

EVMS User Guide

Joy Goodreau

IBM

Kylie Smith

IBM

Christine Lorenz

IBM

Copyright © 2003 IBM

March 31, 2003

Special Notices

The following terms are registered trademarks of International Business Machines corporation in the United States and/or other countries: AIX, OS/2, System/390. A full list of U.S. trademarks owned by IBM may be found at <http://www.ibm.com/legal/copytrade.shtml>.

Intel is a trademark or registered trademark of Intel Corporation in the United States, other countries, or both.

Windows is a trademark of Microsoft Corporation in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds.

Other company, product, and service names may be trademarks or service marks of others.

UNIX is a registered trademark of The Open Group in the United States and other countries.

This document is provided "AS IS," with no express or implied warranties. Use the information in this document at your own risk.

License Information

This document may be reproduced or distributed in any form without prior permission provided the copyright notice is retained on all copies. Modified versions of this document may be freely distributed provided that they are clearly identified as such, and this copyright is included intact.

Table of Contents

<u>Preface</u>	1
<u>Chapter 1. What is EVMS?</u>	3
<u>1.1. Why choose EVMS?</u>	3
<u>1.2. The EVMS user interfaces</u>	3
<u>1.3. EVMS terminology</u>	4
<u>1.4. What makes EVMS so flexible?</u>	5
<u>1.5. Plug-in layer definitions</u>	5
<u>Chapter 2. Downloading and installing EVMS</u>	8
<u>2.1. Preparing your system</u>	8
<u>2.1.1. Choosing capabilities</u>	8
<u>2.1.2. Obtaining the Device Mapper and a kernel source tree</u>	8
<u>2.2. Downloading EVMS</u>	8
<u>2.3. Patching the kernel</u>	9
<u>2.4. Configuring the kernel</u>	9
<u>2.5. Building and installing the new kernel</u>	11
<u>2.6. Installing the cluster manager</u>	12
<u>2.7. Installing the Engine</u>	12
<u>2.7.1. Installing from source</u>	12
<u>2.7.2. Installing the EVMS Debian packages</u>	13
<u>2.7.3. Installing from CVS</u>	13
<u>2.8. Activating EVMS volumes</u>	14
<u>2.9. The root filesystem on an EVMS volume</u>	15
<u>2.10. Setting up clustered EVMS</u>	15
<u>Chapter 3. Using the EVMS interfaces</u>	17
<u>3.1. EVMS GUI</u>	17
<u>3.1.1. Using context sensitive and action menus</u>	17
<u>3.1.2. Saving changes</u>	17
<u>3.1.3. Refreshing changes</u>	17
<u>3.1.4. Using the GUI "+"</u>	18
<u>3.1.5. Using the accelerator keys</u>	18
<u>3.2. EVMS Ncurses interface</u>	19
<u>3.2.1. Navigating through EVMS Ncurses</u>	20
<u>3.2.2. Saving changes</u>	20
<u>3.3. EVMS Command Line Interpreter</u>	20
<u>3.3.1. Using the EVMS CLI</u>	21
<u>3.3.2. Notes on commands and command files</u>	22
<u>Chapter 4. The EVMS log file and error data collection</u>	23
<u>4.1. About the EVMS log file</u>	23
<u>4.2. Log file logging levels</u>	23
<u>4.3. Specifying the logging levels</u>	24
<u>Chapter 5. Viewing compatibility volumes after migrating</u>	25
<u>5.1. Using the EVMS GUI</u>	25
<u>5.2. Using Ncurses</u>	25

