volume element from its parent. When you detach a volume element, a new volume 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. If you explicitly name this generated volume, the name is stored in label space and persists across reboots. 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 leg of an open mirror from that mirror because this will cause the mirror to go offline.

An element of an open subvolume can only be detached if this will not cause the subvolume to go offline. The only element that can be detached without putting the subvolume offline is a mirror leg that is not the last leg of that mirror. 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.

For example:

- To detach the volume element concat0 from its parent and associate concat0 with the volume fred:

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

- To detach 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
  ```

- To detach concat0 but ensure 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
  ```

remake

The remake command reorganizes volume elements by collapsing holes in its XVM topology tree or by rearranging its children. You can use a single remake command as a convenient alternative to executing a series of attach and detach commands.

When you rearrange the children, you can specify one of the following rearrangement methods:

Method	Description
swap	Swaps the positions of two children under a volume element
reorder	Reorders all of the children under a volume element

If you do not specify a method, XVM will collapse any holes in the XVM topology tree for the volume element.

For example:

- To reorganize the children of concat0, swapping pieces numbered 0 and 1:

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

  For more information, see "Specifying Objects by Piece Number" on page 74.

- To reorganize concat0, reversing the order of its three children:

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

- To collapse any holes in the XVM topology tree for concat0:

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

Modifying Volume Elements on a Running System

This section discusses the following commands:

- "insert" on page 100
- "collapse" on page 101

insert

The insert command inserts a mirror or a concat 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, which 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
</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
```

collapse

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

You can use the `collapse` command 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                          0 online,tempname
    subvol/vol9/data            5926338 online
        concat/concat1              5926338 online,tempname
            slice/pebbless0             2963169 online
            slice/bettys0               2963169 online
```

Displaying Volume Elements with the show Command

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

For example:

- To show name, size, and state information for the object named concat0:

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

- To show all XVM slices:

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

- To show the names of all XVM volumes:

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

- To show the names of all unlabeled disks:

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

- To show statistics for the physvol named `fred`:

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

- To show the topology of the a volume and display the stripe unit size of the stripes in the volume as well as the name and device path of the physvols associated with any slices in the volume:

```
xvm:cluster> show -topology -extend volume
```

For example:

```
xvm:cluster> show -topology vol/drives0
     subvol/drives0/data       111406464 online,reviving,accessible
         mirror/mirror0            111406464
online,tempname,reviving:1%,accessible
             concat/concat0            111406464 online,tempname,accessible
                 slice/drives0              55703232 online,accessible
                 slice/drives1              55703232 online,accessible
             stripe/stripe1            167109504
online,tempname,reviving,accessible
                 slice/drives2              55703232 online,accessible
                 slice/drives3              55703232 online,accessible
                 slice/drives4              55703232 online,accessible
```

After the revive completes:

```
xvm:local> show -top vol/drives0
vol/drives0                        0 online,accessible
     subvol/drives0/data       111406464 online,accessible
         mirror/mirror0            111406464 online,tempname,accessible
             concat/concat0            111406464 online,tempname,accessible
                 slice/drives0              55703232 online,accessible
                 slice/drives1              55703232 online,accessible
             stripe/stripe1            167109504 online,tempname,accessible
                 slice/drives2              55703232 online,accessible
                 slice/drives3              55703232 online,accessible
                 slice/drives4              55703232 online,accessible
```

Reconstructing Volume Elements with the dump Command

The dump command outputs XVM configuration commands that you can use to reconstruct volume elements and physvols.

Note: You must explicitly dump the physvol separately from a volume element topology tree.

For example:

- To dump a one-line creation command for the volume named fred:

  ```
  xvm:cluster> dump vol/fred
  volume -volname fred -uuid e5570d23-a491-4f03-9e4c-0a047ae100ff
  ```

- To dump information for all volume elements under each volume:

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

- To dump the contents of all physvol and volume topology trees to the file foo:

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

Deleting Volume Elements with 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 the following mutually exclusive options that override this restriction:

- The -all option deletes a volume element and all volume elements below it
- The -nonslice option deletes a volume element and all non-slice volume elements below it, detaching and keeping the slices

Removing Configuration Information with the reprobe Command

If a disk becomes inaccessible and must 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 to remove previous configuration information from the kernel. The following example illustrates a situation where you can use the reprobe command. In this example, there is a volume configured with a mirror:

```
vol/test
   mirror/mirror0
      slice/lun9s0
      slice/lun8s0
```

If lun8 begins to generate I/O errors, you may need to replace the disk. If you try to detach slice/lun8s0, however, the system will be unable to write the update to the disk and will not perform the detach. In this case, you can execute the following command:

```
xvm:cluster> reprobe lun8
```

This removes all the places where lun8 is used in a configuration, even though you cannot write to the lun8 physvol. After executing this command, you can attach a new slice to mirror/mirror0.

You can also use the reprobe command to recover from an inconsistent configuration that could arise when you use the steal command to take a disk from a running system. In this case, the disk label may not match the configuration information in the kernel and the configuration may show the disk as both owned and foreign. The reprobe command will remove the configuration information for the physvol from the kernel.

The reprobe command will remove configuration information for the indicated physvol only if the disk is inaccessible or if the configuration on the disk label does not match the current information in the kernel.

The following example removes the configuration information in the kernel for the XVM physvol named fred if fred is not available or if the configuration information in the disk label of fred does not match the configuration information in the kernel:

```
xvm:cluster> reprobe fred
```

Chapter 6

XVM Path Failover

This chapter discusses the following:

XVM Path Failover Concepts

This section discusses the following:

Purpose of XVM Failover

XVM path failover creates an infrastructure for the definition and management of multiple paths to a single disk device or logical unit (LUN). XVM path failover can be used on any supported underlying storage, such as RAID devices.

Figure 6-1 shows a simple case of a host computer connected through a fabric to a RAID that offers four LUNs.

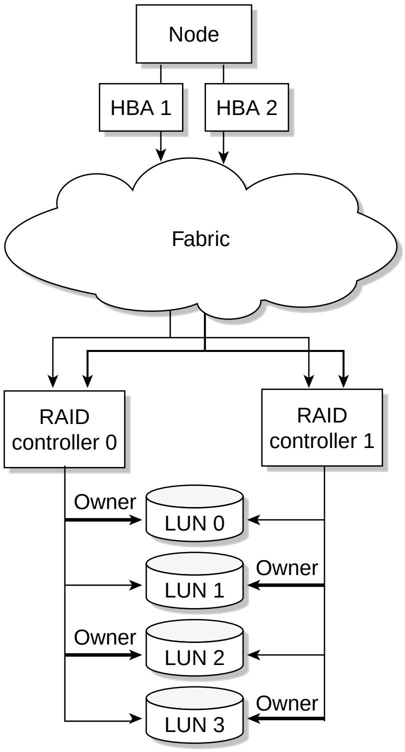


Figure 6-1 Disk Paths

XVM selects a path for I/O from the host through the fabric to the RAID, using the least loaded HBA at the host end and a particular controller at the RAID end. You can improve I/O performance by configuring XVM to select the RAID controllers for each LUN so that the I/O load is distributed across them.

All LUNs are visible from each RAID controller, therefore each LUN can be accessed by each path. However, in this example controller 0 is preferred for LUN 0 and LUN 2 while controller 1 is preferred for LUN 1 and LUN 3. Unnecessary switching between RAID controllers to access a LUN can degrade performance considerably.

Controller Configuration

If the RAID supports the asymmetric logical unit access (ALUA) feature, XVM will automatically read the RAID's settings for preferred controller for each LUN. If ALUA is not supported, you should add a line with the keyword `preferred` to the `/etc/failover2.conf` file to configure the preferred controller.

If XVM detects an I/O error, it will attempt path failover and select another path. XVM must know the controller used by each path so that it can choose a path through the same controller if possible.

If the RAID supports the ALUA feature, XVM will automatically read each path's controller connection from the RAID's configuration. If ALUA is not supported, you should add a line for each path in the `/etc/failover2.conf` file that has an affinity setting such as `affinity=1`, where an affinity value is arbitrarily associated with a controller.

Note: For ALUA RAID, you do not use the `affinity` or `preferred` keywords.

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 (which would result in poor disk performance).

For a RAID supporting the ALUA feature, configuration information provided in the `/etc/failover2.conf` file overrides the equivalent configuration read from the RAID.

Note: In a CXFS cluster configuration, be sure that all of your `/etc/failover2.conf` lines for non-ALUA RAIDs correctly identify the path's affinity and that they all agree on the preferred affinity value.

Non-ALUA RAID devices must be set to `SGIAVT` mode.

HBA Configuration

Note: The ALUA feature of a RAID has no effect on HBA configuration.

If the `preferred` keyword is used in a line of `/etc/failover2.conf` for a non-ALUA RAID, then the preferred affinity and therefore a preferred controller is selected.

Note: For ALUA RAID, you do not use the `preferred` keyword. Instead, set the controller preference in the RAID software.

If the dynamic selection of HBAs for each path is not satisfactory, you may use the `priority=1` setting in the line in the `/etc/failover2.conf` file. There can be a path marked `priority=1` for each controller of a LUN.

Other paths may be marked `priority=2`, `priority=3`, and so on up to `priority=7`. Paths with higher numbers are used only if lower numbers are not available. Paths without the priority value are used last.

HBA Configuration Differences for SGI UV Systems

The SGI UV system consists of a number of compute and central-memory nodes interconnected by a NUMAlink ® network. Each node could have an HBA that originates paths to the RAID. The data for a particular I/O is on some node and XVM selects an HBA that is close to the data (according to the NUMAlink) in order to minimize use of NUMAlink bandwidth. On an SGI UV system, the selection of an I/O path is made by considering both the proximity of the data buffer to the HBA and the current I/O load on the HBA. You can optionally use the `priority` tag to provide an additional bias toward a particular path.

RAID Units and XVM Failover V2

This section discusses the following:

- "TP9100 and RM610/660" on page 111
- "TP9300, TP9500, TP9700, and S330" on page 111

TP9100 and RM610/660

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 RAIDs behave and have the same characteristics as the `SGIAVT` mode discussed below.

TP9100 1 GB and 2 GB `SGIAVT` mode requires that the array is set to `multitid`.

TP9300, TP9500, TP9700, and S330

The TP9300, TP9500, and TP9700 RAID `SGIAVT` mode 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 node 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.

XVM failover requires that you configure TP9300, TP9500, TP9700, and S330 RAID units with `SGIAVT` host type and the 06.12.18.*xx* code or later be installed.

TP9700 use of `SGIAVT` requires that 06.15.17*xx*. code or later be installed.

Configuring the `/etc/failover2.conf` File

This section discusses the following steps to configure the `/etc/failover2.conf` file:

- "Show the Available Unlabeled Paths" on page 112
- "Create a Preliminary `/etc/failover2.conf` File" on page 114
- "Edit the `/etc/failover2.conf` File" on page 114

Show the Available Unlabeled Paths

Use the XVM `show unlabeled` command in both the local and (if using CXFS) cluster domains to see the disks available to XVM. For example, the following shows that there are 5 unlabeled LUNs:

```
# xvm show unlabeled
unlabeled/dev/pm/SGI-TP9700--lun0-600a0b8000269d1e0000c9b14d31a849          * *
unlabeled/dev/pm/SGI-TP9700--lun1-600a0b8000269d1e0000c9b14d31a849          * *
unlabeled/dev/pm/SGI-TP9700--lun2-600a0b8000269d1e0000c9b14d31a849          * *
unlabeled/dev/pm/SGI-TP9700--lun3-600a0b8000269d1e0000c9b14d31a849          * *
unlabeled/dev/pm/SGI-TP9700--lun4-600a0b8000269d1e0000c9b14d31a849          * *
```

To see more information about a specific LUN, use the `show` command to display verbose information:

```
# xvm show -v /dev/pm/pathname
```

For example:

```
# xvm show -v /dev/pm/SGI-TP9700--lun0-600a0b8000269d1e0000c9b14d31a849
Unlabeled disk unlabeled/dev/pm/SGI-TP9700--lun0-600a0b8000269d1e0000c9b14d31a849
================================
using paths:
/dev/disk/by-path/pci-0000:08:03.0-fc-0x21000011c61dd97e-lun-0 <sdbm68:0> affinity=none ws
/dev/disk/by-path/pci-0000:08:03.1-fc-0x21000011c61dd97e-lun-0 <sdq65:0> affinity=none ws
/dev/disk/by-path/pci-0000:08:03.0-fc-0x22000011c61dd97e-lun-0 <sdbf67:144> affinity=none ws
/dev/disk/by-path/pci-0000:08:03.1-fc-0x22000011c61dd97e-lun-0 <sdp8:240> affinity=none ws
```

The above output shows that there are four paths to the LUN and, because `affinity=none`, that the affinities have not been configured. An ALUA LUN is configured automatically, therefore this output indicates that this is a non-ALUA LUN and therefore you should configure the path failover in the `/etc/failover2.conf` file.

The four lines showing paths can be used as lines of the `/etc/failover2.conf` file, with configuration for each path added to its line. For example:

```
/dev/disk/by-path/pci-0000:08:03.0-fc-0x21000011c61dd97e-lun-0 <sdbm68:0> affinity=1
/dev/disk/by-path/pci-0000:08:03.1-fc-0x21000011c61dd97e-lun-0 <sdq65:0> affinity=1
/dev/disk/by-path/pci-0000:08:03.0-fc-0x22000011c61dd97e-lun-0 <sdbf67:144> affinity=2 preferred
/dev/disk/by-path/pci-0000:08:03.1-fc-0x22000011c61dd97e-lun-0 <sdp8:240> affinity=2
```

In another example, suppose that the following LUN is not already represented in the /etc/failover2.conf file and it produces the following show output:

```
# xvm show -v /dev/pm/SGI-TP9700--lun1-600a0b8000269d1e0000c9b14d31a849
Unlabeled disk unlabeled/dev/pm/SGI-TP9700--lun1-600a0b8000269d1e0000c9b14d31a849
=================================
using paths:
/dev/disk/by-path/pci-0000:08:03.1-fc-0x20360080e524a0d2-lun-1 <sdcu70:32> <ALUA opt> affinity=0 ws
/dev/disk/by-path/pci-0000:08:03.1-fc-0x20460080e524a0d2-lun-1 <sdcq69:224> <ALUA opt> affinity=0 ws
/dev/disk/by-path/pci-0000:08:03.1-fc-0x20370080e524a0d2-lun-1 <sdcm69:160> <ALUA nonopt pref> affinity=1
/dev/disk/by-path/pci-0000:08:03.1-fc-0x20470080e524a0d2-lun-1 <sdci69:96> <ALUA nonopt pref> affinity=1
/dev/disk/by-path/pci-0000:08:03.0-fc-0x20470080e524a0d2-lun-1 <sdam66:96> <ALUA nonopt pref> affinity=1
/dev/disk/by-path/pci-0000:08:03.0-fc-0x20370080e524a0d2-lun-1 <sdaq66:160> <ALUA nonopt pref> affinity=1
/dev/disk/by-path/pci-0000:08:03.0-fc-0x20360080e524a0d2-lun-1 <sday67:32> <ALUA opt> affinity=0 ws
/dev/disk/by-path/pci-0000:08:03.0-fc-0x20460080e524a0d2-lun-1 <sdau66:224> <ALUA opt> affinity=0 ws
```

The above output indicates that the paths with affinity=0 are the *working set* (ws), which is set of paths that are currently used for I/O. Activity is only shown for the current working set. A working set starts out as all of the paths of the preferred affinity, but paths can be removed if they get errors, and the whole working set can be changed to the paths of another controller by a failover to another affinity.

In this case, the path reported for the LUN contains the ALUA tag, which indicates that the RAID has the ALUA feature set. The affinity= values are integers based on the RAID's target port group number. The following tags provide additional information about the ALUA LUN:

Tag	**State**
nonopt	This path is not optimized. That is, it runs through the RAID controller that provides slower performance.
opt	This path is optimized. That is, it runs through the RAID controller that currently gives it the best performance.
pref	This affinity is configured as preferred in the RAID.

For ALUA RAIDs, XVM will use these configurations to select preferred paths if there are no lines for this LUN in the /etc/failover2.conf file. If you require changes, you should make them via the RAID software (recommended) or else override them in the /etc/failover2.conf file.

For non-ALUA RAIDs that have no /etc/failover2.conf file entries, an arbitrary path is chosen by path manager for I/O.

Create a Preliminary /etc/failover2.conf File

The path report lines from the following xvm command are usable as a preliminary /etc/failover2.conf file:

```
#  xvm show -v unlabeled
```

The persistent pathname is used by xvm to associate the other attributes on the line with the path. Other parts of the show output should be removed, for instance by using the following command line:

```
# xvm show -v unlabeled | grep -e affinity > /etc/failover2.conf
```

In this preliminary file, non-ALUA LUNs will have a affinity=none setting. (ALUA LUNs will automatically have an affinity=*integer* setting that is based on the RAID configuration, therefore it is not necessary to include them in the /etc/failover2.conf file.

Note: Normally, if changes for an ALUA LUN are required, you should make them via the RAID. However, you can override the settings via the /etc/failover2.conf by including the path and including the affinity=*integer* and preferred keywords, just as for non-ALUA LUNs. (If the nonopt, opt, and pref keywords are present in the /etc/failover2.conf, they are ignored). See:

- "Do Not Override ALUA RAID Settings via /etc/failover2.conf" on page 59
- "Set Appropriate affinity Values for Non-ALUA LUNs" on page 115
- "Set the preferred Path for Each Non-ALUA LUN" on page 117

Edit the /etc/failover2.conf File

This section discusses the following:

- "Set Appropriate affinity Values for Non-ALUA LUNs" on page 115
- "Set the preferred Path for Each Non-ALUA LUN" on page 117
- "Select HBA Usage for Each LUN" on page 117
- "Example /etc/failover2.conf File Excerpt" on page 119

Set Appropriate `affinity` Values for Non-ALUA LUNs

The following rules apply to the `/etc/failover2.conf` file for non-ALUA LUNs:

- The order of the paths listed in the `/etc/failover2.conf` file is not significant.
- The valid range of affinity values is `affinity=0` (lowest and default) through `affinity=15` (highest). The path used starts with the `affinity` of the currently used path and increases from there. Paths with the same `affinity` number are all tried before XVM failover moves to the next highest `affinity` number. For example, if the currently used path is `affinity=2`, all `affinity=2` paths are tried, then all `affinity=3`, then all `affinity=4`, and so on; after `affinity=15`, failover wraps back to `affinity=0` and starts over.

  By default, every path has a default of `affinity=0`.
- A path that fails will be removed from the working set, leaving the rest of the working set to continue handling the I/O. If there are no more paths in the working set, then failover to the next `affinity` will occur.
- Strings within `< >` characters in the file are considered comments and will be ignored.
- If a path is not actually present, none of the attributes assigned to it in the `/etc/failover2.conf` file will take effect. If a path is not available, XVM cannot determine the LUN to which an `/etc/failover2.conf` entry is referring.

  Note: If a preferred path ceases to be available, you must edit the `/etc/failover2.conf` file so that path selection takes place correctly for the remaining paths. You can then inject the changes to into the kernel by using the following command:

  ```
  xvm:cluster> foconfig -init
  ```
- The `affinity` values for a particular RAID controller should be identical on every node in the CXFS cluster.

  Note: A given `affinity` group must all go to the same RAID controller.