Table of Contents

<u>Chapter 5. Viewing compatibility volumes after migrating</u>	
<u>5.3. Using the CLI</u>	26
<u>Chapter 6. Obtaining interface display details</u>	28
<u>6.1. Using the EVMS GUI</u>	28
<u>6.2. Using Ncurses</u>	28
<u>6.3. Using the CLI</u>	28
<u>Chapter 7. Assigning a segment manager</u>	30
<u>7.1. When to assign a segment manager</u>	30
<u>7.2. Types of segment managers</u>	30
<u>7.2.1. DOS Segment Manager</u>	30
<u>7.2.2. GUID Partitioning Table (GPT) Segment Manager</u>	30
<u>7.2.3. S/390 Segment Manager</u>	31
<u>7.2.4. Cluster segment manager</u>	31
<u>7.2.5. BSD segment manager</u>	31
<u>7.3. Assigning a segment manager to an existing disk</u>	32
<u>7.4. Assigning a segment manager to a new disk</u>	32
<u>7.5. Example: assign a segment manager</u>	32
<u>7.5.1. Using the EVMS GUI</u>	33
<u>7.5.2. Using Ncurses</u>	33
<u>7.5.3. Using the CLI</u>	33
<u>Chapter 8. Creating segments</u>	35
<u>8.1. When to create a segment</u>	35
<u>8.2. Example: create a segment</u>	35
<u>8.2.1. Using the EVMS GUI</u>	35
<u>8.2.2. Using Ncurses</u>	36
<u>8.2.3. Using the CLI</u>	36
<u>Chapter 9. Creating a container</u>	37
<u>9.1. When to create a container</u>	37
<u>9.2. Example: create a container</u>	37
<u>9.2.1. Using the EVMS GUI</u>	37
<u>9.2.2. Using Ncurses</u>	37
<u>9.2.3. Using the CLI</u>	38
<u>Chapter 10. Creating regions</u>	39
<u>10.1. When to create regions</u>	39
<u>10.2. Example: create a region</u>	39
<u>10.2.1. Using the EVMS GUI</u>	39
<u>10.2.2. Using Ncurses</u>	40
<u>10.2.3. Using the CLI</u>	40
<u>Chapter 11. Creating drive links</u>	41
<u>11.1. What is drive linking?</u>	41
<u>11.2. How drive linking is implemented</u>	41
<u>11.3. Creating a drive link</u>	41

Table of Contents

<u>Chapter 11. Creating drive links</u>	
<u>11.4. Example: create a drive link</u>	42
<u>11.4.1. Using the EVMS GUI</u>	42
<u>11.4.2. Using Ncurses</u>	42
<u>11.4.3. Using the CLI</u>	43
<u>11.5. Expanding a drive link</u>	43
<u>11.6. Shrinking a drive link</u>	44
<u>11.7. Deleting a drive link</u>	44
<u>Chapter 12. Creating snapshots</u>	45
<u>12.1. What is a snapshot?</u>	45
<u>12.2. Creating and activating snapshot objects</u>	45
<u>12.2.1. Creating a snapshot</u>	45
<u>12.2.2. Activating a snapshot</u>	45
<u>12.3. Example: create a snapshot</u>	46
<u>12.3.1. Using the EVMS GUI</u>	46
<u>12.3.2. Using Ncurses</u>	46
<u>12.3.3. Using the CLI</u>	47
<u>12.4. Reinitializing a snapshot</u>	47
<u>12.5. Deleting a snapshot</u>	47
<u>12.6. Rolling back a snapshot</u>	47
<u>Chapter 13. Creating volumes</u>	49
<u>13.1. When to create a volume</u>	49
<u>13.2. Example: create an EVMS native volume</u>	49
<u>13.2.1. Using the EVMS GUI</u>	49
<u>13.2.2. Using Ncurses</u>	50
<u>13.2.3. Using the CLI</u>	50
<u>13.3. Example: create a compatibility volume</u>	50
<u>13.3.1. Using the GUI</u>	50
<u>13.3.2. Using Ncurses</u>	51
<u>13.3.3. Using the CLI</u>	51
<u>Chapter 14. FSIMs and file system operations</u>	52
<u>14.1. The FSIMs supported by EVMS</u>	52
<u>14.1.1. JFS</u>	52
<u>14.1.2. XFS</u>	52
<u>14.1.3. ReiserFS</u>	52
<u>14.1.4. Ext2/3</u>	53
<u>14.1.5. SWAPFS</u>	53
<u>14.2. Example: add a file system to a volume</u>	53
<u>14.2.1. Using the EVMS GUI</u>	53
<u>14.2.2. Using Ncurses</u>	54
<u>14.2.3. Using the CLI</u>	54
<u>14.3. Example: check a file system</u>	54
<u>14.3.1. Using the EVMS GUI</u>	55
<u>14.3.2. Using Ncurses</u>	55
<u>14.3.3. Using the CLI</u>	55

Table of Contents

Chapter 15. Clustering operations	56
15.1. Rules and restrictions for creating cluster containers	56
15.2. Example: create a private cluster container	56
15.2.1. Using the EVMS GUI	57
15.2.2. Using Ncurses	57
15.2.3. Using the CLI	58
15.3. Example: create a shared cluster container	58
15.3.1. Using the EVMS GUI	58
15.3.2. Using Ncurses	58
15.3.3. Using the CLI	59
15.4. Example: convert a private container to a shared container	59
15.4.1. Using the EVMS GUI	60
15.4.2. Using Ncurses	60
15.4.3. Using the CLI	60
15.5. Example: convert a shared container to a private container	61
15.5.1. Using the EVMS GUI	61
15.5.2. Using Ncurses	61
15.5.3. Using the CLI	62
15.6. Example: deport a private or shared container	62
15.6.1. Using the EVMS GUI	62
15.6.2. Using Ncurses	63
15.6.3. Using the CLI	63
15.7. Deleting a cluster container	64
15.8. Failover and Failback of a private container on Linux-HA	64
15.9. Remote configuration management	64
15.9.1. Using the EVMS GUI	65
15.9.2. Using Ncurses	65
15.9.3. Using the CLI	65
15.10. Forcing a cluster container to be imported	65
Chapter 16. Converting volumes	67
16.1. When to convert volumes	67
16.2. Example: convert compatibility volumes to EVMS volumes	67
16.2.1. Using the EVMS GUI	67
16.2.2. Using Ncurses	68
16.2.3. Using the CLI	68
16.3. Example: convert EVMS volumes to compatibility volumes	68
16.3.1. Using the EVMS GUI	68
16.3.2. Using Ncurses	69
16.3.3. Using the CLI	69
Chapter 17. Expanding and shrinking volumes	70
17.1. Why expand and shrink volumes?	70
17.2. Example: shrink a volume	70
17.2.1. Using the EVMS GUI	71
17.2.2. Using Ncurses	71
17.2.3. Using the CLI	71
17.3. Example: expand a volume	72

Table of Contents

<u>Chapter 17. Expanding and shrinking volumes</u>	
<u>17.3.1. Using the EVMS GUI</u>	72
<u>17.3.2. Using Ncurses</u>	72
<u>17.3.3. Using the CLI</u>	73
<u>Chapter 18. Adding features to an existing volume</u>	74
<u>18.1. Why add features to a volume?</u>	74
<u>18.2. Example: add drive linking to an existing volume</u>	74
<u>18.2.1. Using the EVMS GUI</u>	74
<u>18.2.2. Using Ncurses</u>	75
<u>18.2.3. Using the CLI</u>	75
<u>Chapter 19. Plug-in operations tasks</u>	76
<u>19.1. What are plug-in tasks?</u>	76
<u>19.2. Example: complete a plug-in operations task</u>	76
<u>19.2.1. Using the EVMS GUI</u>	76
<u>19.2.2. Using Ncurses</u>	77
<u>19.2.3. Using the CLI</u>	77
<u>Chapter 20. Destroying EVMS objects</u>	78
<u>20.1. How to delete objects: delete and delete recursive</u>	78
<u>20.2. Example: perform a delete recursive operation</u>	78
<u>20.2.1. Using the EVMS GUI</u>	78
<u>20.2.2. Using Ncurses</u>	79
<u>20.2.3. Using the CLI</u>	79
<u>Appendix A. Building an init-ramdisk to use with EVMS</u>	80
<u>A.1. Build and install EVMS</u>	80
<u>A.2. Kernel support for initrd</u>	80
<u>A.3. Build the initrd image</u>	81
<u>A.3.1. Create a new, blank initrd</u>	81
<u>A.3.2. Mount the initrd</u>	81
<u>A.3.3. Set up the basic directory structure</u>	81
<u>A.3.4. Copy helpful utilities</u>	81
<u>A.3.5. Copy supporting libraries</u>	82
<u>A.3.6. Copy the EVMS tools</u>	82
<u>A.3.7. Set up disk devices</u>	83
<u>A.3.8. Copy kernel modules</u>	84
<u>A.3.9. Write the linuxrc script</u>	85
<u>A.3.10. Unmount the initrd image</u>	86
<u>A.3.11. Compress the initrd image</u>	86
<u>A.4. Set up the boot loader</u>	86
<u>A.4.1. LILO procedure</u>	86
<u>A.4.2. GRUB procedure</u>	87
<u>A.5. Update the file system configuration</u>	87
<u>A.6. Reboot the system</u>	87