Usually, you want to configure all paths to the same controller with the same `affinity` value and thus only two `affinity` values are used. 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. Consider the following:

- Because the default is affinity=0, it would be sufficient to include entries only for those paths that are a non-zero affinity. It would also be sufficient to include a preferred entry for the preferred path only. However, SGI recommends including definitions for all paths.
- You may find it useful to specify values starting with affinity=1 and specify a nonzero value for all paths. This makes it easy to detect those paths that have not yet been configured because they are assigned the default of affinity=0. For example, if you added a new HBA but forgot to add its paths to the /etc/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.

⚠ **Caution: For non-ALUA RAID:** If you explicitly define values other than affinity=0 in the /etc/failover2.conf file but you do not define every path, those undefined paths (which by default have affinity=0) will use an unspecified controller, which can have negative results. For example, suppose you explicitly define affinity=1 and affinity=2 and the currently active affinity is affinity=2 when there is a failover. XVM will fail over to a path that has affinity=0, which would be one of the undefined paths; this path might use the failed RAID controller. If there are multiple unspecified paths with the default affinity=0, those paths might use different RAID controllers, which would be a performance issue. To avoid these problems, you should explicitly define every path in the /etc/failover2.conf file.

Following is a simple strategy that works well for most sites:

- Set to affinity=1 for all paths to a LUN that go through controller A
- Set to affinity=2 for all paths to a LUN that go through controller B

Note:

For SGI InfiniteStorage platforms, the WWN of a path to controller A always uses an even number in the fourth digit of the hexadecimal name. For example:

```
0x202400a0b8119204
     ^
```

The WWN of a path to controller B always uses an odd number in the fourth digit of the hexadecimal name. For example:

```
0x201500a0b8119204
     ^
```

Set the `preferred` Path for Each Non-ALUA LUN

For a non-ALUA RAID, add the `preferred` key to one path for each LUN. To determine which path to identify as `preferred`, consult the TPSSM GUI or the **RAID Array** profile.

For an ALUA RAID, the ALUA feature provides the affinities and a preferred affinity.

Select HBA Usage for Each LUN

You can set HBA usage for a LUN by adding the `priority=`*n* setting to some or all of the `/etc/failover2.conf` lines for that LUN. The value of the priority ranges from `1` (highest priority) to `7`; lines without a `priority=`*n* setting are given the lowest priority, the same as `priority=0`.

If several paths with the same `affinity` have the same `priority`, then one of them will be selected arbitrarily if these are the highest `priority` paths available in the `affinity`.

Because the `affinity` and `preferred` settings are provided automatically by an ALUA LUN, the only need for an `/etc/failover2.conf` file is to bias the selection of an HBA by providing `priority` values; in this case, you could add the high-priority paths to the file.

Do the following:

1. Display the paths of an unlabeled ALUA LUN:

```
petrel:~ # xvm show -v /dev/pm/60080e500024a0d20000046f5006d2c2
Unlabeled disk unlabeled/dev/pm/60080e500024a0d20000046f5006d2c2
================================
using paths:
/dev/disk/by-path/pci-0000:08:03.1-fc-0x20360080e524a0d2-lun-2 <sdcv 70:48> <ALUA opt pref> affinity=0 ws
/dev/disk/by-path/pci-0000:08:03.1-fc-0x20370080e524a0d2-lun-2 <sdcn 69:176> <ALUA nonopt> affinity=1
/dev/disk/by-path/pci-0000:08:03.1-fc-0x20460080e524a0d2-lun-2 <sdcr 69:240> <ALUA opt pref> affinity=0 ws
/dev/disk/by-path/pci-0000:08:03.1-fc-0x20470080e524a0d2-lun-2 <sdcj 69:112> <ALUA nonopt> affinity=1
/dev/disk/by-path/pci-0000:08:03.0-fc-0x20470080e524a0d2-lun-2 <sdan 66:112> <ALUA nonopt> affinity=1
/dev/disk/by-path/pci-0000:08:03.0-fc-0x20370080e524a0d2-lun-2 <sdar 66:176> <ALUA nonopt> affinity=1
/dev/disk/by-path/pci-0000:08:03.0-fc-0x20360080e524a0d2-lun-2 <sdaz 67:48> <ALUA opt pref> affinity=0 ws
/dev/disk/by-path/pci-0000:08:03.0-fc-0x20460080e524a0d2-lun-2 <sdav 66:240> <ALUA opt pref> affinity=0 ws
```

2. Choose a path of each `affinity` that uses the same HBA and add just those paths to the `/etc/failover2.conf` file, giving each a high priority value:

```
/dev/disk/by-path/pci-0000:08:03.1-fc-0x20370080e524a0d2-lun-2  priority=1
/dev/disk/by-path/pci-0000:08:03.1-fc-0x20460080e524a0d2-lun-2  priority=1
```

3. Label and show the LUN to see the priority selection take effect (line breaks shown for readability):

```
petrel:~ # xvm label -name clalua2 /dev/pm/60080e500024a0d20000046f5006d2c2
clalua2
Performing automatic probe for alternate paths.
Performing automatic path switch to preferred path for phys/clalua2.

petrel:~ # xvm show -v clalua2
XVM physvol phys/clalua2
=========================
size: 1142784000 blocks  sectorsize: 512 bytes  state: online,local,accessible
uuid: 0f605448-d84b-4029-962c-00e45970bfc2
system physvol:  no
path manager device:  /dev/pm/60080e500024a0d20000046f5006d2c2 on host petrel
using paths:
/dev/disk/by-path/pci-0000:08:03.1-fc-0x20360080e524a0d2-lun-2 <sdcv 70:48> <ALUA opt pref> affinity=0 ws
/dev/disk/by-path/pci-0000:08:03.1-fc-0x20370080e524a0d2-lun-2 <sdcn 69:176> <ALUA nonopt> affinity=1
    priority=1
/dev/disk/by-path/pci-0000:08:03.1-fc-0x20460080e524a0d2-lun-2 <sdcr 69:240> <ALUA opt pref> affinity=0 ws
    priority=1
/dev/disk/by-path/pci-0000:08:03.1-fc-0x20470080e524a0d2-lun-2 <sdcj 69:112> <ALUA nonopt> affinity=1
```

```
/dev/disk/by-path/pci-0000:08:03.0-fc-0x20470080e524a0d2-lun-2 <sdan 66:112> <ALUA nonopt> affinity=1
/dev/disk/by-path/pci-0000:08:03.0-fc-0x20370080e524a0d2-lun-2 <sdar 66:176> <ALUA nonopt> affinity=1
/dev/disk/by-path/pci-0000:08:03.0-fc-0x20360080e524a0d2-lun-2 <sdaz 67:48> <ALUA opt pref> affinity=0 ws
/dev/disk/by-path/pci-0000:08:03.0-fc-0x20460080e524a0d2-lun-2 <sdav 66:240> <ALUA opt pref> affinity=0 ws
```

Example `/etc/failover2.conf` File Excerpt

Suppose the following:

- There is one PCI card with HBA two ports (highlighting the name differences in bold):

  ```
  pci-0000:04:00.0
  pci-0000:04:00.1
                 ^
  ```

- There are two RAID controllers (controller A and controller B), each with two ports. Each controller port has a unique world wide number (WWN) that is part of the persistent pathname, as follows (highlighting the name differences in bold):

  – Controller A ports:

  ```
  0x202400a0b8119204
  0x204400a0b8119204
      ^^
  ```

  – Controller B ports:

  ```
  0x201500a0b8119204
  0x203500a0b8119204
      ^^
  ```

  Note: These details may vary depending on the RAID vendor.

- Controller B is the preferred controller for LUN 1.

In this situation, the LUN has eight paths (via two PCI cards, two RAID controllers, and two ports on the controllers). You want to group the paths so that all paths to controller B will be tried before any of the paths to controller A. To do this, use the following settings in the `/etc/failover2.conf` file:

- Two `affinity` groups for `lun1`:

  - `affinity=1` (highlighted in green) for the paths that go to controller B (`0x2015`... and `0x2035`...)
  - `affinity=2` for the paths that go to controller A (`0x2024`... and `0x2044`...)
- A `preferred` path that goes to one of the ports in controller B (highlighted in orange)

Figure 6-2 depicts the paths, highlighting the four `affinity=1` paths in green and the one `preferred` path in orange. Figure 6-3 shows the corresponding portion of the `/etc/failover2.conf` file.

Given the above, failover will exhaust all paths to `lun1` from controller B (with `affinity=1` and the `preferred` path) before moving paths that use controller A (with `affinity=2`).

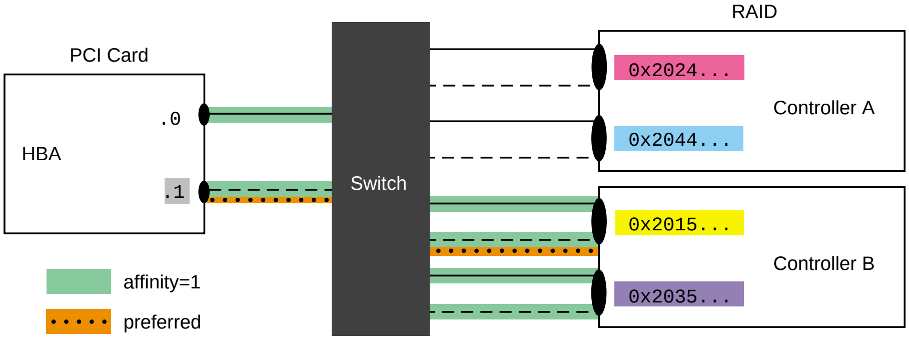
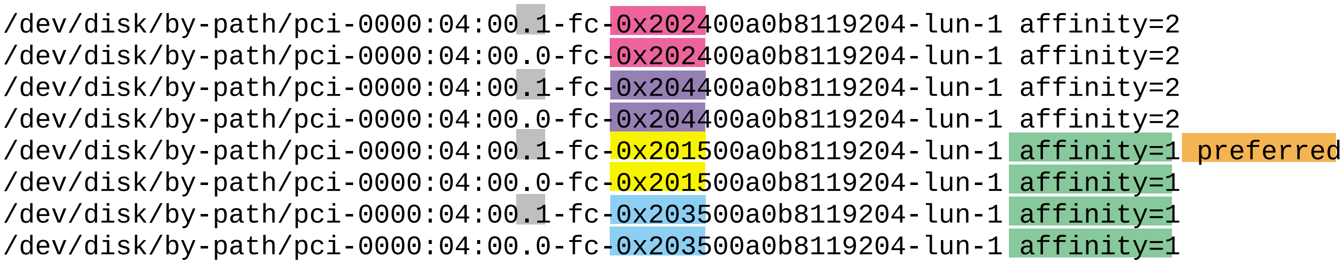


Figure 6-2 Paths



```
/dev/disk/by-path/pci-0000:04:00.1-fc-0x202400a0b8119204-lun-1 affinity=2
/dev/disk/by-path/pci-0000:04:00.0-fc-0x202400a0b8119204-lun-1 affinity=2
/dev/disk/by-path/pci-0000:04:00.1-fc-0x204400a0b8119204-lun-1 affinity=2
/dev/disk/by-path/pci-0000:04:00.0-fc-0x204400a0b8119204-lun-1 affinity=2
/dev/disk/by-path/pci-0000:04:00.1-fc-0x201500a0b8119204-lun-1 affinity=1 preferred
/dev/disk/by-path/pci-0000:04:00.0-fc-0x201500a0b8119204-lun-1 affinity=1
/dev/disk/by-path/pci-0000:04:00.1-fc-0x203500a0b8119204-lun-1 affinity=1
/dev/disk/by-path/pci-0000:04:00.0-fc-0x203500a0b8119204-lun-1 affinity=1
```

Figure 6-3 `/etc/failover2.conf` file with Two Affinity Groups and a Preferred Path

Label the Paths

Assign the disks to XVM by using the `label` command. Paths to the same LUN are detected automatically by the path-manager feature. If you run the `label` command and exit the `xvm` CLI, the XVM autoprobe feature can probe for additional paths (see "Controlling Automatic Probing with the `label` and `set` Commands" on page 83). In a CXFS environment, the CXFS `reprobe` script is run automatically in order to discover new alternate paths.

Note: If storage that was labeled in one domain is later given to a node or cluster in another domain, you must explicitly run an XVM `probe` command in order for XVM to recognize the disk as an XVM disk.

Pushing the Contents of the `/etc/failover2.conf` File to the Kernel

The contents of the `/etc/failover2.conf` file is pushed to the kernel upon reboot or when you execute the following command:

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

Note: To force a change of affinity immediately, use the `foswitch` command as described in "Setting All LUNs to the Preferred Path" on page 122.

You should examine the messages produced by the `foconfig` command to find potential errors in the `/etc/failover2.conf` file. For more messages, use the `-verbose` option.

The XVM `foconfig` command allows you to override the default `/etc/failover2.conf` filename with the `-f` option. The following command pushes to the kernel the contents of the file `myfailover2.conf` in the current working directory:

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

Manually Changing Physvol Paths

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 working set on a running system, without rebooting. This section discusses the following:

- "Setting All LUNs to the Preferred Path" on page 122
- "Switching to a New Device" on page 122
- "Setting a New Affinity" on page 122

Setting All LUNs to the Preferred Path

To set all physvols to their `preferred` path, enter the following:

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

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

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

Switching to a New Device

The following command requests that a path that has been removed from a working set because of I/O errors be added back to the working set so that it can be retried (if it is already in the working set, there is no change):

```
xvm:cluster> foswitch -dev 8:32 phys/lun22
```

Setting a New Affinity

The following command switches physvol `phys/lun33` to the `affinity=2` group if the working set does not have `affinity=2`:

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

Note: If the working set already had `affinity=2`, no switch is made.

The following command switches physvol `phys/lun33` to a path of `affinity=2` for all nodes in the cluster if the working set does not have `affinity=2`:

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

Note: In the cluster domain, you should include the `-cluster` option so that the `affinity` setting is consistent for all nodes in the cluster.

Chapter 7

XVM Administration Procedures

This chapter discusses the following:

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.

Assessing the System and Disk Status

Before configuring an XVM 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 GPT disk:
  - For information on how the partitions must be configured on a GPT disk, see "Partition Layout with GPT Disk Format" on page 30.

  - For information on formatting a disk for use in XVM under Linux, see "Preparing to Configure XVM Volumes in the Local Domain" on page 31.

Note: If you attempt to use XVM to label a disk that is not 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 the following command to view the names of the disks on the system that have not been assigned to XVM:

  ```
  # xvm show unlabeled/*
  ```

- 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 (physvols) that will be shared must be physically connected to all nodes in the cluster.
- Use the options of the `show` command to view your system configuration and status. For example, the following command displays the stripe unit for `stripe0`:

  ```
  xvm:cluster> show -v stripe0
  ```

- To configure XVM volumes, you must be logged in as `root`. However, you can display volume configuration information even if you do not have `root` privileges.

Note: As you configure a volume, remember that you can view 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
```

Creating a Volume with a Three-Way Stripe

Procedure 7-1 shows an example of creating a simple volume that stripes data across three disks. In this example, the entire usable space of each disk is used for the slice. Figure 7-1 shows the volume that Procedure 7-1 creates.

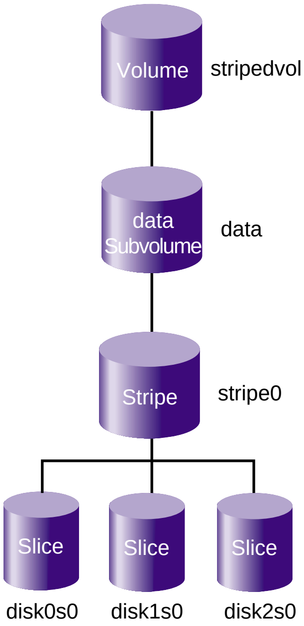


Figure 7-1 Volume with a Three-Way Stripe

Procedure 7-1 Creating a Volume with a Three-Way Stripe

1. Assign disks to XVM. You must perform this procedure only once for each disk that you will be using to create volumes.

   For example, to assign three disks to XVM:

```
# xvm
xvm:cluster> label -name disk0 unlabeled/dev/pm/SGI-TP9700--lun0-600a0b8000269d1e0000c9b14d31a849
disk0
xvm:cluster> label -name disk1 unlabeled/dev/pm/SGI-TP9700--lun1-600a0b8000269d1e0000c9b14d31a849
disk1
xvm:cluster> label -name disk2 unlabeled/dev/pm/SGI-TP9700--lun2-600a0b8000269d1e0000c9b14d31a849
disk2
```

2. View all of the disks that have been assigned as XVM physvols to verify what you have labeled. For example:

```
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 in each of the XVM physvols. For example:

```
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 slices and explicitly name the volume (stripedvol in this case):

```
xvm:cluster> stripe -volname stripedvol slice/disk0s0 slice/disk1s0 slice/disk2s0
</dev/cxvm/stripedvol> stripe/stripe0
```

In this example:

- A data subvolume will automatically be generated.
- /dev/cxvm/stripedvol is the name of the volume on which you can execute the mkfs command. The volume was named explicitly and will therefore persist across reboots
- stripe/stripe0 is the name of the stripe object. The name of the stripe object is automatically generated and is therefore subject to change upon reboot.

5. View the resulting topology of the volume:

```
xvm:cluster> show -topology 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:

```
xvm:cluster> exit
```

7. Make the filesystem for the volume. For example:

```
# 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
```

   For more information, see the `mkfs`(8) man page.

8. Mount the filesystem:

   - For a shared filesystem in a CXFS cluster, mount the filesystem with the CXFS GUI or the `cxfs_admin` command, as described in *CXFS 7 Administrator Guide for SGI InfiniteStorage*.

   - For a local filesystem, put the volume in the `fstab` file and use the `mount` command to mount the filesystem. For more information, see the `fstab`(5) and `mount`(8) man pages.

Striping a Portion of a Disk

Procedure 7-2 shows an example of creating a stripe on the outer third of a disk. It also includes advice about naming volume elements. Figure 7-2 shows the volume this example creates.

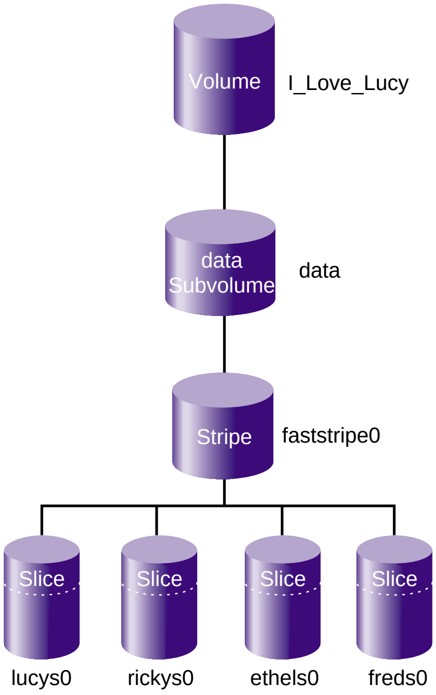


Figure 7-2 Striping a Portion of a Disk

Procedure 7-2 Striping a Portion of a Disk

1. Assign disks to XVM. For example, to assign four disks to XVM:

```
# xvm
xvm:cluster> label -name lucy unlabeled/dev/pm/SGI-TP9700--lun0-600a0b8000269d1e0000c9b14d31a849
lucy
xvm:cluster> label -name ricky unlabeled/dev/pm/SGI-TP9700--lun1-600a0b8000269d1e0000c9b14d31a849
ricky
xvm:cluster> label -name ethyl unlabeled/dev/pm/SGI-TP9700--lun2-600a0b8000269d1e0000c9b14d31a849
ethyl
xvm:cluster> label -name fred unlabeled/dev/pm/SGI-TP9700--lun3=600a0b8000269d1e0000c9b14d31a849
fred
```

Note: Four separate controllers are chosen for better stripe performance.