Table of Contents

<u>Appendix B. The DOS link plug-in</u>	88
<u>B.1. How the DOS plug-in is implemented</u>	88
<u>B.2. Assigning the DOS plug-in</u>	89
<u>B.3. Creating DOS partitions</u>	89
<u>B.4. Expanding DOS partitions</u>	90
<u>B.5. Shrinking DOS partitions</u>	90
<u>B.6. Deleting partitions</u>	91
<u>Appendix C. The MD region manager</u>	92
<u>C.1. Creating an MD region</u>	92
<u>C.2. Adding and removing a spare object (RAID-1 and RAID-4/5)</u>	93
<u>C.3. Removing an active object (RAID-1 only)</u>	93
<u>C.4. Removing a faulty object (RAID-1 and RAID-4/5)</u>	93
<u>C.5. Marking an object faulty (RAID-4/5 only)</u>	93
<u>C.6. Replacing an object</u>	93
<u>C.7. Characteristics of Linux RAID levels</u>	93
<u>C.7.1. Linear mode</u>	93
<u>C.7.2. RAID-0</u>	94
<u>C.7.3. RAID-1</u>	94
<u>C.7.4. RAID-4</u>	95
<u>C.7.5. RAID-5</u>	95
<u>Appendix D. The LVM plug-in</u>	97
<u>D.1. How LVM is implemented</u>	97
<u>D.2. Container operations</u>	97
<u>D.2.1. Creating LVM containers</u>	97
<u>D.2.2. Adding objects to LVM containers</u>	97
<u>D.2.3. Removing objects from LVM containers</u>	98
<u>D.2.4. Deleting LVM containers</u>	98
<u>D.3. Region operations</u>	98
<u>D.3.1. Creating LVM regions</u>	98
<u>D.3.2. Expanding LVM regions</u>	99
<u>D.3.3. Shrinking LVM regions</u>	99
<u>D.3.4. Deleting LVM regions</u>	99
<u>Appendix E. The CSM plug-in</u>	100
<u>E.1. Assigning the CSM plug-in</u>	100
<u>E.2. Unassigning the CSM plug-in</u>	101
<u>E.3. Deleting a CSM container</u>	101
<u>Appendix F. JFS file system interface module</u>	102
<u>F.1. Creating JFS file systems</u>	102
<u>F.2. Checking JFS file systems</u>	102
<u>F.3. Removing JFS file systems</u>	102
<u>F.4. Expanding JFS file systems</u>	103
<u>F.5. Shrinking JFS file systems</u>	103

Table of Contents

<u>Appendix G. XFS file system interface module.....</u>	104
<u>G.1. Creating XFS file systems.....</u>	104
<u>G.2. Checking XFS file systems.....</u>	104
<u>G.3. Removing XFS file systems.....</u>	104
<u>G.4. Expanding XFS file systems.....</u>	104
<u>G.5. Shrinking XFS file systems.....</u>	104
<u>Appendix H. ReiserFS file system interface module.....</u>	105
<u>H.1. Creating ReiserFS file systems.....</u>	105
<u>H.2. Checking ReiserFS file systems.....</u>	105
<u>H.3. Removing ReiserFS file systems.....</u>	105
<u>H.4. Expanding ReiserFS file systems.....</u>	105
<u>H.5. Shrinking ReiserFS file systems.....</u>	105
<u>Appendix I. Ext-2/3 file system interface module.....</u>	106
<u>I.1. Creating Ext-2/3 file systems.....</u>	106
<u>I.2. Checking Ext-2/3 file systems.....</u>	106
<u>I.3. Removing Ext-2/3 file systems.....</u>	106
<u>I.4. Expanding and shrinking Ext-2/3 file systems.....</u>	106

Preface

This guide tells how to install, configure, and manage Enterprise Volume Management System (EVMS). EVMS is a storage management program that provides a single framework for managing and administering your system's storage.

This guide is intended for Linux system administrators and users who are responsible for setting up and maintaining EVMS.

For additional information about EVMS or to ask questions specific to your distribution, refer to the EVMS mailing lists. You can view the list archives or subscribe to the lists from the [EVMS Project web site](#).

The following table shows how this guide is organized:

Table 1. Organization of the EVMS User Guide

Chapter or appendix title	Contents
1. What is EVMS?	Discusses general EVMS concepts and terms.
2. Downloading and installing EVMS	Tells how to access and install EVMS and configure it for use on your system.
3. Using the EVMS interfaces	Describes the three EVMS user interfaces and how to use them.
4. The EVMS log file and error data collection	Discusses the EVMS information and error log file and explains how to change the logging level.
5. Viewing compatibility volumes after migrating	Tells how to view existing files that have been migrated to EVMS.
6. Obtaining interface display details	Tells how to view detailed information about EVMS objects.
7. Assigning a segment manager	Discusses segments and explains how to assign a segment manager.
8. Creating segments	Explains when and how to create segments.
9. Creating containers	Discusses containers and explains when and how to create them.
10. Creating regions	Discusses regions and explains when and how to create them.
11. Creating drive links	Discusses the drive linking feature and tells how to create a drive link.
12. Creating snapshots	Discusses snapshotting and tells how to create a snapshot.
13. Creating volumes	Explains when and how to create volumes.
14. FSIMS and file system operations	Discusses the standard FSIMs shipped with EVMS and provides examples of adding file systems and coordinating file checks with the FSIMs.

15. Clustering operations	Describes EVMS clustering and how to create private and shared containers.
16. Converting volumes	Explains how to convert EVMS native volumes to compatibility volumes and compatibility volumes to EVMS native volumes.
17. Expanding and shrinking volumes	Tells how to expand and shrink EVMS volumes with the various EVMS user interfaces.
18. Adding features to an existing volume	Tells how to add additional features, such as drive linking and bad block relocation, to an existing volume.
19. Plug-in operations tasks	Discusses the plug-in tasks that are available within the context of a particular plug-in.
20. Destroying EVMS objects	Tells how to safely destroy EVMS objects.
A. Building an init-ramdisk to use with EVMS	Explains the steps necessary to build a ram-based device that acts a temporary root file system at boot time.
B. The DOS link plug-in	Provides details about the DOS link plug-in, which is a segment manager plug-in.
C. The MD region manager	Explains the Multiple Disks (MD) support in Linux that is a software implementation of RAID.
D. The LVM plug-in	Tells how the LVM plug-in is implemented and how to perform container operations.
E. The CSM plug-in	Explains how the Cluster Segment Manager (CSM) plug-in is implemented and how to perform CSM operations.
F. JFS file system interface module	Provides information about the JFS FSIM.
G. XFS file system interface module	Provides information about the XFS FSIM.
H. ReiserFS file system interface module	Provides information about the ReiserFS FSIM.
I. Ext-2/3 file system interface module	Provides information about the Ext-2/3 FSIM.

Chapter 1. What is EVMS?

EVMS brings a new model of volume management to Linux®. EVMS integrates all aspects of volume management, such as disk partitioning, Linux logical volume manager (LVM) and multi-disk (MD) management, OS2 and AIX volume managers, and file system operations into a single cohesive package. With EVMS, various volume management technologies are accessible through one interface, and new technologies can be added as plug-ins as they are developed.

1.1. Why choose EVMS?

EVMS lets you manage storage space in a way that is more intuitive and flexible than many other Linux volume management systems. Practical tasks, such as migrating disks or adding new disks to your Linux system, become more manageable with EVMS because EVMS can recognize and read from different volume types and file systems. EVMS provides additional safety controls by not allowing commands that are unsafe. These controls help maintain the integrity of the data stored on the system.

You can use EVMS to create and manage data storage. With EVMS, you can use multiple volume management technologies under one framework while ensuring your system still interacts correctly with stored data. With EVMS, you can use bad block relocation, shrink and expand volumes, create snapshots of your volumes, and set up RAID (redundant array of independent devices) features for your system. You can also use many types of file systems and manipulate these storage pieces in ways that best meet the needs of your particular work environment.

EVMS also provides the capability to manage data on storage that is physically shared by nodes in a cluster. This shared storage allows data to be highly available from different nodes in the cluster.

1.2. The EVMS user interfaces

There are currently three user interfaces available for EVMS: graphical (GUI), text mode (Ncurses), and the Command Line Interpreter (CLI). Additionally, you can use the EVMS Application Programming Interface to implement your own customized user interface.

[Table 1-1](#) tells more about each of the EVMS user interfaces.

Table 1-1. EVMS user interfaces

User interface	Typical user	Types of use	Function
GUI	All	All uses except automation	Allows you to choose from available options only, instead of having to sort through all the options, including ones that are not available at that point in the process.
Ncurses	Users who don't have GTK libraries or X Window Systems on their machines	All uses except automation	Allows you to choose from available options only, instead of having to sort through all the options, including ones that are not available at that point in the process.