2. Partition one-third of each disk for the stripe, using one of the following methods:

   - 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
     ```

   - 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 by using the `show` command. For example:

   - If using 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
     ```

   - If allocating space explicitly:

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

4. Create the stripe. For example, to explicitly name the volume `I_Love_Lucy` and name the stripe `faststripe0`:

```
xvm:cluster> stripe -volname I_Love_Lucy -vename faststripe0 \
-unit 128 slice/lucys2 slice/rickys0 slice/ethys0 slice/freds0
</dev/cxvm/I_Love_Lucy> stripe/faststripe0
```

5. View the stripe. For example:

   ```
   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
   ```

6. Exit the `xvm` tool:

   ```
   xvm:cluster> exit
   ```

7. Make the filesystem for the volume. For example:

```
# 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
```

8. Mount the filesystem:

   - For a shared filesystem in a CXFS cluster, mount the filesystem with the CXFS GUI or the `cxfs_admin` command, as described in *CXFS 7 Administrator Guide for SGI InfiniteStorage*.

   - For a local filesystem, put the volume in the `fstab` file and use the `mount` command to mount the filesystem. For more information, see the `fstab`(5) and `mount`(8) man pages.

Creating a Volume with a `log` Subvolume and a Concat

Procedure 7-3 describes an example of creating a volume that includes both a `data` subvolume and a `log` subvolume as well as a concat. 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 7-3 shows the volume this example creates.

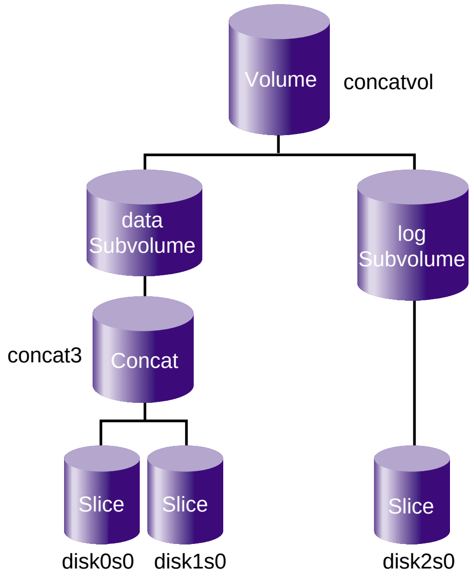


Figure 7-3 Volume with a `log` Subvolume and a Concat

Procedure 7-3 Volume with a `log` Subvolume and a Concat

1. Assign three disks to XVM:

```
# xvm
xvm:cluster> label -name disk0 unlabeled/dev/pm/SGI-TP9700--lun0-600a0b8000269d1e0000c9b14d31a849
disk0
xvm:cluster> label -name disk1 unlabeled/dev/pm/SGI-TP9700--lun1-600a0b8000269d1e0000c9b14d31a849
disk1
xvm:cluster> label -name disk2 unlabeled/dev/pm/SGI-TP9700--lun2-600a0b8000269d1e0000c9b14d31a849
disk2
```

2. Create a slice that consists of all of the usable blocks in each of the XVM physvols:

```
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 and name the associated new volume `concatvol` and view the topology results:

```
xvm:cluster> concat -volname concatvol slice/disk0s0 slice/disk1s0
</dev/cxvm/concatvol>  concat/concat3
xvm:cluster> show -topology vol/concatvol
vol/concatvol                     0 online
    subvol/concatvol/data      35554848 online
        concat/concat3             35554848 online,tempname
            slice/disk0s0              17777424 online
            slice/disk1s0              17777424 online
```

Note: The name of the concat is generated automatically by XVM and therefore may not be persistent across reboots. The name `concat3` indicates that this is the fourth generated concat name created by XVM.

4. Create the `log` subvolume consisting of the third slice and generate a temporary name for the volume:

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

   Note: You do not need to name this volume because you will attach the `log` subvolume to the existing `concatvol` volume in step 5.

5. Attach the `log` subvolume to the existing `concatvol` volume and display the resulting topology of the volume:

   ```
   xvm:cluster> attach subvol/vol7/log vol/concatvol
   vol/concatvol
   xvm:cluster> show -topology 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 Volume from the Bottom Up

When you configure an XVM volume, you can create the volume's hierarchy from the bottom up or from the top down. The following example creates a volume that includes a `data` subvolume, a `log` subvolume, and an `rt` subvolume.

Figure 7-4 shows the volume this example creates.

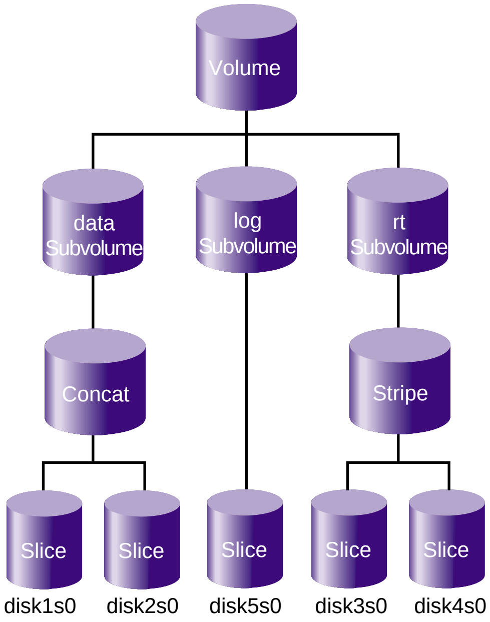


Figure 7-4 Creating a Volume from the Bottom Up

Procedure 7-4 Create a Volume from the Bottom Up

1. Assign five disks to XVM:

```
# xvm
xvm:local> label -name disk1 unlabeled/dev/pm/SGI-TP9700--lun0-600a0b8000269d1e0000c9b14d31a849
disk1
xvm:local> label -name disk2 unlabeled/dev/pm/SGI-TP9700--lun1-600a0b8000269d1e0000c9b14d31a849
disk2
xvm:local> label -name disk3 unlabeled/dev/pm/SGI-TP9700--lun2-600a0b8000269d1e0000c9b14d31a849
disk3
xvm:local> label -name disk4 unlabeled/dev/pm/SGI-TP9700--lun3-600a0b8000269d1e0000c9b14d31a849
disk4
xvm:local> label -name disk5 unlabeled/dev/pm/SGI-TP9700--lun4-600a0b8000269d1e0000c9b14d31a849
disk5
```

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

```
xvm:local> slice -all disk*
</dev/lxvm/disk1s0>  slice/disk1s0
</dev/lxvm/disk2s0>  slice/disk2s0
</dev/lxvm/disk3s0>  slice/disk3s0
</dev/lxvm/disk4s0>  slice/disk4s0
</dev/lxvm/disk5s0>  slice/disk5s0
```

3. Create the concat that will constitute the `data` subvolume:

```
xvm:local> concat -tempname slice/disk1s0 slice/disk2s0
</dev/lxvm/vol0>  concat/concat0
```

4. Create the stripe that will constitute the `rt` subvolume:

```
xvm:local> stripe -tempname slice/disk3s0 slice/disk4s0
</dev/lxvm/vol1>  stripe/stripe1
```

5. Create the `data` subvolume:

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

6. Create the `rt` subvolume:

```
xvm:local> subvolume -tempname -type rt stripe/stripe1
</dev/lxvm/vol3,rt>  subvol/vol3/rt
```

7. Create the `log` subvolume:

```
xvm:local> subvolume -tempname -type log slice/disk5s0
</dev/lxvm/vol4,log>  subvol/vol4/log
```

8. Create a volume that contains the three subvolumes and display the resulting topology:

```
xvm:local> volume -volname myvol subvol/vol2/data \
subvol/vol4/log subvol/vol3/rt
vol/myvol
xvm:local> show -topology myvol
vol/myvol                         0 online
    subvol/myvol/data          35558032 online
        concat/concat0             35558032 online,tempname
            slice/disk1s0              17779016 online
            slice/disk2s0              17779016 online
    subvol/myvol/log               8192 online
        slice/disk5s0                  8192 online
    subvol/myvol/rt            35557888 online
        stripe/stripe1             35557888 online,tempname
            slice/disk3s0              17779016 online
            slice/disk4s0              17779016 online
```

Creating a Volume from the Top Down

The example in this section creates a volume similar to the one created in the example in "Creating a Volume with a `log` Subvolume and a Concat" on page 133 (shown in Figure 7-3), but it creates an empty volume first before attaching the child volume elements for that volume.

Procedure 7-5 Creating a Volume from the Top Down

1. Assign three disks to XVM:

```
# xvm
xvm:cluster> label -name disk0 unlabeled/dev/pm/SGI-TP9700--lun0-600a0b8000269d1e0000c9b14d31a849
disk0
xvm:cluster> label -name disk1 unlabeled/dev/pm/SGI-TP9700--lun1-600a0b8000269d1e0000c9b14d31a849
```

```
disk1
xvm:cluster> label -name disk2 unlabeled/dev/pm/SGI-TP9700--lun2-600a0b8000269d1e0000c9b14d31a849
disk2
```

2. Create a slice that consists of all of the usable blocks of each of the XVM physvols 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` and display the resulting topology:

   ```
   xvm:cluster> volume -volname topdownvol
   vol/topdownvol
   xvm:cluster> show -topology vol/top*
   vol/topdownvol                    0 offline
       (empty)                           * *
   ```

4. Create an empty concat and display the resulting topology:

   ```
   xvm:cluster> concat -tempname
   </dev/cxvm/vol8>  concat/concat5
   xvm:cluster> show -topology vol/vol8
   vol/vol8                          0 offline,tempname
       subvol/vol8/data                  0 offline,pieceoffline
           concat/concat5                    0 offline,tempname
               (empty)                           * *
   ```

5. Attach the generated `data` subvolume that contains the concat to `topdownvol` and display the resulting topology:

   ```
   xvm:cluster> attach subvol/vol8/data vol/topdownvol
   vol/topdownvol
   xvm:cluster> show -topology vol/topdownvol
   vol/topdownvol                    0 offline
       subvol/topdownvol/data            0 offline,pieceoffline
           concat/concat5                    0 offline,tempname
               (empty)                           * *
   ```

6. Attach two slices to fill the empty concat and display the resulting topology:

```
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
```

7. Create a `log` subvolume:

```
xvm:cluster> subvol -tempname -type log
</dev/cxvm/vol9,log>  subvol/vol9/log
```

8. Attach the `log` subvolume to `topdownvol` and display the resulting topology:

```
xvm:cluster> attach subvol/vol9/log vol/topdownvol
vol/topdownvol
xvm:cluster> show -topology 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)                           * *
```

9. Attach the third slice to the `log` subvolume and display the resulting topology:

```
xvm:cluster> attach slice/disk2s0 subvol/topdownvol/log
</dev/cxvm/topdownvol,log>  subvol/topdownvol/log
xvm:cluster> show -topology 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 a Volume with Striped Mirrors

The following example creates a volume with striped mirrors. shown in Figure 7-5. In this example, the volume contains a stripe that consists of two mirrors, each mirroring a slice.

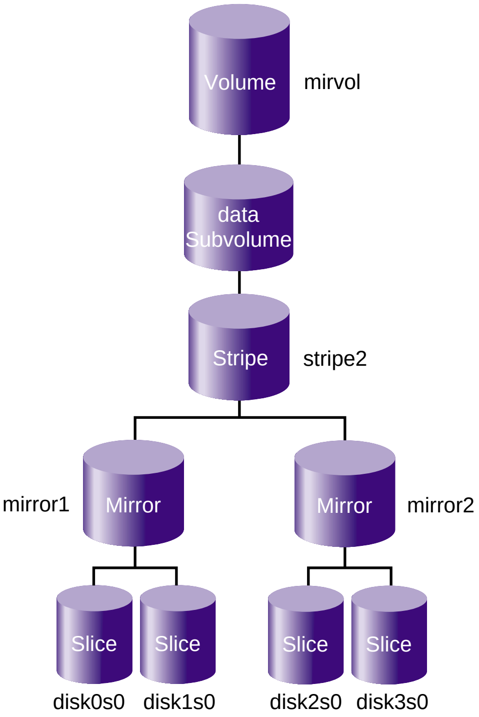


Figure 7-5 Volume with Striped Mirrors

1. Assign four disks to XVM:

```
xvm:cluster> label -name disk0 unlabeled/dev/pm/SGI-TP9700--lun0-600a0b8000269d1e0000c9b14d31a849
disk0
xvm:cluster> label -name disk1 unlabeled/dev/pm/SGI-TP9700--lun1-600a0b8000269d1e0000c9b14d31a849
disk1
xvm:cluster> label -name disk2 unlabeled/dev/pm/SGI-TP9700--lun2-600a0b8000269d1e0000c9b14d31a849
disk2
xvm:cluster> label -name disk3 unlabeled/dev/pm/SGI-TP9700--lun3-600a0b8000269d1e0000c9b14d31a849
disk3
```

2. Create a slice using all of the usable blocks on each XVM physvol:

   ```
   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. Because 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 system log, and another message would be written to the system log when the revive completes.

   You do 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 volume:

```
xvm:cluster> show -topology 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
```

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 7-6 shows the configuration of the original filesystem that has been built and mounted.

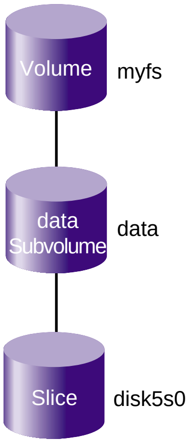


Figure 7-6 Original Online Filesystem

The original filesystem `myfs` is configured as follows:

```
xvm:cluster> show -topology myfs
vol/myfs                          0 online
    subvol/myfs/data             102400 online,open
        slice/disk5s0                102400 online,open
```

Procedure 7-6 reconfigures this filesystem into one that consists of a four-way stripe.

Procedure 7-6 Reconfigure an Online Volume with a Four-way Stripe by using a Mirror

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
   ```

2. Create the stripe using a system-generated name and the default stripe unit (128 512–byte blocks) and then display the resulting topology of the stripe:

   Note: This uses all of the blocks of each slice because the slices are multiples of the stripe unit in size.

   ```
   xvm:cluster> stripe -tempname slice/disk[1234]s0
   </dev/cxvm/vol5> stripe/stripe5
   xvm:cluster> show -topology stripe5
   stripe/stripe5               102400 online,tempname
       slice/disk1s0                 25600 online
       slice/disk2s0                 25600 online
       slice/disk3s0                 25600 online
       slice/disk4s0                 25600 online
   ```

3. Insert a temporary mirror above the point that will be reconfigured (in this case, slice/disk5s0) and display the resulting topology of the volume:

   ```
   xvm:cluster> insert mirror slice/disk5s0
   </dev/cxvm/myfs>  mirror/mirror5
   xvm:cluster> show -topology myfs
   vol/myfs                                 0 online
       subvol/myfs/data                102400 online,open
           mirror/mirror5                  102400 online,tempname,open
               slice/disk5s0                   102400 online,open
   ```

   Figure 7-7 shows the configuration of the filesystem myfs after the insertion of the mirror.

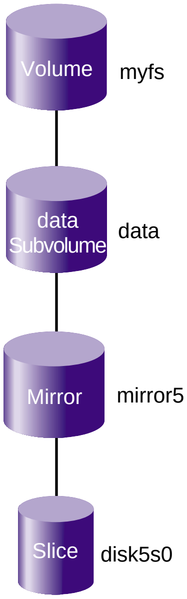


Figure 7-7 Filesystem after Insertion of Mirror

4. Attach the stripe to the mirror (mirror5), which will initiate a revive that replicates the data of slice/disk5s0 on stripe5 and display the resulting topology of the volume:

```
xvm:cluster> attach stripe/stripe5 mirror/mirror5
</dev/cxvm/myfs>  mirror/mirror5
xvm:cluster> show -topology 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 7-8 shows the configuration of the filesystem myfs after the stripe has been attached to the mirror.

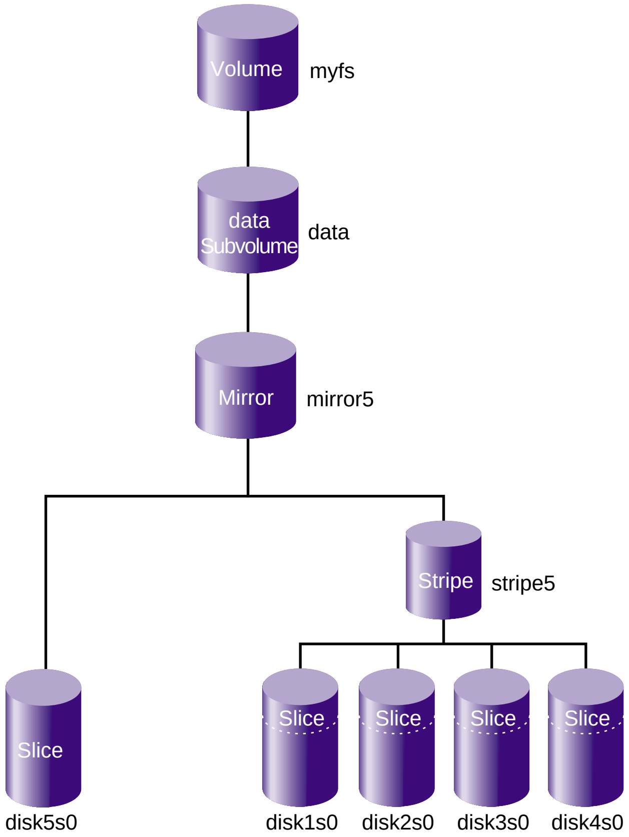


Figure 7-8 Filesystem after Attaching Stripe to Mirror

Note: You can use the `show` command to display the status of the revive operation. The length of time required depends on the size of the mirror.

---

5. After the mirror revive completes, detach `slice/disk5s0` from the mirror:

```
xvm:cluster> detach slice/disk5s0
</dev/cxvm/disk5s0>  slice/disk5s0
```

Note: You must wait for the mirror revive to complete before you can detach the slice because you cannot detach the last valid piece of an open mirror. Until the revive completes, the slice is the only valid leg of the mirror.

Figure 7-9 shows the configuration of the filesystem `myfs` after the original slice has been detached.