Command Line	Expert	All uses	Allows easy automation of tasks
--------------	--------	----------	---------------------------------

1.3. EVMS terminology

To avoid confusion with other terms that describe volume management in general, EVMS uses a specific set of terms. These terms are listed, from most fundamental to most comprehensive, as follows:

Logical disk

Representation of anything EVMS can access as a physical disk. In EVMS, physical disks are logical disks.

Sector

The lowest level of addressability on a block device. This definition is in keeping with the standard meaning found in other management systems.

Disk segment

An ordered set of physically contiguous sectors residing on the same storage object. The general analogy for a segment is to a traditional disk partition, such as DOS or OS/2 ®

Storage region

An ordered set of logically contiguous sectors that are not necessarily physically contiguous.

Storage object

Any persistent memory structure in EVMS that can be used to build objects or create a volume. Storage object is a generic term for disks, segments, regions, and feature objects.

Storage container

A collection of storage objects. A storage container consumes one set of storage objects and produces new storage objects. One common subset of storage containers is volume groups, such as AIX® or LVM.

Storage containers can be either of type private or cluster.

Cluster storage container

Specialized storage containers that consume only disk objects that are physically accessible from all nodes of a cluster.

Private storage container

A collection of disks that are physically accessible from all nodes of a cluster, managed as a single pool of storage, and owned and accessed by a single node of the cluster at any given time.

Shared storage container

A collection of disks that are physically accessible from all nodes of a cluster, managed as a single pool of storage, and owned and accessed by all nodes of the cluster simultaneously.

Deported storage container

A shared cluster container that is not owned by any node of the cluster.

Feature object

A storage object that contains an EVMS native feature, such as bad block relocation.

An *EVMS Native Feature* is a function of volume management designed and implemented by EVMS. These features are not intended to be backwards compatible with other volume management technologies.

Logical volume

A volume that consumes a storage object and exports something mountable. There are two varieties of logical volumes: *EVMS Volumes* and *Compatibility volumes*.

EVMS Volumes contain EVMS native metadata and can support all EVMS features.
`/dev/evms/my_volume` would be an example of an EVMS Volume.

Compatibility volumes do not contain any EVMS native metadata. Compatibility volumes are backward compatible to their particular scheme, but they cannot support EVMS features.
`/dev/evms/md/md0` would be an example of a compatibility volume.

1.4. What makes EVMS so flexible?

There are numerous drivers in the Linux kernel, such as Device Mapper and MD (software RAID), that implement volume management schemes. EVMS is built on top of these drivers to provide one framework for combining and accessing the capabilities.

The EVMS Engine handles the creation, configuration, and management of volumes, segments, and disks. The EVMS Engine is a programmatic interface to the EVMS system. User interfaces and programs that use EVMS must go through the Engine.

EVMS provides the capacity for plug-in modules to the Engine that allow EVMS to perform specialized tasks without altering the core code. These plug-in modules allow EVMS to be more extensible and customizable than other volume management systems.

1.5. Plug-in layer definitions

EVMS defines a layered architecture where plug-ins in each layer create abstractions of the layer or layers below. EVMS also allows most plug-ins to create abstractions of objects within the same layer. The following list defines these layers from the bottom up.

Device managers

The first (bottom) layer consists of device managers. These plug-ins communicate with hardware device drivers to create the first EVMS objects. Currently, all devices are handled by a single plug-in. Future releases of EVMS might need additional device managers for network device management (for example, to manage disks on a storage area network (SAN)).

Segment managers

The second layer consists of segment managers. These plug-ins handle the segmenting, or partitioning, of disk drives. The Engine components can replace partitioning programs, such as **fdisk** and Disk Druid, and EVMS uses Device Mapper to replace the in-kernel disk partitioning code. Segment managers can also be "stacked," meaning that one segment manager can take as input the output from another segment manager.

EVMS provides the following segment managers: DOS, GPT, System/390® (S/390), Cluster, and BSD. Other segment manager plug-ins can be added to support other partitioning schemes.

Region managers

The third layer consists of region managers. This layer provides a place for plug-ins that ensure compatibility with existing volume management schemes in Linux and other operating systems. Region managers are intended to model systems that provide a logical abstraction above disks or partitions.