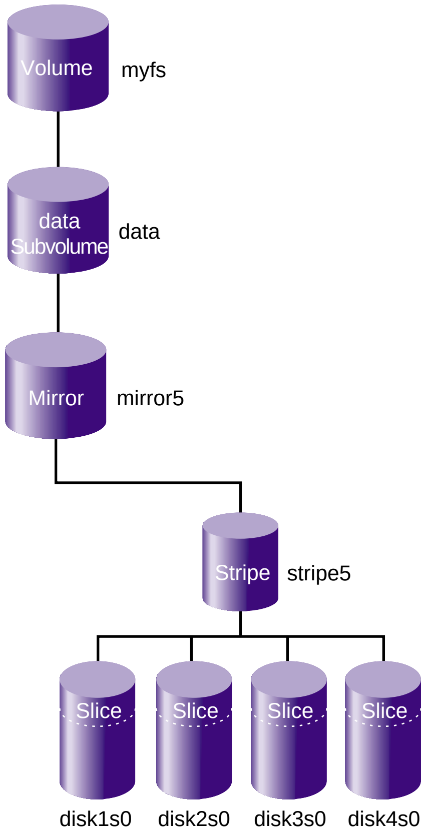


Figure 7-9 Filesystem after Detaching Original Slice

6. Remove the mirror layer from the tree and display the resulting topology of the volume (which is now a four-way stripe):

```
xvm:cluster> collapse mirror/mirror5
xvm:cluster> show -topology 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 7-10 shows the final configuration of the filesystem `myfs`.

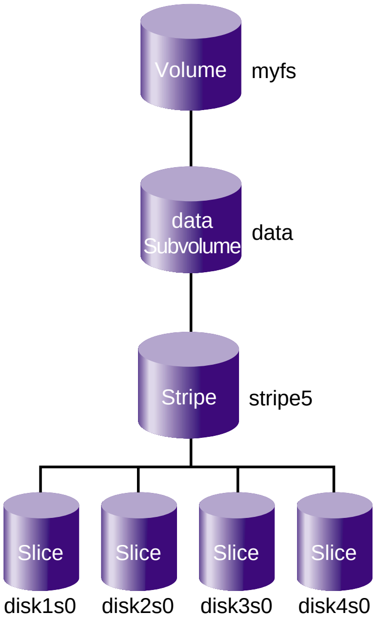


Figure 7-10 Reconfigured Filesystem

Online Modification of a Volume

Note: In these procedures, the modifications to the volume are performed after you have made a filesystem on the volume and mounted the filesystem.

The following sections describe example procedures for creating and modifying a volume:

- "Creating the Volume" on page 150
- "Growing the Volume" on page 152
- "Mirroring Data on the Volume" on page 154
- "Converting a Concat to a Stripe using Mirroring" on page 156
- "Expanding a Mirror" on page 158
- "Removing a Mirror" on page 159
- "Mirroring Individual Stripe Members" on page 161

Creating the Volume

Procedure 7-7 creates a simple volume that contains a single slice. Figure 7-11 shows the simple volume this procedure creates.

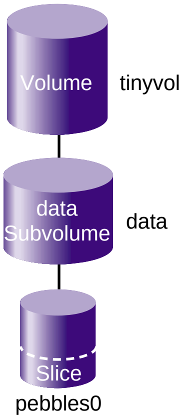


Figure 7-11 Simple Volume with One Slice

Procedure 7-7 Creating a Volume with One Slice

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:

```
xvm:cluster> exit
```

3. Make the filesystem:

```
# 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
```

4. Mount the filesystem:

   - For a shared filesystem in a CXFS cluster, mount a filesystem with the CXFS GUI or the `cxfs_admin` command, as described in *CXFS 7 Administrator Guide for SGI InfiniteStorage*.

   - For a local filesystem, put the volume in the `fstab` file and use the `mount` command to mount the filesystem. For more information, see the `fstab`(5) and `mount`(8) man pages.

Growing the Volume

Procedure 7-8 grows the volume created in Procedure 7-7, page 151. Figure 7-12 shows the volume after the insertion of a concat to grow the volume.

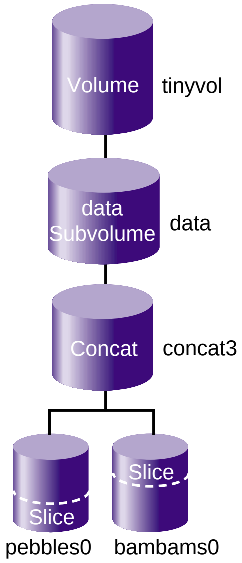


Figure 7-12 Volume after Insert

Procedure 7-8 Growing the Volume

1. Display the topology of volume `tinyvol`:

```
xvm:cluster> show -topology 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 and display the resulting topology of the volume:

```
xvm:cluster> insert concat slice/pebbles0
</dev/cxvm/tinyvol> concat/concat3
xvm:cluster> show -topology tinyvol
vol/tinyvol                       0 online
    subvol/tinyvol/data          177792 online,open
        concat/concat3               177792 online,tempname,open
            slice/pebbles0               177792 online,open
```

3. Create a free slice on the physvol `bambam`:

```
xvm:cluster> slice -start 0 -length 177792 bambam
</dev/xvm/bambams0> slice/bambams0
```

4. Attach the slice to `tinyvol`. There are two different ways to specify the concat volume element to which you are attaching the slice:

   - Attach the slice by the relative location of the volume element:

     ```
     xvm:cluster> attach slice/bambams0 tinyvol/data/0
     </dev/cxvm/tinyvol> concat/concat3
     ```

   - Attach 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 71.

5. Display the results of the `attach` command:

```
xvm:cluster> show -topology 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
```

6. Exit the xvm CLI:

   ```
   xvm:cluster> exit
   ```

7. Grow the filesystem, using the mount point where you mounted the filesystem in step 4 of Procedure 7-7, page 151 (in this example, the mount point is /clusterdisk):

```
# 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 Volume

Procedure 7-9 creates a mirror for the data in the filesystem. Figure 7-13 shows the volume after the insertion of the mirror.

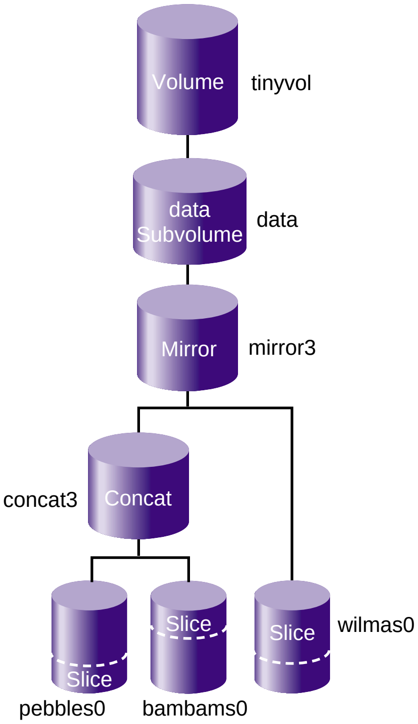


Figure 7-13 Volume After Mirroring

Procedure 7-9 Mirroring Data on the Volume

1. Change `tinyvol` to include a mirror container and display the resulting topology:

```
# xvm
xvm:cluster> insert mirror tinyvol/data/0
</dev/cxvm/tinyvol>  mirror/mirror3
xvm:cluster> show -topology 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
```

2. 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
```

3. Attach the slice to the mirror and display the results:

```
xvm:cluster> attach slice/wilmas0 tinyvol/data/0
</dev/cxvm/tinyvol>  mirror/mirror3
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%
```

Note: In this example, the revive that was initiated when the slices were attached to the mirror has not yet completed.

Converting a Concat to a Stripe using Mirroring

Figure 7-14 converts the previously created concat to a stripe that replaces the concat in the mirror. Figure 7-14 shows the resulting volume.

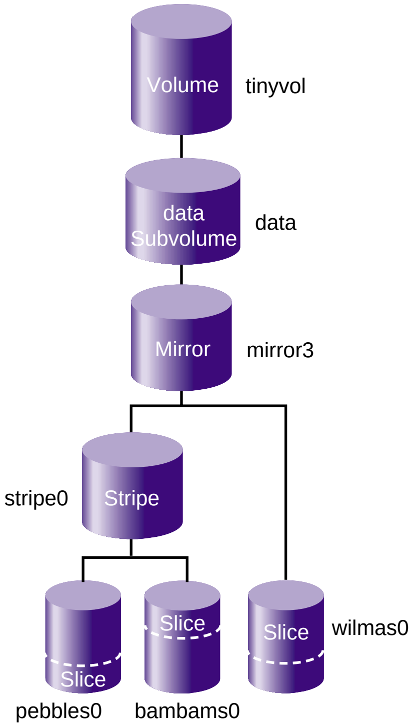


Figure 7-14 Volume after Conversion from Concat to Mirror

Note: The new storage must be at least as large as the existing storage in order to attach as a mirror leg (and the same storage in a stripe will be slightly smaller than the concat because it automatically does alignment, where a concat uses all space).

Procedure 7-10 Converting a Concat to a Stripe using Mirroring

1. Break the mirror:

```
xvm:cluster> detach -tempname mirror3/0
</dev/cxvm/vol6>  concat/concat3
```

2. Delete the concat object, detaching and keeping its slices:

```
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 and display the results:

```
xvm:cluster> attach stripe0 mirror3
xvm:cluster> show -topology 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
```

Note: In this example, the revive that was initiated when the stripes were attached to the mirror has not yet completed.

Expanding a Mirror

If you increase the storage size beneath a mirror, the mirror itself will not automatically increase in size. To make it reflect the current size, you can insert and collapse a new mirror. For example:

```
xvm:local> show -top vol/drives0
vol/drives0                       0 online,accessible
     subvol/drives0/data       111406464 online,accessible
         mirror/mirror0            111406464 online,tempname,incomplete,accessible
             (empty)                           * *
             stripe/stripe1            167109504 online,tempname,accessible
                 slice/drives2              55703232 online,accessible
                 slice/drives3              55703232 online,accessible
                 slice/drives4              55703232 online,accessible
xvm:local> insert mirror stripe1
</dev/lxvm/drives0>  mirror/mirror1
xvm:local> show -top vol/drives0
vol/drives0                       0 online,accessible
```

```
     subvol/drives0/data       111406464 online,accessible
         mirror/mirror0            111406464 online,tempname,incomplete,accessible
             (empty)                           * *
             mirror/mirror1            167109504 online,tempname,accessible
                 stripe/stripe1            167109504 online,tempname,accessible
                     slice/drives2              55703232 online,accessible
                     slice/drives3              55703232 online,accessible
                     slice/drives4              55703232 online,accessible
xvm:local> collapse mirror0
xvm:local> show -top vol/drives0
vol/drives0                       0 online,accessible
     subvol/drives0/data       167109504 online,accessible
         mirror/mirror1            167109504 online,tempname,accessible
             stripe/stripe1            167109504 online,tempname,accessible
                 slice/drives2              55703232 online,accessible
                 slice/drives3              55703232 online,accessible
                 slice/drives4              55703232 online,accessible
```

Removing a Mirror

Procedure 7-11 removes the mirror layer from the volume. Figure 7-15 shows the volume after the mirror has been removed.

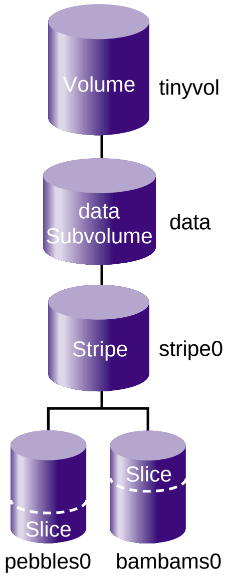


Figure 7-15 Volume after Mirror Removal

Procedure 7-11 Removing a Mirror

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 and display the results:

```
xvm:cluster> collapse mirror3
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

Procedure 7-12 mirrors the individual slices that make up the stripe. Figure 7-16 shows the volume this example yields.

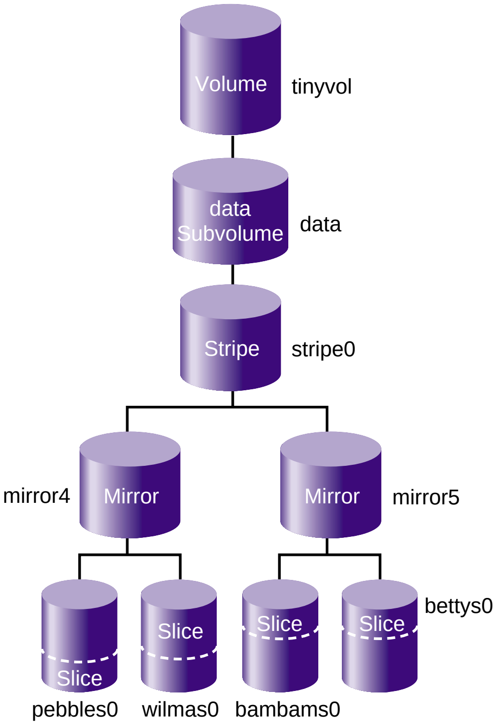


Figure 7-16 Volume after Mirroring Slices

Procedure 7-12 Mirroring Individual Stripe Members

1. Place the slices within mirror containers and display the results (the following examples demonstrate alternative 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
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
```

2. 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
```

3. Attach the slices to the mirrors and display the resulting topology:

Note: `wilmas0` is larger than `pebbles0` but 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 stripe0/1
</dev/cxvm/tinyvol>  mirror/mirror4
xvm:cluster> show -topology 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

This section discusses the following:

- "Overview of Using a GPT Label" on page 163
- "Making a GPT Label" on page 164
- "Making the XVM Label and Slices" on page 166

Overview of Using a GPT Label

SGI storage solutions (TP-9*XXX* 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 alternating between controllers 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 `SGIAVT`.

It is important when planning your XVM volumes and GPT labels to use this exclusion zone to minimize path failover during boot time. Figure 7-17 shows the optimal way of creating XVM volumes and a GPT label on a LUN using LSI RAID:

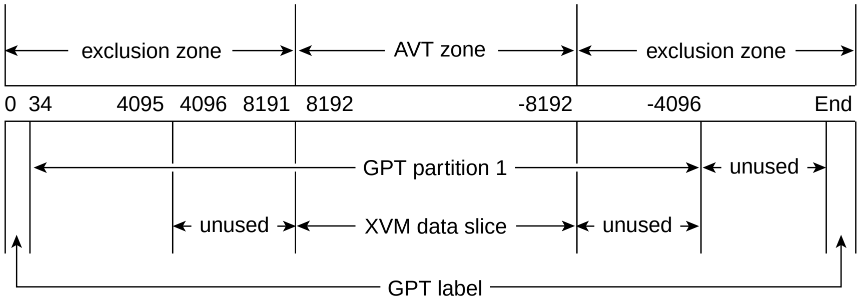


Figure 7-17 Creating XVM Volumes and a GPT Label on a LUN

The `xvm label` command automatically lays out the LUN to meet these requirements. See "Do Not Create Slices Within the RAID Exclusion Zone" on page 56.

Making a GPT Label

Note: GPT label itself reserves blocks `0-34` for it 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. Create the label and then create one partition. This partition will contain the XVM label data followed by the user data. The partition must start on or before block 63 and end at *DISK_SIZE*-8192. 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.

The following examples show how to create a GPT label 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/disk/by-path/pci-0000:08:03.1-fc-0x22000011c61dd850-lun-0
GNU Parted 2.3
Using /dev/sdca
Welcome to GNU Parted! Type ’help’ to view a list of commands.
(parted) mklabel gpt
Warning: The existing disk label on /dev/sdca will be destroyed and all data on this disk will be lost. Do
you want to continue?
Yes/No? y
(parted) mkpart
Partition name?  []?
File system type?  [ext2]? xfs
Start? 0
End? -1
Warning: You requested a partition from 0s to 585937499s.
The closest location we can manage is 34s to 585937466s.
Is this still acceptable to you?
Yes/No? y
Warning: The resulting partition is not properly aligned for best performance.
Ignore/Cancel? i
(parted) print
Model: SGI ST336753FC (scsi)
Disk /dev/sdca: 36.7GB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
```

```
Number  Start   End     Size    File system  Name  Flags
 1      17.4kB  36.7GB  36.7GB

(parted) quit
Information: You may need to update /etc/fstab.
```

Note: You can ignore the warning about alignment from `parted`. You can align the data areas by using the XVM `slice -align` option.

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

```
# xvm label -name test1 /dev/disk/by-path/pci-0000:08:03.1-fc-0x22000011c61dd850-lun-0
test1
Performing automatic probe for alternate paths.
Performing automatic path switch to preferred path for phys/test1.
# xvm show -v phys/test1
XVM physvol phys/test1
=========================
size: 71670988 blocks  sectorsize: 512 bytes  state: online,local,accessible
uuid: ced104db-2e1c-4d6d-ada5-89c7b3d8ee59
system physvol:  no
path manager device:  /dev/pm/SGI-TP9700--lun0-600a0b8000269d1e0000c9b14d31a849 on host petrel
using paths:
 /dev/disk/by-path/pci-0000:08:03.0-fc-0x22000011c61dd850-lun-0 <sdao 66:128> affinity=none ws
 /dev/disk/by-path/pci-0000:08:03.0-fc-0x21000011c61dd850-lun-0 <sdaq 66:160> affinity=none
 /dev/disk/by-path/pci-0000:08:03.1-fc-0x22000011c61dd850-lun-0 <sdca 68:224> affinity=none
 /dev/disk/by-path/pci-0000:08:03.1-fc-0x21000011c61dd850-lun-0 <sdcb 68:240> affinity=none
Disk has the following XVM label:
  Clusterid:  0
  Host Name:  petrel
  Disk Name:  test1  Magic:  0x786c6162 (balx)     Version 2
  Uuid: ced104db-2e1c-4d6d-ada5-89c7b3d8ee59
  last update:  Tue Jun 26 16:10:21 2012
  state: 0x91<online,local,accessible> flags: 0x0<idle>
  secbytes:  512
  label area:  8157 blocks starting at disk block 35 (0 used)
  user area:   71670988 blocks starting at disk block 8192

Physvol Usage:
Start        Length       Name
```

```
---------------------------------------------------
0            71670988     (unused)

Local stats for phys/test1 since being enabled or reset:
------------------------------------------------------------------------------
stats collection is not enabled for this physvol
```

Making the XVM Label and Slices

You can make any number of slices, but the usual case is just one slice. In some cases, especially where striped volumes are used, good performance requires that the slice is aligned on certain boundaries of the underlying LUN. You can align the slice by using the `-align` option of the `slice` command

Procedure 7-13 Making the Slice

1. Create the slice and align it (assuming that the boundary of interest is every 768 KB, or 1536 sectors):

   ```
   xvm:cluster> slice -align 1536 phys/test1
   </dev/lxvm/test1s0>  slice/test1s0
   ```

2. Show the resulting physvol:

```
xvm:cluster> show -v phys/test1
XVM physvol phys/test1
=========================
size: 71670988 blocks  sectorsize: 512 bytes  state: online,local,accessible
uuid: cf38c332-42e4-4d5a-8f01-0f741790ac34
system physvol:  no
path manager device:  /dev/pm/SGI-TP9700--lun0-600a0b8000269d1e0000c9b14d31a849 on host petrel
using paths:
 /dev/disk/by-path/pci-0000:08:03.1-fc-0x21000011c61dd850-lun-0 <sdcb 68:240> affinity=none ws
 /dev/disk/by-path/pci-0000:08:03.1-fc-0x22000011c61dd850-lun-0 <sdca 68:224> affinity=none
 /dev/disk/by-path/pci-0000:08:03.0-fc-0x21000011c61dd850-lun-0 <sdaq 66:160> affinity=none
 /dev/disk/by-path/pci-0000:08:03.0-fc-0x22000011c61dd850-lun-0 <sdao 66:128> affinity=none
Disk has the following XVM label:
  Clusterid:  0
  Host Name:  petrel
  Disk Name:  test1
  Magic:  0x786c6162 (balx)     Version 2
  Uuid: cf38c332-42e4-4d5a-8f01-0f741790ac34
```

```
  last update:  Mon Jul  2 16:14:51 2012
  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/test1s0
71669248     1740         (unused)

Local stats for phys/test1 since being enabled or reset:
------------------------------------------------------------------------------
stats collection is not enabled for this physvol
```

Determining the Size of an XVM Volume

To determine the size of an XVM volume, use the `show` command to display the size of the volume's children in 512-byte blocks and add the sizes:

```
xvm:cluster> show vol/volumename/*
```

For example, the following shows that volume `drives10` has one child (the `data` subvolume) and consists of a total of 55,703,232 blocks:

```
xvm:cluster> show vol/drives10/*
subvol/drives10/data       55703232 online,open,accessible
```

For example, the following shows that volume `drives11` has two children (the `data` and `log` subvolumes) and consists of a total of 65,703,232 blocks:

```
xvm:cluster> show vol/drives11/*
subvol/drives11/data       55703232 online,open,accessible
subvol/drives11/log        10000000 online,open,accessible
```

Determining the Size of a Physvol

To determine the size of a physvol in 512-byte blocks, use the `show` command:

```
xvm:cluster> show phys/physvolname
```

For example:

```
xvm:cluster> show phys/drive
phys/drive                2339536896 online,cluster,accessible
```

Chapter 8

Statistics

This chapter discusses the following:

- "Overview of XVM Statistics" on page 169
- "Physvol Statistics" on page 170
- "Subvolume Statistics" on page 170
- "Stripe Statistics" on page 170
- "Concat Statistics" on page 171
- "Mirror Statistics" on page 172
- "Slice Statistics" on page 172

Note: The examples in this chapter assume that statistics have been turned on for the objects shown.

Accurate statistics are not displayed for volumes; the output shows "0" for all fields.

Overview of XVM Statistics

XVM can maintain statistics for physvols, subvolumes, stripes, concats, mirrors, and slices.

You use the stat option of the change command to enable/disable the collection of statistics and to reset the statistics for a volume element. Statistics are on by default. Statistics are enabled/disabled only for the specified layer of the XVM volume topology tree. If you want to collect statistics for multiple layers, you must specify each layer explicitly.

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

Statistics for all volume elements and physical volume (physvols) show the number of read and write operations as well as the number of 512-byte blocks read and written.

You can display statistics by using either show -stat or show -v in the xvm CLI.

Physvol Statistics

The following example displays the statistics of a physvol 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

Subvolume statistics show the number of read / write requests and the number of 512–byte blocks read / written. For example:

```
xvm:cluster> show -stat subvol/Stat/data
Local stats for subvol/Stat/data since being enabled or reset:
---------------------------------------------------------------------
read requests:          165
write requests:         32
512 byte blks read:     1958
512 byte blks written:  4096
```

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. For example:

```
xvm:cluster> show -stat stripe/stripe356
Local stats for stripe/stripe356 since being enabled or reset:
--------------------------------------------------------------------
read requests:          165
write requests:         32
```

```
512 byte blks read:     1958
512 byte blks written:  4096
Requests aligned at both start and end
        equal to stripe width:       46
        greater than stripe width:  0
        less than stripe width:     0
Requests aligned at start
        greater than stripe width:  0
        less than stripe width:     6
Requests aligned at end
        greater than stripe width:  0
        less than stripe width:     4

Requests unaligned
        equal to stripe width:      0
        greater than stripe width:  0
        less than stripe width:     141
```

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. For example:

```
xvm:cluster> show -stat concat/concat178
Local stats for concat/concat178 since being enabled or reset:
-------------------------------------------------------------------
read requests:          165
write requests:         32
512 byte blks read:     1958
512 byte blks written:  4096

reads straddling slices:       0
writes straddling slices:       0
```

Mirror Statistics

Mirror statistics show the read / write requests and the mirror synchronization reads / writes. For example:

```
xvm:cluster> show -stat mirror/mirror265
Local stats for mirror/mirror265 since being enabled or reset:
-------------------------------------------------------------------
read requests:          165
write requests:         32
512 byte blks read:     1958
512 byte blks written:  4096
Mirror synchronization reads:         0
Mirror synchronization writes:        0

Leg        Reads        Writes
0        165        32
```

Slice Statistics

Slice statistics show the number of read / write operations and the number of 512–byte blocks read / written. For example:

```
xvm:cluster> show -stat slice/temps0
Local stats for slice/temps0 since being enabled or reset:
--------------------------------------------------------------------
read requests:          165
write requests:         32
512 byte blks read:     1958
512 byte blks written:  4096
```

Chapter 9

XVM Operation

This chapter discusses the following:

- "Cluster System Startup"
- "Mirror Revives" on page 173

Cluster System Startup

The following operations take place when booting a CXFS cluster that includes XVM volumes:

1. The system boots and probes all disks (SGI SAN disks and FC-hub disks, internal SCSI, and so on).
2. A boot script initiates the reading of all the labels and creates a view of all local volumes. Cluster volumes are not visible at this point.
3. The cluster is initialized.
4. On each node in the cluster, XVM reads all of the labels on the disk and creates a cluster-wide view of all volumes including the third-party SAN volumes.

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 leg is initially constructed
- A leg is attached to a mirror
- The system is booted with mirrors that are not synchronized
- A node in a CXFS cluster crashes when the mirror is open (see "Slow Mirror Revives" on page 225)

A message is written to the system log when a mirror begins reviving. Another message is written to the system log when this process is complete. If the revive fails, a message is written to the system console and the system log.

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 legs 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 "Mirror Revive at Creation" on page 43 and "Mirror Revive at Reboot" on page 44.

While a mirror is in the process of reviving, you can configure the volume that contains the mirror and you can perform I/O to the mirror. Displaying the mirror volume element with the `show` command will show the percentage of the mirror blocks that 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 system tunable parameters. For more information, see:i

- "Slow Mirror Revives" on page 225
- "Viewing XVM Parameters with `modinfo`" on page 235

Chapter 10

XVM Manager GUI

This chapter discusses the following:

- "XVM Manager GUI Overview" on page 175
- "Volume Element Tasks" on page 195
- "Disks Tasks" on page 206
- "Filesystems Tasks" on page 214
- "Privileges Tasks" on page 218

XVM Manager GUI Overview

This section discusses the following:

- "XVM Manager and CXFS Manager" on page 175
- "Starting the GUI via the Command Line" on page 176
- "Starting the GUI from the Web" on page 176
- "Summary of GUI Platforms" on page 177
- "Logging In" on page 178
- "GUI Features" on page 178
- "Key to Icons and States" on page 192

XVM Manager and CXFS Manager

The **XVM Manager** GUI and the **CXFS Manager** GUI contain the same tasks for configuring and administering XVM volumes. This guide documents only those tasks that pertain to XVM volume elements, disks, local filesystems, and privileges. For information about cluster functions and CXFS filesystems, see the *CXFS 7 Administrator Guide for SGI InfiniteStorage*.

Starting the GUI via the Command Line

To start the GUI using the command line, do the following:

1. Ensure that the following line is not commented out in the file `/etc/ld.so.conf`:

   ```
   /usr/lib64/sysadm/lib
   ```

2. Enter the following command line:

   ```
   # /usr/sbin/xvmgr
   ```

Note: If CXFS is installed, the `xvmgr` command will automatically invoke the **CXFS Manager** GUI, which provides access to the same set of XVM tasks as the **XVM Manager** GUI.

Starting the GUI from the Web

If you want to use a web-based version of the GUI, do the following:

1. Ensure that the following software products are installed on the node that you will connect to (by means of a Java-enabled web browser) for performing administrative operations:

   ```
   sgi-sysadm_xvm-web
   sgi-sysadm_cxfs-web
   ```

   These software products are part of the software normally installed with CXFS.

2. Ensure that an apache Web server is installed and running on the node.

3. Enable and restart the required web service:

   - RHEL:

     ```
     rhel# chkconfig httpd on
     rhel# service httpd restart
     ```

   - SLES:

     ```
     sles# chkconfig apache2 on
     sles# service apache2 restart
     ```

4. Close all browser windows and restart the browser.

5. Enter the URL `http://`*server*`/XVMManager/` where *server* is the name of the node on which XVM will run

6. At the resulting webpage, click the **XVM Manager** icon.

Summary of GUI Platforms

Table 10-1 describes the platforms where the GUI may be started, connected to, and displayed.

Table 10-1 GUI Platforms

GUI Mode	Where You Start the GUI	Where You Connect the GUI	Where the GUI Displays
`xvmgr`	A system with `sgi-sysadm_xvm-web` installed	The node that you want to use for XVM administration	The system where the GUI was invoked
Web	Any system with a web browser and Java2 1.6 or 1.6 plug-in installed and enabled	The node that you want to use for XVM administration	The same system with the web browser

Logging In

To ensure that the required GUI 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 using the GUI privilege tasks; see "Privileges Tasks" on page 218.

A dialog box will prompt you to log in to an XVM 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 (`xinetd` must be turned on via `chkconfig` and running)
- **Remote Shell** connects to the server via a user-specified command shell, such as `rsh` or `ssh`. 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`. Otherwise, the 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

GUI Features

The **XVM Manager** GUI provides access to the tools that help you configure and administer XVM:

- *Tasks* let you set up and monitor individual XVM objects.
- *Guided configurations* consist 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 creating a new volume and allows you to launch the necessary individual tasks by clicking their titles.

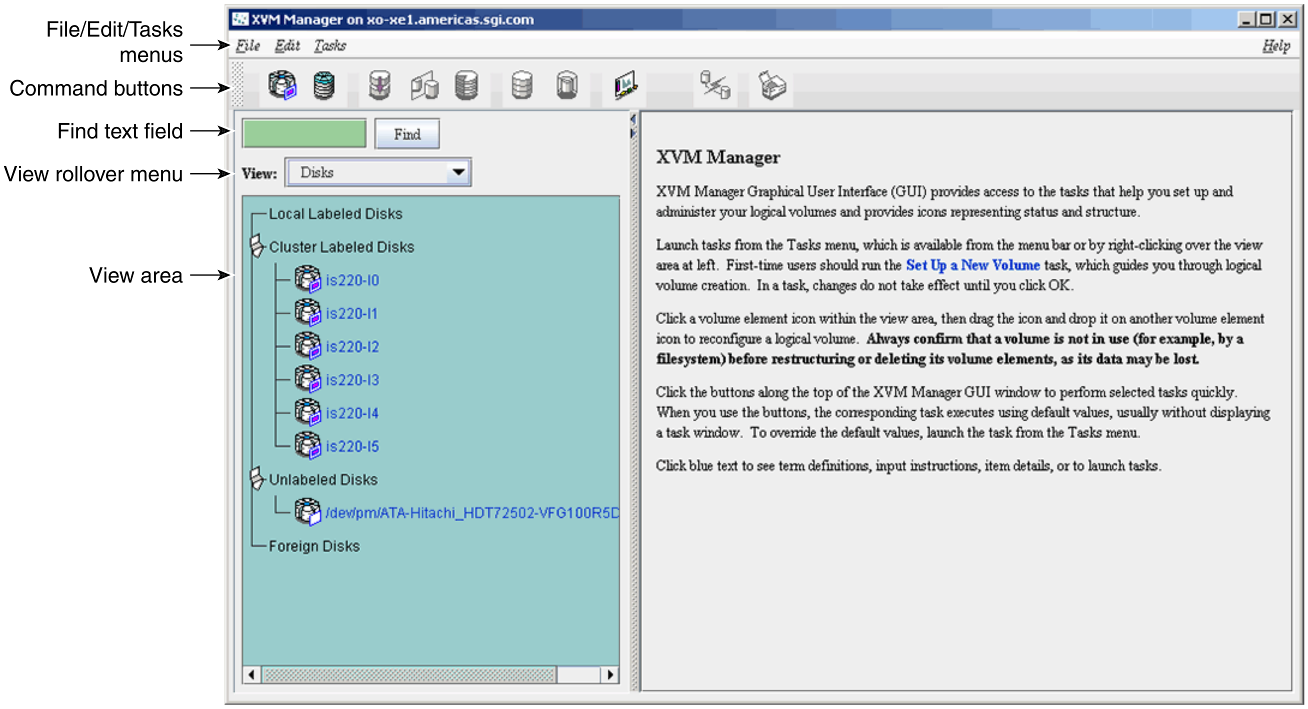
This section discusses the following:

- "GUI Window Layout" on page 179
- "Making Changes Safely" on page 183

- "File Menu" on page 183
- "Edit Menu" on page 184
- "Tasks Menu" on page 184
- "Help Menu" on page 184
- "Shortcuts Using Command Buttons" on page 184
- "View Menu" on page 186
- "Performing Tasks" on page 187
- "Using Drag-and-Drop" on page 189
- "Structuring Volume Topologies" on page 190
- "Configuring Disks" on page 190
- "Displaying State" on page 191
- "Getting More Information" on page 191
- "Selecting Items to View or Modify" on page 191
- "Important GUI and `xvm` Command Differences" on page 191

GUI Window Layout

Figure 10-1 shows the **XVM Manager** window.



XVM Manager

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.

Launch tasks from the Tasks menu, which is available from the menu bar or by right-clicking over the view area at left. First-time users should run the **Set Up a New Volume** task, which guides you through logical volume creation. In a task, changes do not take effect until you click OK.

Click a volume element icon within the view area, then drag the icon and drop it on another volume element icon to reconfigure a logical volume. **Always confirm that a volume is not in use (for example, by a filesystem) before restructuring or deleting its volume elements, as its data may be lost.**

Click the buttons along the top of the XVM Manager GUI window to perform selected tasks quickly. When you use the buttons, the corresponding task executes using default values, usually without displaying a task window. To override the default values, launch the task from the Tasks menu.

Click blue text to see term definitions, input instructions, item details, or to launch tasks.

Figure 10-1 XVM Manager GUI

By default, the window is divided into two sections: the *view area* to the left and the *details area* to the right. The details area shows generic overview text if no item is selected in the view area. You can use the arrows in the middle of the window to shift the display.

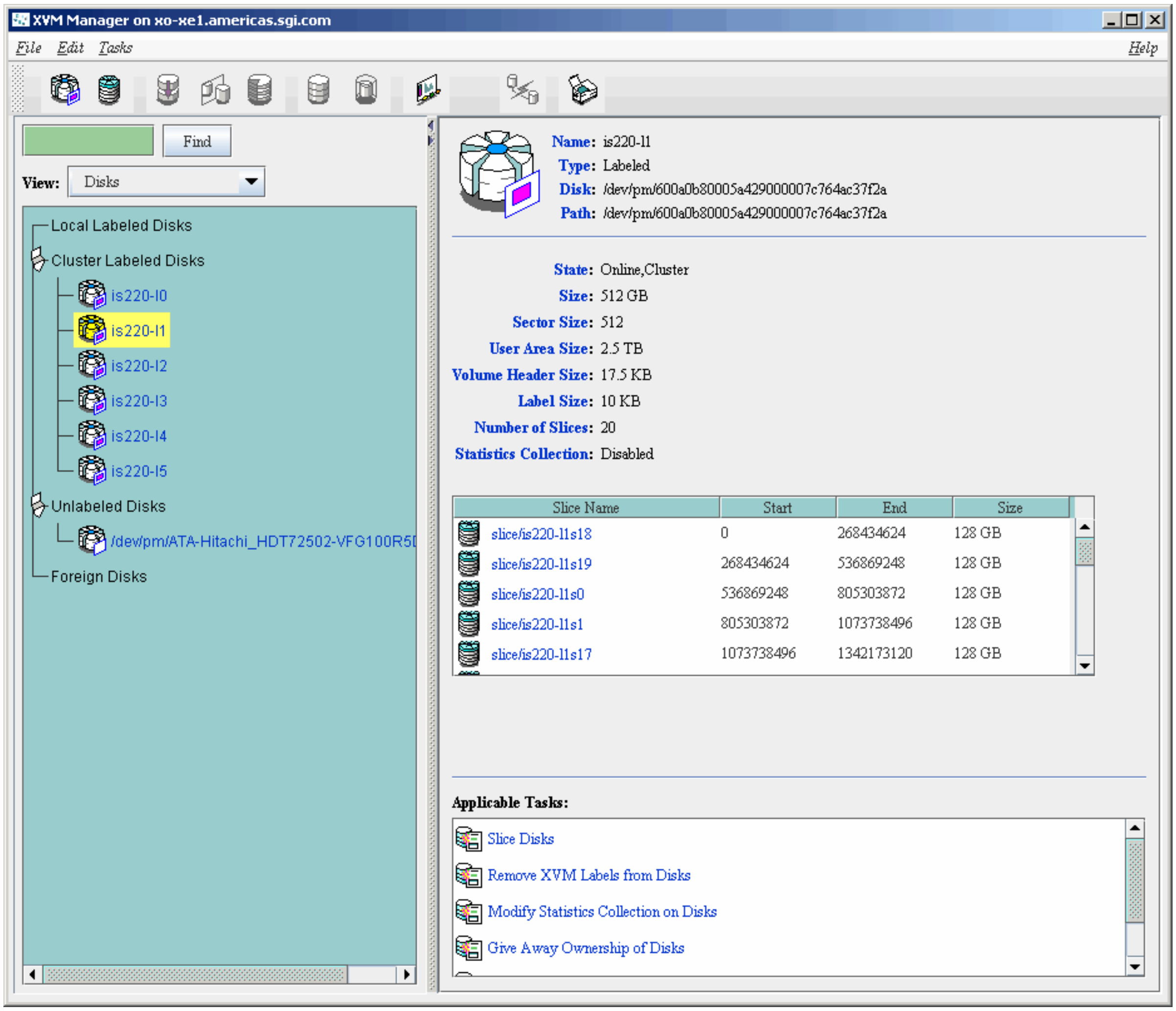
The command buttons along the top of the window provide a method of performing tasks quickly. When you click a button, the corresponding task executes using default values, usually without displaying a task window. For more information on the command buttons, see "Shortcuts Using Command Buttons" on page 184.

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** menu.

To view the details of any volume element, select it; 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 "Performing Tasks" on page 187.

Figure 10-2 shows an XVM GUI window with a selected component and the details area showing the details for that component.



Name: is220-l1
Type: Labeled
Disk: /dev/pm/600a0b80005a429000007c764ac37f2a
Path: /dev/pm/600a0b80005a429000007c764ac37f2a

State: Online,Cluster
Size: 512 GB
Sector Size: 512
User Area Size: 2.5 TB
Volume Header Size: 17.5 KB
Label Size: 10 KB
Number of Slices: 20
Statistics Collection: Disabled

Slice Name	Start	End	Size
slice/is220-l1s18	0	268434624	128 GB
slice/is220-l1s19	268434624	536869248	128 GB
slice/is220-l1s0	536869248	805303872	128 GB
slice/is220-l1s1	805303872	1073738496	128 GB
slice/is220-l1s17	1073738496	1342173120	128 GB

Applicable Tasks:

- Slice Disks
- Remove XVM Labels from Disks
- Modify Statistics Collection on Disks
- Give Away Ownership of Disks

Figure 10-2 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

When you are running XVM as a standalone product, the **Domain** is set to **local** by default in various tasks, indicating that the XVM objects that you create in this session are in the local domain. For more information about XVM domains, see "Local Domain and Cluster Domain" on page 26.

Making Changes Safely

Do not make configuration changes on two different nodes simultaneously, or use the XVM GUI and `xvm`(8) command simultaneously to make changes. You should run one instance of the `xvm` command or the XVM GUI on a single node when making changes at any given time. However, you can use any node when requesting status or configuration information. 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.

You should wait for a change to appear in the *view area* before making another change.

File Menu

The **File** menu lets you display the following:

- Multiple windows for this instance of the GUI
- System log file:`/var/log/messages`
- System administration log file: `/var/lib/sysadm/salog`

  The `salog` file shows the commands run directly by this instance of the GUI or some other instance of the GUI running commands on the system. (Changes should not be made simultaneously by multiple instances of the GUI or the GUI and `xvm`(8).)

The **File** menu also lets you close the current window and exit the GUI completely.

Edit Menu

The **Edit** menu lets you expand and collapse the contents of 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 items in the view menu or clear the current selections.

Tasks Menu

The **Tasks** menu contains the following:

- **Guided Configuration** contains the tasks to configure various volume elements
- **Volume elements** contains tasks to create and manage volumes, subvolumes, concats, mirrors, and stripes
- **Disks** contains tasks to label and manage disks and slices
- **Filesystems** contains tasks to define and manage local filesystems
- **Privileges** lets you grant or revoke access to a specific task for one or more users
- **Find Tasks** lets you use keywords to search for a specific task

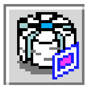
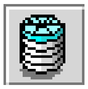
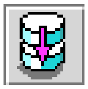
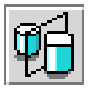
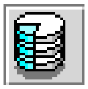
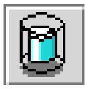
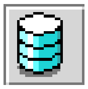
Help Menu

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.

Shortcuts Using Command Buttons

The command buttons along the top of the GUI 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. To override the defaults, launch the task from the **Tasks** menu. Table 10-2 summarizes the shortcuts.

Table 10-2 **XVM Manager** GUI Command Buttons

Button	Task
	Labels selected disks (if the selected disks include foreign and/or labeled disks, the **Label Disks** task will be run)
	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 `data` subvolume with a temporary name
	Creates a volume with a temporary name

Button	Task
	Starts the Performance Co-Pilot XVM I/O monitor `pmgxvm` on the server, displaying via X Windows to your local administration station (for information on using Performance Co-Pilot, see *Performance Co-Pilot for Linux User's and Administrator's Guide*)
	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

View Menu

Choose the item that you want to view from the **View** menu.

Selecting Items to View or Modify

You can use the following methods to select items:

- Click to select one item at a time
- `Shift`+click to select a block of items
- `Ctrl`+click 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.

Viewing Component Details

To view the details on any component, click its name in the view area; see "Selecting Items to View or Modify" on page 186.

The configuration and status details for the component will appear in the details area to the right. At the bottom of the details area will be the **Applicable Tasks** list, which displays tasks you may wish to launch after evaluating the component's configuration details. To launch a task, click the task name; based on the component selected, default values will appear in the task window.

To see more information about an item in the details area, select its name (which will appear in blue); details will appear in a new window. Terms with glossary definitions also appear in blue.

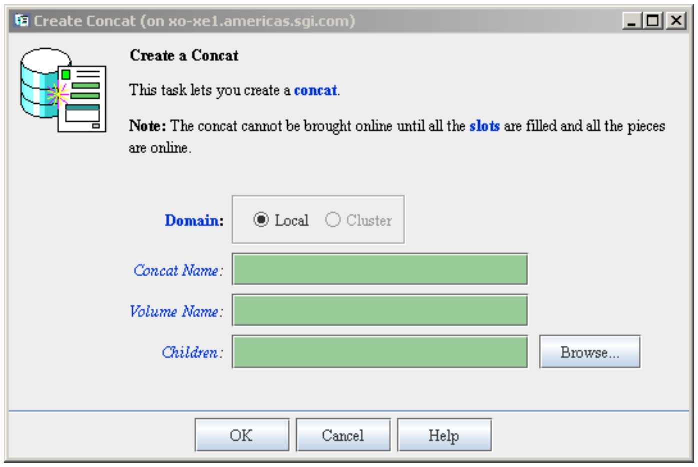
Performing Tasks

To perform an individual task, do the following:

1. Select the task name from the **Task** menu. For example:

   Task > Volume Elements > Create a Concat

   The task window appears. Figure 10-3 shows the task window for the **Create a Concat** task.



Create Concat (on xo-xe1.americas.sgi.com)

Create a Concat

This task lets you create a **concat**.

Note: The concat cannot be brought online until all the **slots** are filled and all the pieces are online.

Domain: ● Local ○ Cluster

Concat Name:

Volume Name:

Children:

Figure 10-3 Create a Concat

As a shortcut, you can right-click an item in the view area to bring up a list of tasks applicable to that item; information will also be displayed in the details area.

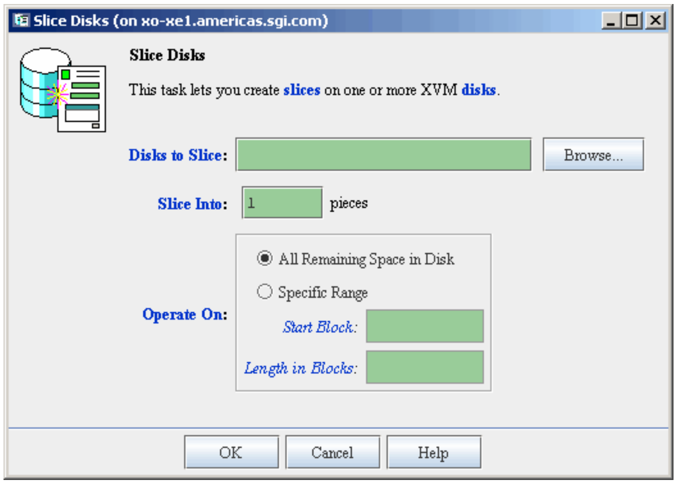
Note: You can click any blue text to get more information about that concept or input field.

---

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 10-4 shows the task window for the **Slice Disks** task.



Slice Disks (on xo-xe1.americas.sgi.com)

Slice Disks

This task lets you create **slices** on one or more XVM **disks**.

Disks to Slice:

Slice Into: 1 pieces

Operate On:

● All Remaining Space in Disk

○ Specific Range

Start Block:

Length in Blocks:

Figure 10-4 Slice Disks

Clicking on the **Browse** button displays a list of available disks to label, as shown in Figure 10-5. In this window, you can enter a text pattern to match.

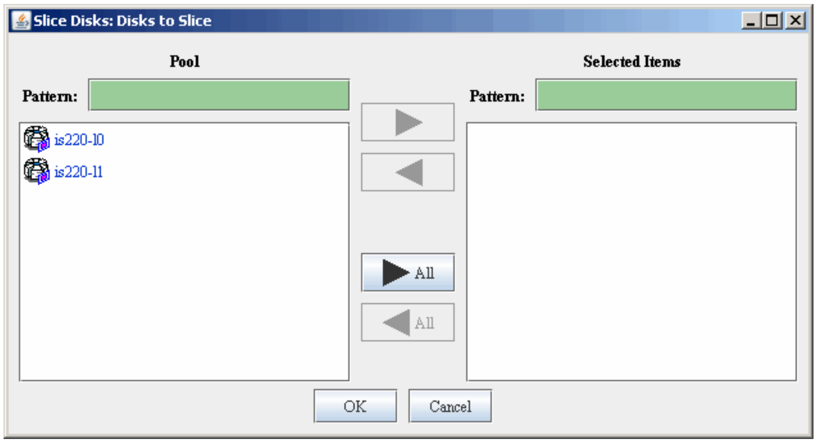


Figure 10-5 Browse Window

Note: In every task, changes do not take effect until you click **OK**.

3. Continue launching tasks as needed.

To perform an individual task, do the following:

Using Drag-and-Drop

The GUI lets you use drag-and-drop to do the following:

- Structure volume topologies
- Administer XVM disks

⚠ **Caution:** Always exercise care when restructuring volume elements with drag-and-drop because data that resides on the volume element can be lost. The GUI attempts to warn the user when it can predict that there is a high likelihood of data loss. However, when a volume is not associated with a mounted filesystem, neither the `xvm` command nor the GUI can determine whether that volume holds important data.

To select multiple GUI icons, select the first icon by clicking the left mouse button, then press the `Ctrl` button while clicking on the additional icons. To select consecutive icons, select the first icon and press shift while selecting the last icon.

You cannot use drag-and-drop between two GUI windows or with shortcut command buttons.

Structuring Volume Topologies

To reconfigure a volume, select an icon and then drag the icon and drop it on another 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 text field.

Configuring Disks

To label or unlabel disks using drag-and-drop, select the following:

View
  > **Disks**

Select an unlabeled disk then drag and drop it on the **Labeled Disks** heading, or select a labeled disk then drag and drop it on the **Unlabeled Disks** heading.

You can give away a disk using the task menu or drag-and-drop. In the **Disks** view, select a disk and then drag and drop it on the **Cluster Disks** heading.

Note: Giving away a disk presents less risk of data loss than stealing a disk.

You can label a disk by clicking a selected (highlighted) disk and typing a name into the resulting name text field.

Displaying State

The GUI shows static and dynamic state. For example, suppose the database contains the static information that a filesystem is enabled for mount; the GUI will display the dynamic information showing one of the following:

- A blue icon indicating that the filesystem is mounted (the static and dynamic states match)
- A grey icon indicating that the filesystem is configured to be mounted but the procedure cannot complete
- An error (red) icon indicating that the filesystem is supposed to be mounted, but it is not (the static and dynamic states do not match, and there is a problem)

Getting More Information

Click blue text to launch tasks or display one of the following:

- Term definitions
- Input instructions
- Item details
- The selected task window

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.

Important GUI and `xvm` Command Differences

When volume elements other than volumes are created or detached, the system automatically creates a volume and a `data` subvolume that are associated with the volume element. You can explicitly name this generated volume, in which case the volume name is stored in label space and persists across 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 as **Unattached**. In contrast, the `xvm` command-line interface (CLI) shows all volumes and subvolumes.

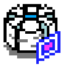
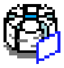
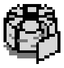
The XVM GUI displays filesystems that are on volumes that were not named explicitly, but lists the volumes as **None**. Volumes and subvolumes that the system generated automatically with temporary names are mentioned in the full paths of **Unattached** volume elements (for example, `/vol96/datav`), but the GUI ignores them otherwise.

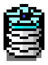
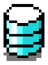
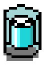
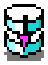
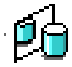
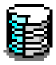
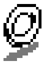
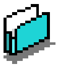
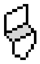
To reduce the risk of data loss, you should name volumes explicitly when using the GUI. If you have created volumes that you did not name explicitly, you can use the `xvm` CLI to assign these volumes permanent names before proceeding.

Key to Icons and States

Table 10-3 shows keys to the icons used in the XVM Manager GUI.

Table 10-3 Key to XVM Manager GUI Icons

Icon	Entity
	XVM disk
	Unlabeled disk
	Foreign disk

Icon	Entity
	Slice
	Volume
	Subvolume
	Concat
	Mirror
	Stripe
	Slot
	Local filesystem
	Expanded tree

Icon	Entity
	Collapsed tree
	Copy on write
	Repository

Table 10-4 shows keys to the states used in the XVM Manager GUI.

Table 10-4 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

Icon	State
	(Orange arrow) Mirror reviving
	Disabled
	(Red icon) Error detected, down or mounted with error

Volume Element Tasks

This section tells you how to perform XVM administrative tasks on volume element.

Note: When running the **XVM Manager** GUI as a standalone product, **Domain** is always set to `local` in these tasks.

- "Create a Concat" on page 196
- "Create a Mirror" on page 197
- "Create a Stripe" on page 198
- "Delete Volume Elements" on page 199
- "Rename a Volume Element" on page 200
- "Insert Mirrors or Concats Above Volume Elements" on page 200
- "Remove Unneeded Mirrors and Concats" on page 201
- "Enable Volume Elements" on page 201
- "Disable Volume Elements" on page 202
- "Bring Volume Elements Online" on page 202

- "Remake a Volume Element" on page 202
- "Detach Volume Elements" on page 203
- "Create a Volume" on page 203
- "Create a Subvolume" on page 204
- "Modify Subvolumes" on page 205
- "Modify Statistics Collection on Volume Elements" on page 206

Create a Concat

A concat is a volume element that concatenates all of its children into one address space.

When you create a concat, the system automatically creates a parent volume and `data` subvolume for the concat. If you explicitly name this generated volume, the volume name is stored in label space and persists across reboots.

Note: To achieve good I/O performance, use the `/etc/failover2.conf` file. See Chapter 6, "XVM Path Failover" on page 107.

To create a concat, do the following:

1. **Domain:** Select 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 children of the new concat, or click the **Browse** button to display a list and select from that list.
5. Click **OK**.

Create a Mirror

A mirror is a volume element that maintains identical data images on its underlying volume elements. A mirror cannot have more than eight legs. You can specify whether a particular leg of a mirror is to be preferred for reading by marking it as a primary leg.

Note: 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 you create a mirror, the system automatically creates a parent volume and `data` subvolume for the mirror. If you explicitly name this generated volume, the name is stored in label space and persists across reboots.

Note: To achieve good I/O performance, use the `/etc/failover2.conf` file. See Chapter 6, "XVM Path Failover" on page 107.

To create a mirror, do the following:

1. **Domain:** Select 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

   Round Robin Balances the I/O load among the legs of the mirror, blindly reading in a round-robin fashion.

   Sequential Routes sequential I/O operations to the same leg
5. **Revive Option:** Select the revive option for the mirror.

   By default, when you create a mirror that has more than one leg, the mirror begins the reviving process; this means that the system begins the process of synchronizing the data in each of the legs. For large mirror components, this revive process may take a long time. See "Avoid Unnecessary Revives " on page 54.

Note: You cannot halt a mirror revive once it has begun except by detaching all but one of the legs of the mirror.

6. **Children:** (*Optional*) Enter one or more children of the new mirror, or click the **Browse** button to display a list 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 distributes a set of volume elements across an address space.

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

The actual size of the stripe 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) and you create a stripe consisting of two slices that are each 256,000 blocks, all of the space on each slice 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 because the same amount of space in each volume element that makes up the slice is used.

Following is the general formula for determining the stripe size, where *stripe_width* is the number of volume elements that make up the stripe (using integer arithmetic):

stripe_size = (*smallest_stripe_leg* / *stripe_unit*) * *stripe_unit* * *stripe_width*

When you create a stripe, the system automatically creates a parent volume and `data` subvolume for the stripe. If you explicitly name this generated volume, the name is stored in label space and persists across reboots.

Note: To achieve good I/O performance, use the `/etc/failover2.conf` file. See Chapter 6, "XVM Path Failover" on page 107.

To create a stripe, do the following:

1. **Domain:** Select 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 children of the new stripe, or click the **Browse** button to display a list and select from that list.
6. Click **OK**.

Delete Volume Elements

When you delete a volume element, its parent remains and has an open slot in the XVM topology tree. An element of an open subvolume can only be deleted if its deletion will not cause the subvolume to go offline. The only element that can be deleted without putting the subvolume offline is a mirror leg that is not the last leg of that mirror.

If a volume element contains any attached children, it cannot be deleted unless you choose to delete all of its descendents, or all of its descendants except slices (detaching and keeping the slices).

To delete one or more volume elements, do the following:

1. **Domain:** Select 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 and select from that list.
3. **Also Delete Descendants:** Select whether to delete no children, the selected volume elements and all of their descendants except slices, or the selected volume elements and all of their descendents including slices.
4. Click **OK**.

Rename a Volume Element

You can rename a volume, subvolume, concat, stripe, or mirror. (You cannot change the name of a slice.) The name you give an object when you explicitly rename it remains persistent across reboots.

To rename a volume element, do the following:

1. **Domain:** Select the domain of the volume element to rename. 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 above another volume element other than 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/concat above an existing volume element, do the following:

1. **Domain:** Select the domain for the new volume element to be inserted.
2. **Volume Elements to Insert Above:** Enter the volume elements above which to insert the mirror/concat or click the **Browse** button to display a list 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 topology 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:** Select the domain for the mirrors and concats to be removed. 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 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:** Select the domain for the volume elements to be enabled. 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 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 you explicitly enable the volume element. An object remains disabled across reboots.

To disable one or more volume elements, do the following:

1. **Domain:** Select the domain for the volume elements to be disabled. 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 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 leg shows an I/O error.

To bring a set of volume elements online, do the following:

1. **Domain:** Select the domain for the volume elements to be brought 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 and select from that list.
3. Click **OK**.

Remake a Volume Element

When you remake a volume element, you collapse holes in the XVM topology tree beneath volume element. The children remain in their current order.

To remake a volume element, do the following:

1. **Domain:** Select 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.

An element of an open subvolume can only be deleted if its deletion will not cause the subvolume to go offline. The only element that can be deleted without putting the subvolume offline is a mirror leg that is not the last leg of that mirror. You cannot detach the last valid leg of an open mirror from that mirror, because this will cause the mirror to go offline.

To detach volume elements, do the following:

1. **Domain:** Select the domain for the volume elements to be detached. 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 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.

A volume cannot have more than one subvolume of a given type. The system-defined subvolumes are `data`, `log`, and `rt`. See "Subvolume" on page 15.

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. See "Explicitly Name Volume Elements" on page 52.
2. **Domain:** Select the domain that will own the volume.

3. **Volume Children:** Select one of the following:
   - **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 concat, mirror, stripe, or slice to the subvolume. The volume element attached to the subvolume cannot be a volume or another subvolume.

There cannot be more than one subvolume with the same type under a volume. For example, there can be only one `data` subvolume under a volume. A user-defined subvolume type is in the range `16` through `255`. You can have multiple user-defined subvolumes under a volume if each has a unique type number. Subvolume types in the user-defined range are not interpreted in any way by the system. See "Subvolume" on page 15.

When you create a subvolume, you specify the user ID of the subvolume owner (default `0`), the group ID of the subvolume owner (default is `0`), and the mode of the subvolume (default is `0644`).

If you do not specify a child, an empty subvolume is created that can be attached to later.

When you create a subvolume, the system automatically creates a parent volume for it. If you explicitly name this generated volume, the name is stored in label space and persists across reboots.

To create a subvolume, do the following:

1. **Domain:** Select the domain that will own the subvolume.
2. **Subvolume Type:** Specify whether the subvolume is a `data`, `log`, `rt` (real-time), 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 permissions mode of the block and character special files corresponding to the subvolume. For more information, see the `chmod`(1) man page.
9. Click **OK**.

Modify Subvolumes

You can change user ID, group ID, and the permissions mode that apply to one or more subvolumes.

Do the following:

1. **Domain:** Select the domain. This selection determines what will appear in the list of available 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 permissions mode of the block and character special files corresponding to the subvolume. For more information, see the `chmod`(1) man page.
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` and you can reset the current statistics to `0`. Statistics are enabled / disabled only for the volume elements that you specify. 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 node only.

To modify statistics collection on one or more volume elements, do the following:

1. **Domain:** Select the domain, which 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**.

Disks Tasks

This section discusses the following:

- "Label Disks" on page 207
- "Slice Disks" on page 208
- "Rename a Disk" on page 209
- "Remove XVM Labels from Disks" on page 209
- "Modify Statistics Collection on Disks" on page 210
- "Give Away Ownership of Disks" on page 210
- "Steal a Foreign Disk" on page 211
- "Probe Disks for Labels" on page 212
- "Dump Volume Element or Physical Volume Structure to File" on page 213

Note: 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

To create XVM logical volumes on a physical disk, you must label the disk as an XVM disk. Labeling a disk writes an XVM physical volume (physvol) label on a disk and allows XVM to control the partitioning on the disk. In a CXFS cluster, any physvols that will be shared must be physically connected to all nodes in the cluster.

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.

⚠ **Caution:** 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 default values for the volume header size and label area size support approximately 5000 XVM objects; this should be sufficient for most volume configurations. If you will have more than that many objects on the physvol that the label area must 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: A volume element and a name for a volume element count as two objects.

If you add a new disk that has already been labeled as a physvol to a running system, you must manually probe the disk with XVM 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, because XVM 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.

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. In the browse window, you can enter a text pattern to match.
2. **Disk Name:** 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:** Deselect this option if you want to label the disk, 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 a physvol. You can specify the starting block and the length of a slice. You can choose one of the following methods

- Create a slice out of all of the blocks of a physvol
- Divide a specified address range into equal parts, with each part a different slice
- Slice multiple physvols at once

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

If you do not supply a slice length, 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 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. Alternatively, 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 from all of the remaining space on the XVM disk or from 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 disk (physvol). The name you give is persistent across reboots. 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 a physvol from a system, you must remove the XVM label from the physvol. After unlabeling, 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 physvol that is not attached to the system you are working from.

You cannot unlabel physvols containing slices unless you also delete the slices on those physvols. 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 fails when you are unlabeling a physvol, the unlabel will fail. If all deletes succeed, the physvol will be unlabeled.

To remove XVM labels from one or more physvols with the GUI, do the following:

1. **Disks to Unlabel:** Enter the physvol 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 for physvols show the number of read and write operations as well as the number of 512-byte blocks read and written. You can turn statistics collection for a physvol to `on` or `off` and you can reset the current statistics to `0`.

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

To modify statistics collection on one or more disks with the GUI, do the following:

1. **Disks to Modify:** Enter the 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

Note: This task is valid in a clustered environment only.

You can gracefully change the ownership of a physvol by giving it to the local domain of another machine (by default) or to another cluster (if you select **Another Cluster**).

Note: However, you cannot give away ownership of a physvol that has slices that are part of open subvolumes. For this reason, an attempt to give away a disk will fail during a mirror revive. In general, you must unmount filesystems on XVM volumes that contain the physvol and wait for mirror revives to complete before giving away a physvol.

Giving a disk away will result in all slices on the disk (and any empty parents that result) being deleted. 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 either automatically on reboot or when the new owner probes the disk.

You can specify a physvol to give away by either the physvol name or the disk name.

To give ownership of one or more disks to another node or cluster, do the following:

1. **Disks to Give Away:** Enter the 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 owning host or cluster.
3. Click **OK**.

Steal a Foreign Disk

Note: This task is valid in a clustered environment only.

In some circumstances, the node or cluster that currently owns the physvol may be unable to give a disk away. In these cases, you can steal a disk to change the ownership of a physvol. Only physvols that are foreign to the current node or cluster can be the targets of a steal.

⚠ **Caution:** Stealing a disk unconditionally resets the owner of a physvol to the current node or cluster. No attempt is made to inform the previous owner of the ownership change. If another host or cluster has the physvol instantiated, configuration corruption or data corruption could occur. You should forcibly steal a disk only when ownership cannot be changed by giving the disk away. In a situation where you must steal a disk, you can determine the ownship by using the `xvm`(8) command `show -v foreign`; see "Foreign Disks" on page 86.

You cannot steal a physvol that has slices that are part of open subvolumes. In general, you must unmount filesystems on XVM volumes that contain the physvol and wait for mirror revives to complete before stealing the physvol.

To take control of a foreign disk, 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**.

Probe Disks for Labels

If you add to a running system a disk that was already labeled as a physvol, you must manually probe the disk in order for the system to recognize it as an XVM disk. (You do not have to do this when you label a new XVM disk because XVM automatically probes the disk as part of the label process.)

Note: It is assumed that the disk to be probed is available in the hardware inventory (that is, the controller that it is connected to has been probed outside of XVM).

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**.

If the disk being probed has not been previously labeled by XVM, an error is returned.

Dump Volume Element or Physical Volume Structure to File

You can dump configuration information for an individual volume element or all of its children. You can also use dump configuration commands for a physvol; you must explicitly dump the physvol topology tree separately from a volume element topology 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, 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:** Select the domain of the object 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 and select from that list.

   If you dump selected volume elements, you can indicate whether to dump the children of the specified volume element as well.
4. **Also Dump Descendants:** If you specify only certain disks to dump, you can also choose to dump their descendants.
5. Click **OK**.

Filesystems Tasks

This section discusses the following:

- "Make Filesystems" on page 214
- "Grow a Filesystem" on page 216
- "Mount a Filesystem Locally" on page 217
- "Unmount a Locally Mounted Filesystem" on page 217
- "Remove Filesystem Mount Information" on page 218

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, log size, and real-time section 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. If the volume contains an `rt` subvolume, the real-time section size will be set to the size of the `rt` 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`(8) 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**: *(Obsolete)*
   - **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. If the volume does not contain a `log` subvolume, the log section will be a portion of the data section on the `data` subvolume. This option only appears if the filesystem has a `log` 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.
   - **Real–Time Section Size**: Enter the size of the real–time section of the filesystem as a number of 512-byte blocks. This value is usually equal to the size of the `rt` subvolume, if there is one. You should specify a size other than `0` only if the real-time section should occupy less space than the size of the `rt` subvolume. This option only appears if the filesystem has an `rt` 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 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 of its ancestor.

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 and real-time section 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. If the volume contains an `rt` subvolume, the real-time section size will be set to the size of the `rt` 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`(8) 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.

- **Real-Time Size**: Enter the size of the real-time section of the filesystem as a number of 512-byte blocks. This value is usually equal to the size of the `rt` subvolume, if there is one. You should specify a size other than `0` only if the real-time section should occupy less space than the size of the `rt` subvolume. This option only appears if the filesystem has an `rt` 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`(8) command. For more information about available options, see the `fstab`(5) 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`(8) 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 the `/etc/fstab` file.

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:

- "Grant Task Access to a User or Users" on page 218
- "Revoke Task Access from a User or Users" on page 220

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.

1. Select the tasks to grant access to, using the above selection methods. Click **Next** to move to the next page.
2. 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.

Grant 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**.

Grant 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 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.

Chapter 11

Troubleshooting

This section discusses the following:

- "Common Problems" on page 223
- "Troubleshooting Strategies" on page 225
- "Reporting Problems to SGI" on page 229

In a CXFS environment, also see the troubleshooting information in the *CXFS 7 Administrator Guide for SGI InfiniteStorage*.

Common Problems

- "Cannot Switch Domains" on page 223
- "Need to Steal" on page 224
- "Disk is Both Owned and Foreign" on page 224
- "Mirror Revives on Recovery in a Cluster" on page 224
- "Slow Mirror Revives" on page 225
- "Volume Element in `inconsistent` State" on page 225

Cannot Switch Domains

To enter to the cluster domain, the appropriate CXFS services must be started. See "CXFS Service Requirements for Cluster Domain" on page 26.

If you execute the `xvm`(8) command when the services are started, you will automatically enter the cluster domain. If the services are not started, you will automatically enter the local domain.

If you try to explicitly enter cluster domain when the services are not started, you will get an error:

```
# xvm -d cluster
could not start up in the specified domain:  domain name is invalid or unavailable
```

See:

- "Local Domain and Cluster Domain" on page 26
- "Interactive Use" on page 64

Need to Steal

If a cluster or host that currently owns the disk does not exist, you must perform a steal operation. This situation might occur after a mistaken give operation or after deleting a host or cluster.

See:

- "Give Rather than Steal Ownership" on page 60
- "Gracefully Transferring Ownership of a Disk or Physvol with the give Command" on page 90
- "Unmount Filesystems Before Making XVM Topology Changes" on page 60

Disk is Both Owned and Foreign

If the steal command is used to take a disk from a running system, the configuration can become inconsistent and the disk may appear as both owned and foreign. You can use the reprobe command to recover from this situation; see "Removing Configuration Information with the reprobe Command" on page 105.

To avoid problems like this, see "Give Rather than Steal Ownership" on page 60.

Mirror Revives on Recovery in a Cluster

When a node in a CXFS cluster crashes, a mirror 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 legs of the mirror unequal. When this occurs, XVM must forcibly resynchronize all of the legs. This can be a lengthy process.

Slow Mirror Revives

If your system performance of mirror revives seems slow, you may need to reconfigure the mirror revive resources set by the `xvm_max_revive_rsc` and `xvm_max_revive_threads` XVM system tunable kernel parameters. See:

- "Set Mirror Revive Resources Properly" on page 54
- "Viewing XVM Parameters with `modinfo`" on page 235

Volume Element in `inconsistent` State

If a mirror leg is disconnected while a change is occurring, the leg or the parents or children of the mirror may temporarily display an `inconsistent` state in the `xvm` CLI `show` command output. This is expected behavior and does not require any administrative action.

If you notice an `inconsistent` state that is not associated with a mirror, contact SGI Support for assistance.

Troubleshooting Strategies

This section discusses the following:

- "Returning to Preferred Path" on page 225
- "Switching Domains to Find All Objects" on page 226
- "Using the System Dump Analysis Tool " on page 227
- "Using SGI Knowledgebase" on page 229

Returning to Preferred Path

If a hardware problem causes the system to switch the path to an XVM physvol, you can run the following command to return to the preferred path after solving the hardware issue:

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

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

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

See:

- "Change Affinity Consistently Across the Cluster" on page 59
- "Switching to a New Device" on page 122

Switching Domains to Find All Objects

The objects displayed and acted on by `xvm` commands depend upon the domain setting; if you are in the cluster domain, only objects owned by the cluster will be shown.

For example, the following output displays the scenario where you must switch from the cluster domain to the local domain in order to see the physvol `lp`:

```
xvm:cluster> show phys/*
phys/first              2339536896 online,cluster,accessible
phys/fourth             2339536896 online,cluster,accessible
phys/second             2339536896 online,cluster,accessible
xvm:cluster> set domain local
xvm:local> show phys/*
phys/lp                 11112861696 online,local,accessible
```

For more information, see:

- "Local Domain and Cluster Domain" on page 26
- "Invoking the `xvm` CLI" on page 63
- "Changing the Current Domain with the `set` Command" on page 82

Using the System Dump Analysis Tool

For system dump analysis, use the `crash`(8) tool provided with SLES or RHEL.

To enable the collection of crash dumps, do the following:

1. Install the following RPMs, where *kernelrev* matches your installed kernel:

   - RHEL:
     - `kernel-debuginfo-`*kernelrev*
     - `kernel-debuginfo-common-`*kernelrev*
     - `system-config-kdump-`*kernelrev*
   - SLES:
     - `kernel-default-debuginfo-`*kernelrev*
     - `kdump-`*version*
     - `kexec-tools-`*version*

   For example, for the SLES `2.6.27.19-5-default` kernel, you would require `kernel-default-debuginfo-2.6.27.19-5.1` RPM, which would install the following file:

   ```
   /usr/lib/debug/boot/vmlinux-2.6.27.19-5-default.debug
   ```

   When you install the RHEL `kdump-`*version* RPM, it will automatically add the following information onto the `kernel` lines in the `/boot/grub/menu.lst` file:

   ```
   crashkernel=256m-:128M@16M
   ```

   Note: When you install the `kdump` RPM, `kdump` is automatically enabled.

2. Reboot, which activates the kernel and reserves the required memory. You will see the following message on the console:

   ```
   Loading kdump                                        done
   ```

3. Verify that the machine is set up correctly by requesting an NMI from the console:

   ```
   console# echo "c">/proc/sysrq-trigger
   ```

Note: If there are several old dump files, the oldest one might be deleted by this process.

For example:

```
console# echo "c">/proc/sysrq-trigger
SysRq : Trigger a crashdump
Initializing cgroup subsys cpuset
Initializing cgroup subsys cpu
...
(pages of output)
```

The key piece of information to look for are lines such as the following at the end of the output:

```
Saving dump using makedumpfile
-------------------------------------------------------------------------------
Copying data                       : [ 100 %]

The dumpfile is saved to /root/var/crash/2009-10-28-13:05/vmcore.

makedumpfile Completed.
-------------------------------------------------------------------------------
Generating README              Finished.
Copying System.map             Finished.
Copying kernel                 Finished.
Copying kernel.debug           Finished.
```

Then the machine will reboot normally.

4. Go to the /var/crash directory and look for the dump directories that named according to the date and time. Each date directory will contain the files required for analysis. For example:

   ```
   # cd /var/crash
   console# ls
   2009-10-13-21:02/  2009-10-26-15:55/
   # ls -1 2009-10-26-15:55
   README.txt
   System.map-2.6.27.19-5-default
   vmlinux-2.6.27.19-5-default.debug
   vmlinux-2.6.27.19-5-default.gz
   vmcore
   ```

For more information, see the crash(8) man page.

Note: The sgidb tool is supported for systems that have CXFS installed.

Using SGI Knowledgebase

If you encounter problems and have an SGI support contract, you can log on to Supportfolio and access the Knowledgebase tool to help find answers.

To log in to Supportfolio Online, see:

https://support.sgi.com/login

Then click on **Search the SGI Knowledgebase** and select the type of search you want to perform.

If you need further assistance, contact SGI Support.

Reporting Problems to SGI

Before reporting a problem to SGI, you should do the following:

- Retain any messages that appeared in the system logs immediately before the system exhibited the problem.
- If there was system crash, obtain a system dump if possible.

- If you suspect that there are problems with a particular client (such as if you see error messages indicating that a client is timing out), collect a system dump if possible or else a kernel-stack backtrace for all threads from that client. If it is unclear which client is causing the problem, collect this information for all clients.

  For Linux systems, you can obtain kernel-stack backtraces of all threads by running the following `crash`(8) command:

  ```
  foreach bt > somefile
  ```

  For more information, see the `crash`(8) man page.

  Note: Normally, you can use the `crash` command to get the kernel-stack backtraces on a running system (without having to first crash the system in order to get a dump).

- If XVM has cluster problems, you should also provide system logs and systems dumps from other nodes in the cluster. This can be helpful if you encounter problems such as the following:

  - The `xvm` command will not enter cluster domain
  - You cannot mount a CXFS filesystem
  - You see messages about a client not responding

- If your system is set up with KDB, retain the debugger information from the KDB built-in kernel debugger after a system kernel panic.

- Provide the core files and the following associated information:

  - The application that created the core file:

    ```
    file core_filename
    ```

  - The binaries listed by the following command:

    ```
    ldd application_path
    ```

- Gather the XVM subsystem parameters by executing the following:

  ```
  xvm:cluster> show -subsystem
  ```

For example:

```
xvm:local> show -subsystem
XVM Subsystem Information:
--------------------------
apivers:              19
config gen:           15
privileged:           1
clustered:            0
cluster initialized:  0
```

Parameter	Description
`apivers`	The version of the application programming interface (API) that XVM is using. All nodes in the cluster must use the same version.
`config gen`	A generation number that increments every time the XVM configuration in the kernel is changed.
`privileged`	Indicates whether the current invocation of the `xvm` CLI is capable of making configuration changes (`1`) or not (`0`).
`clustered`	Indicates whether the kernel is cluster-aware (`1`) or not (`0`).
`cluster initialized`	Indicates whether the CXFS cluster services that allow XVM to operate in the cluster domain have been initialized (`1`) or not (`0`).

XVM System Tunable Kernel Parameters

This appendix discusses the following:

- "Overview of the XVM System Tunable Kernel Parameters" on page 233
- "Static Parameters that are Site-Configurable" on page 236
- "Dynamic Parameters that are Site-Configurable" on page 239

Overview of the XVM System Tunable Kernel Parameters

This section discusses the following:

- "Using Appropriate Parameter Settings" on page 233
- "Making Permanent Parameter Changes" on page 234
- "Making Temporary Parameter Changes" on page 234
- "Querying a Current Parameter Setting" on page 235
- "Viewing XVM Parameters with `modinfo`" on page 235

Using Appropriate Parameter Settings

SGI recommends that you use the same settings on all applicable nodes in the cluster.

Note: Before changing any parameter, you should understand the ramifications of doing so on your system. You should change debugging parameters only at the recommendation of SGI Support.

The values of these parameters vary in different releases of the product. When upgrading the product, consult SGI Support to determine whether any changes made to the parameters in this chapter should be carried forward. Setting these parameters incorrectly may render the system unstable or otherwise unusable.

Making Permanent Parameter Changes

You can set a parameter permanently across reboots by adding it to the `/etc/modprobe.d/sgi-xvm.conf` file. Use the following format:

`options` *module systune*=*value*

where:

- *module* is one of the following:

  ```
  sgi-pm
  sgi-xvm
  ```

  The section that describes a parameter lists the module name.

- *systune* is the parameter name, such as `xvm_congestion_check`
- *value* is the value you want to set for the parameter, such as `2`

Note: Do not use spaces around the = character.

There should be only one `options` line per module; if you want to specify multiple parameters, you must place them all on that single line.

For example, to permanently set the `xvm_congestion_check` parameter (which is in the `xvm` module) to `1`, add the following line to `/etc/modprobe.d/sgi-xvm.conf`:

```
options sgi-xvm xvm_congestion_check=1
```

The change will take effect upon reboot.

Making Temporary Parameter Changes

For a temporary change to a dynamic parameter, use the Linux `sysctl`(8) command as follows:

# **`sysctl`** ***prefix*.*systune*=*value***

where:

- *prefix* is one of the following:

  ```
  dev.pathmgr
  dev.xvm
  ```

- *systune* is the parameter name, such as `xvm_congestion_check`
- *value* is the value you want to set for the parameter, such as `1`

Note: Do not use spaces around the = character.

For example, to temporarily set the `xvm_congestion_check` parameter (which has the `dev.xvm` prefix) to `1`, enter the following:

```
# sysctl dev.xvm.xvm_congestion_check=1
dev.xvm.xvm_congestion_check = 1
```

Querying a Current Parameter Setting

To query the current setting of a parameter, use the Linux `sysctl`(8) command:

```
# sysctl prefix.systune
```

where:

- *prefix* is one of the following:

  ```
  dev.pathmgr
  dev.xvm
  ```

- *systune* is the parameter name, such as `xvm_congestion_check`

For example, to query the current setting of the `xvm_congestion_check` parameter (which has the `dev.xvm` prefix):

```
# sysctl dev.xvm.xvm_congestion_check
dev.xvm.xvm_congestion_check = 1
```

Viewing XVM Parameters with `modinfo`

You can execute the `modinfo`(8) command on the XVM module to view descriptions of the XVM tunable parameters, along with their minimum, maximum, and default values.

Static Parameters that are Site-Configurable

Static parameters require a reboot to take affect. This section discusses the following:

- "xvm_defeat_stripe_unit_in_min_io" on page 236
- "xvm_congestion_check" on page 236
- "xvm_max_revive_rsc" on page 237
- "xvm_max_revive_threads" on page 237
- "xvm_mr_daemon_max" on page 237
- "xvm_mr_daemon_min" on page 238

xvm_defeat_stripe_unit_in_min_io

Specifies whether XVM passes the stripe unit or the hardware sector size to XFS or another user via the queue limits field min_io.

Range of values:

- 0 causes min_io to pass the stripe unit (default)
- 1 causes min_io to pass the hardware sector size

Prefix: dev.xvm

Module: sgi-xvm

xvm_congestion_check

Specifies whether or not a congestion query from the filesystem will propagate to all LUNs that make up the XVM volume.

Range of values:

- 0 disables congestion queries (default)
- 1 enables congestion queries

Caution: Enabling congestion check can negatively impact I/O performance.

Prefix: `dev.xvm`

Module: `sgi-xvm`

xvm_max_revive_rsc

Specifies the data throughput per thread using central memory. Increasing this parameter may decrease performance of other jobs running on the system.

Range of values:

- Default: `5`
- Minimum: `1`
- Maximum `10`

Prefix: `dev.xvm`

Module: `sgi-xvm`

xvm_max_revive_threads

Specifies the number of parallel I/O processes used in reviving mirrors. Increasing this parameter may decrease the I/O performance of other tasks on the system.

Range of values:

- Default: `4`
- Minimum: `1`
- Maximum `12`

Prefix: `dev.xvm`

Module: `sgi-xvm`

xvm_mr_daemon_max

Defines the maximum number of mirror daemons started for processing mirror retries and mirror completions that cannot be done from completion threads.

Caution: Adjusting the values can keep more threads alive in the pool rather than spawning and exiting threads, which can cause performance issues.

Range of values:

- Default: `25`
- Minimum: `0`
- Maximum: `255`

Prefix: `dev.xvm`

Module: `sgi-xvm`

`xvm_mr_daemon_min`

Defines the minimum number of mirror daemons started for processing mirror retries and mirror completions that cannot be done from completion threads.

Caution: Adjusting the values can keep more threads alive in the pool rather than spawning and exiting threads, which can cause performance issues.

Range of values:

- Default: `1`
- Minimum: `0`
- Maximum: `255`

Prefix: `dev.xvm`

Module: `sgi-xvm`

Dynamic Parameters that are Site-Configurable

Dynamic parameters take affect as soon as they are changed. This section discusses the following:

- "`pm_display_io_activity`" on page 239
- "`pm_small_io_size_limit`" on page 240

`pm_display_io_activity`

(Linux systems only) Specifies whether or not recent I/O activity (the amount of data transferred and the number of I/O operations for a 1-second period) is reported in the output of the `xvm show` command. The information is reported using the following format:

[*nnn* KB *nnn* ops]

Note: Enabling this parameter will decrease throughput (IOPs/second) by a small percentage. Therefore, you should enable it only as a configuration aid and disable it for production use.

Range of values:

- `0` disables the report of I/O activity in the `show` output (default)
- `1` enables the report of I/O activity in the `show` output

Prefix: `dev.pathmgr`

Module: `sgi-pm`

pm_small_io_size_limit

Note: This parameter has an effect only on NUMAlink systems such as SGI UV or SGI UV 2000.

Specifies the number of bytes of I/O data below which the I/O is directed to a path close to the issuing thread, rather than a path close to the data.

Range of values:

- Default: `32768`
- Minimum: `0`
- Maximum: `1073741824`

Prefix: `dev.pathmgr`

Module: `sgi-pm`

Converting an DVH Label to a GPT Label

This appendix discusses the following:

- "Comparison of DVH and GPT" on page 241
- "GPT Conversion Considerations" on page 243
- "Example Format Conversion" on page 243

Comparison of DVH and GPT

Table B-1 shows the format of an XVM DVH-labeled disk.

Table B-1 DVH-Labeled Disk Layout

Block	Size in blocks	Contents
`0`	1	Label (such as partition information)
`1`	3071	Usually unused
`3072`	1024	XVM metadata
`4096`	*data_length*	User data
	unused_length	Unused space due to 2–MB rounding

Table B-2 shows example format of an XVM GPT-labeled disk.

Note: Although partition 1 could start at block 34, for simplicity the example uses block 40. The first GPT partition for the XVM metadata must start within the first 64 blocks of the start of the disk.

Table B-2 GPT-Labeled Disk Layout

Block	Size in blocks	Contents
0	1	Master boot record
1	1	GUID partition table header
2	32	GUID partition entries
40	*data_length*	Partition 1: XVM metadata (approximately 2 MB) followed by user data
End - *33*	32	Backup GUID partition entries
End - *1*	1	Backup GUID partition table header

A DVH-labeled disk can have data going all the way to the end of the disk but a GPT-labeled disk must have up to 33 blocks at the end of the disk (the normal partitioning process should automatically leave the required space at the end of the disk, and XVM will not try to use it):

- 1 block for the backup GUID partition table header
- Up to 32 blocks for the backup GUID partition entries

Therefore, there must be almost 33 blocks (17 KB if 512–bytes/block) free at the end of a disk so that there is room for the backup GPT information. 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. This is not normally the case, but can happen 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.

GPT Conversion Considerations

To convert to GPT format, note the following:

- *start_of_userdata* is the block number where the user data started
- *end_block_number* is the ending block number of the user data
- *data_length* is the number of blocks of user data

If you do not have 17 KB free at the end of the disk, you can still convert, but it will be more complicated because you must first free 17 KB from the end of the disk. Because the DVH user data partition is using this space, you should move the last 512 KB (or whatever size greater than 17 KB that you want) into the first 4096 blocks of the DVH label that is not used, say starting at block 3072 for simplicity.

Example Format Conversion

Suppose that you have a DVH disk layout shown in Table B-3 to the GPT label format shown in Table B-4 by using Procedure B-1 on page 244.

Table B-3 DVH Example

Block	Size (in blocks)	Contents
`0`	1	Label - such as partition information
`1`	3071	Usually unused
`3072`	1024	XVM metadata
`4096`	*data_length*	User data

Table B-4 GPT Example

Block	Size (in blocks)	Contents
0	1	Master boot record
1	1	GUID partition table header
2	32	GUID partition entries
40	*data_length*	XVM metadata (approximately 1.5 MB) followed by user data
3072	1024	512 KB of data copied from end of disk (make this slice0)
4096	*data_length* - 1024	*original_data* – 256 KB at end that was moved (make this slice1)
end - 33	33	Backup GUID partition entries and header

The following procedure walks through the steps required to move some user data from the end of the LUN address space (where the GPT backup label will be placed) to a place in the front part of the address space once used by the DVH label but not needed by the GPT label.

Procedure B-1 Converting a DVH Label to a GPT Label

1. Gather information about the original physvol using the xvm show command and parted. For example, for a volume named volume_B4:

```
petrel:~ # xvm show -v volume_B4
XVM physvol phys/volume_B4
=========================
size: 419425280 blocks  sectorsize: 512 bytes  state: online,local,accessible
uuid: 36d4ea9f-9a77-4d06-9436-d6abede944f8
system physvol:  no
path manager device:  /dev/pm/SGI-IS300--2-600a0b80002fc28100000856489224f2 on host petrel
using paths:
 /dev/xscsi/pci-0000:0b:00.0-sas-0x500a0b82fc281004-lun1 <sdc 8:32> affinity=none ws
Disk has the following XVM label:
  Clusterid:  0
  Host Name:  petrel
  Disk Name:  volume_B4
  Magic:  0x786c6162 (balx)     Version 2
  Uuid: 36d4ea9f-9a77-4d06-9436-d6abede944f8
  last update:  Thu Mar  7 13:45:31 2013
  state: 0x91<online,local,accessible> flags: 0x0<idle>
  secbytes:  512
  label area:  1024 blocks starting at disk block 4096 (10 used)
  user area:   419425280 blocks starting at disk block 5120

Physvol Usage:
Start        Length       Name
---------------------------------------------------
0            419425280    slice/volume_B4s0

Local stats for phys/volume_B4 since being enabled or reset:
------------------------------------------------------------------------------
stats collection is not enabled for this physvol
```

```
petrel:~ # parted /dev/sdc
GNU Parted 2.3
Using /dev/sdc
Welcome to GNU Parted! Type ’help’ to view a list of commands.
(parted) unit s
(parted) print
Model: SGI IS300 (scsi)
Disk /dev/sdc: 419430400s
Sector size (logical/physical): 512B/512B
Partition Table: dvh

Number  Start  End    Size   Type      File system  Name    Flags
 9      0s     7167s  7168s  extended
17      2s     2s     1s     logical                backvh

(parted) quit
```

This output provides the following information:

- The LUN size in sectors (from `parted`): `419430400`
- *start_of_userdata* (block number where the user data started): `5120`
- *data_length* (the number of blocks of user data): `419425280`
- *end_block_number* is `419430400` (that is, `419425280 + 5120 = 419430400`)

Note: If the *end_block_number* is less than the LUN size by more than 33 blocks, there is room for the backup GPT label and no data copying is necessary.

For this example, the user area runs all the way to the end of the LUN and the slice runs all the way to the end of the user area, therefore some data copying will be necessary.

2. Copy 1024 sectors to the space that will be unused by the GPT label:

   - Source: start `419430400 - 1024 = 419429376`
   - Destination: start `5120 - 1024 = 4096`

   Make sure that the volume is unmounted and use the `dd`(8) command to copy the data:

   ```
   petrel:~ # dd if=/dev/pm/SGI-IS300--2-600a0b80002fc28100000856489224f2 \
                 of=/dev/pm/SGI-IS300--2-600a0b80002fc28100000856489224f2 \
                 skip=419429376 seek=4096 count=1024
   1024+0 records in
   1024+0 records out
   524288 bytes (524 kB) copied, 0.0860166 s, 6.1 MB/s
   ```

3. Use `parted` to put a GPT label and partition on the disk, ignoring the warnings:

```
petrel:~ # parted /dev/pm/SGI-IS300--2-600a0b80002fc28100000856489224f2
GNU Parted 2.3
Using /dev/pm/SGI-IS300--2-600a0b80002fc28100000856489224f2
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) mklabel gpt
Warning: The existing disk label on /dev/pm/SGI-IS300--2-600a0b80002fc28100000856489224f2 will be
destroyed and all data on this disk
will be lost. Do you want to continue?
Yes/No? yes
(parted) unit s
(parted) mkpart primary 40 419429376
Warning: The resulting partition is not properly aligned for best performance.
Ignore/Cancel? i
Error: Error informing the kernel about modifications to partition
/dev/pm/SGI-IS300--2-600a0b80002fc28100000856489224f2_part1 --
Invalid argument.  This means Linux won't know about any changes you made to
/dev/pm/SGI-IS300--2-600a0b80002fc28100000856489224f2_part1
```

```
until you reboot -- so you shouldn’t mount it or use it in any way before rebooting.
Ignore/Cancel? i
(parted) print
Model: Unknown (unknown)
Disk /dev/pm/SGI-IS300--2-600a0b80002fc28100000856489224f2: 419430400s
Sector size (logical/physical): 512B/512B
Partition Table: gpt

Number  Start  End         Size        File system  Name     Flags
 1      40s    419429376s  419429337s               primary

(parted) quit
Information: You may need to update /etc/fstab.
```

Note: The above example starts the GPT partition (*start_of_partition1*) at sector 40, but any value between 34 (the end of the default GPT label) and 63 (required by XVM to speed up probing) could have been chosen.

4. Calculate the length of the XVM label area:

   start_of_userdata - (start_of_partition1 + 1) = XVM_label_area_length

   In this case, given that partition 1 starts at sector 40 as noted in the previous step:

   ```
   4096 - 41 = 4055
   ```

   The user data area should start at the beginning of the piece of user data that was copied from the end of the LUN, at sector 4096. The user data area starts immediately after the XVM label area. In this case, the XVM label area is 4055 blocks.

5. Add an XVM label that is compatible with the new GPT label, specifying the length of the XVM label area:

```
server# xvm label -name volname -use-exclusion-zones -xvmlabelblks XVM_label_area_length
```

For example, showing the results:

```
petrel:~ # xvm label -name volume_B4 -use-exclusion-zones -xvmlabelblks 4055
/dev/pm/SGI-IS300--2-600a0b80002fc28100000856489224f2
volume_B4
Performing automatic probe for alternate paths.
Performing automatic path switch to preferred path for phys/volume_B4.
```

```
petrel:~ # xvm show -v volume_B4
XVM physvol phys/volume_B4
=========================
size: 419425281 blocks  sectorsize: 512 bytes  state: online,local,accessible
uuid: 97e1a578-91aa-47b7-aa52-697e572837fb
system physvol:  no
path manager device:  /dev/pm/SGI-IS300--2-600a0b80002fc28100000856489224f2 on host petrel
using paths:
 /dev/xscsi/pci-0000:0b:00.0-sas-0x500a0b82fc281004-lun1 <sdc 8:32> affinity=none ws
Disk has the following XVM label:
  Clusterid:  0
  Host Name:  petrel
  Disk Name:  volume_B4
  Magic:  0x786c6162 (balx)     Version 2
  Uuid: 97e1a578-91aa-47b7-aa52-697e572837fb
  last update:  Thu Mar  7 15:05:05 2013
  state: 0x91<online,local,accessible> flags: 0x0<idle>
  secbytes:  512
  label area:  4055 blocks starting at disk block 41 (10 used)
  user area:   419425281 blocks starting at disk block 4096

Physvol Usage:
Start        Length       Name
---------------------------------------------------
0            419425281    (unused)

Local stats for phys/volume_B4 since being enabled or reset:
------------------------------------------------------------------------------
stats collection is not enabled for this physvol
```

6. Create two slices:

   - The first slice is the copied data, or the first 1024 blocks in the user data (length 1024)
   - The second slice is the rest of the partition or the *user_data_length* - 1024 (length 419425280 - 1024 = 419424256)

   For example, showing the results:

```
petrel:~ # xvm slice -start 0 -length 1024 phys/volume_B4
</dev/lxvm/volume_B4s0>  slice/volume_B4s0
```

```
petrel:~ # xvm slice -start 1024 -length 419424256 phys/volume_B4
</dev/lxvm/volume_B4s1>  slice/volume_B4s1

petrel:~ # xvm show -v volume_B4
XVM physvol phys/volume_B4
=========================
size: 419425281 blocks  sectorsize: 512 bytes  state: online,local,accessible
uuid: 97e1a578-91aa-47b7-aa52-697e572837fb
system physvol:  no
path manager device:  /dev/pm/SGI-IS300--2-600a0b80002fc28100000856489224f2 on host petrel
using paths:
 /dev/xscsi/pci-0000:0b:00.0-sas-0x500a0b82fc281004-lun1 <sdc 8:32> affinity=none ws
Disk has the following XVM label:
  Clusterid:  0
  Host Name:  petrel
  Disk Name:  volume_B4
  Magic:  0x786c6162 (balx)     Version 2
  Uuid: 97e1a578-91aa-47b7-aa52-697e572837fb
  last update:  Thu Mar  7 15:05:05 2013
  state: 0x91<online,local,accessible> flags: 0x0<idle>
  secbytes:  512
  label area:  4055 blocks starting at disk block 41 (10 used)
  user area:   419425281 blocks starting at disk block 4096

Physvol Usage:
Start        Length       Name
---------------------------------------------------
0            1024         slice/volume_B4s0
1024         419424256    slice/volume_B4s1
419425280    1            (unused)

Local stats for phys/volume_B4 since being enabled or reset:
------------------------------------------------------------------------------
stats collection is not enabled for this physvol
```

Note: You can label, unlabel, create slices, and delete slices until you have the correct configuration.

7. Concatenate the copied data to the end of the original data (minus the copied data) to get a volume back that will exactly match the beginning volume. For example, showing the results:

```
petrel:~ # xvm concat -volname testvol slice/volume_B4s1 slice/volume_B4s0
</dev/lxvm/testvol>  concat/concat1

petrel:~ # xvm show -t vol/testvol
vol/testvol                       0 online,accessible
    subvol/testvol/data       419425280 online,accessible
        concat/concat1            419425280 online,tempname,accessible
            slice/volume_B4s1         419424256 online,accessible
            slice/volume_B4s0              1024 online,accessible
```

XVM Glossary

ALUA

Asymmetric logical unit access, a feature of some RAID devices that permits automatic path failover

child

A logical volume element beneath another volume element in the XVM topology tree hierarchy, also known as a *piece*. A mirror child is known as a *leg*.

CXFS

SGI clustered filesystem product

cluster domain

XVM operating mode in which an XVM physical volume (physvol) is owned by a CXFS cluster and it can be controlled by any of the nodes in that cluster. Cluster domain on a server-capable administration node requires that the appropriate CXFS services are started (see "CXFS Service Requirements for Cluster Domain" on page 26). See also *local domain*.

CXFS cluster services

The enabling/disabling of a node, which changes a flag in the cluster database. This disabling/enabling does not affect the daemons involved. The daemons that control CXFS cluster services are `clconfd` on a server-capable administration node and `cxfs_client` on a client-only node.

concat

Concatenated volume element, which combines other volume elements so that their storage is combined into one logical unit.

data subvolume

Default subvolume for storing the contents of the volume.

domain

The mode in which XVM is operating, which affects the ownership and administration of physvols. See *cluster domain* and *local domain*.

foreign disk

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

GUI

Graphical user interface, either **XVM Manager** in the local domain or **CXFS Manager** in the cluster domain

leg

One child of a mirror, on which a volume is replicated.

local domain

XVM operating outside of a CXFS cluster or in which an XVM physvol is owned by a single node and it can be controlled only by that node. See also *cluster domain*.

logical volume

See *volume*.

log subvolume

Optional subvolume intended for faster storage for the filesystem journaling which may improve the performance of metadata-intensive operations.

LUN

Logical unit, the virtual disk space which can be labeled by XVM to form a physvol. There may be multiple paths to a given LUN.

non-ALUA RAID

A RAID device that does not have ALUA and therefore requires the `/etc/failover2.conf` file to provide path failover

physical volume

See *physvol*.

physvol

A disk or LUN that has been labeled for use by XVM is a physical volume element, abbreviated to *physvol*. A physvol with a cluster domain is owned by a CXFS cluster, while a physvol with a local domain is owned by a single node.

piece

A volume element beneath another volume element in the XVM topology tree, also known as a *child*. A mirror piece is known as a *leg*.

piece number

The number of a child according to its position in the XVM topology tree, beginning with `0`,which can be used to specify the object in `xvm` commands. See also *piece*, *rightmost child*, and *leftmost child*.

RAID

Redundant array of independent disks

rightmost child

The first child of a given volume element in the XVM topology tree, having a piece number of `0`. See also *piece*, *piece number*, and *leftmost child*.

leftmost child

The last child of a given volume element in the XVM topology tree. For example, if a volume element has three children, the leftmost child would have a piece number of `2`. See also *piece*, *piece number*, and *rightmost child*.

safe command

An XVM command that does not affect the address space of a subvolume, such as detaching or deleting all but the last child of a mirror. See also *unsafe command*.

straddled operations

Operations that cross the boundary between one child and the next.

stripe

A volume element composed of multiple chunks alternating across the address space to allow parallel I/O operations for higher performance.

mirror

A volume element that maintains identical data images on its underlying volume elements, useful when data redundancy and availability are desired

revive

The process of synchronizing data on the legs of a mirror.

subvolume

A volume element that can be composed of any combination of a stripes, mirrors, concats, and slices.

topology tree

The hierarchy of volume elements that compose a volume.

unlabeled disk

A disk that has not been labeled as an XVM disk.

unsafe command

An XVM command that affects the address space of a subvolume, such as detaching or deleting a child of a concat. See also *safe command*.

VE

See *volume element*.

volume

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

volume element

A building block of an XVM logical volume topology. The following are all volume elements:

- XVM volume
- subvolume
- Concatenated volume element (*concat*)
- Mirror
- Stripe
- Slice

volume width

The number of volume elements that make up the widest layer in the topology tree of a volume.

working set

The set of paths that are currently used for I/O.

xvm CLI

The XVM command line interface, available via the `xvm`(8) command and subcommands

XVM object

An unlabeled disk, physvol, foreign disk, or volume element.

XVM Manager

The XVM graphical user interface, available via the `xvmgr`(8) command.

Index