Like segment managers, region managers can also be stacked. Therefore, the input object(s) to a region manager can be disks, segments, or other regions.

There are currently four region manager plug-ins in EVMS: Linux LVM, AIX, OS/2, and Multi-Disk (MD).

Linux LVM

The Linux LVM plug-in provides compatibility with the Linux LVM and allows the creation of volume groups (known in EVMS as containers) and logical volumes (known in EVMS as regions).

AIX LVM

The AIX LVM provides compatibility with AIX and is similar in functionality to the Linux LVM by also using volume groups and logical volumes.

OS/2 LVM

The OS/2 plug-in provides compatibility with volumes created under OS/2. Unlike the Linux and AIX LVMs, the OS/2 LVM is based on linear linking of disk partitions, as well as bad-block relocation. The OS/2 LVM does not allow for modifications.

MD

The Multi-Disk (MD) plug-in for RAID provides RAID levels linear, 0, 1, 4, and 5 in software. MD is one plug-in that displays as four region managers that you can choose from.

EVMS features

The next layer consists of EVMS features. This layer is where new EVMS-native functionality is implemented. EVMS features can be built on any object in the system, including disks, segments, regions, or other feature objects. All EVMS features share a common type of metadata, which makes discovery of feature objects much more efficient, and recovery of broken features objects much more reliable. There are three features currently available in EVMS: drive linking, Bad Block Relocation, and snapshotting.

Drive Linking

Drive linking allows any number of objects to be linearly concatenated together into a single object. A drive linked volume can be expanded by adding another storage object to the end or shrunk by removing the last object.

Bad Block Relocation

Bad Block Relocation (BBR) monitors its I/O path and detects write failures (which can be caused by a damaged disk). In the event of such a failure, the data from that request is stored in a new location. BBR keeps track of this remapping. Additional I/Os to that location are redirected to the new location.

Snapshotting

The Snapshotting feature provides a mechanism for creating a "frozen" copy of a volume at a single instant in time, without having to take that volume off-line. This is useful for performing backups on a live system. Snapshots work with any volume (EVMS or compatibility), and can use any other available object as a backing store. After a snapshot is created and made into an EVMS volume, writes to the "original" volume cause the original contents of that location to be copied to the snapshot's storage object. Reads to the snapshot volume look like they come from the original at the time the snapshot was created.

File System Interface Modules

File System Interface Modules (FSIMs) provide coordination with the file systems during certain volume management operations. For instance, when expanding or shrinking a volume, the file system must also be expanded or shrunk to the appropriate size. Ordering in this example is also important; a file system cannot be expanded before the volume, and a volume cannot be shrunk before the file system. The FSIMs allow EVMS to ensure this coordination and ordering.

FSIMs also perform file system operations from one of the EVMS user interfaces. For instance, a user can make new file systems and check existing file systems by interacting with the FSIM.

Cluster Manager Interface Modules

Cluster Manager Interface Modules, also known as the EVMS Clustered Engine (ECE), interface with the local cluster manager installed on the system. The ECE provides a standardized ECE API to the Engine while hiding cluster manager details from the Engine.

Chapter 2. Downloading and installing EVMS

This chapter tells how to obtain and install EVMS, and offers guidelines to help you make decisions about your EVMS installation. If you are setting up EVMS to work in a cluster, execute the steps described in this chapter on each of the nodes in the cluster.

2.1. Preparing your system

Before you install EVMS, decide whether to implement certain kernel capabilities and ensure that you have a Linux kernel source tree. EVMS 2.0 is built on top of the already existing components MD and Device Mapper (DM). If you are using Linux kernel 2.5 or later, the MD and DM components are already available, but you need to apply some patches for bug fixes because the 2.5 kernel is still stabilizing. If you are using Linux kernel 2.4, you need to obtain all the DM patches from the Sistina website and then apply the bug fixes. Steps for performing these tasks are included later in this chapter.

2.1.1. Choosing capabilities

In order to run EVMS, you need to configure the Device Mapper in your kernel either as a module or as a component built directly into the kernel. If you want to use the RAID functions of EVMS, turn on MD and the appropriate RAID personalities in the kernel.

2.1.2. Obtaining the Device Mapper and a kernel source tree

EVMS 2.0 is built on top of the already existing components MD and Device Mapper. If you are using Linux kernel 2.5 or later, the MD and DM components are already available; however, you need to apply some patches for bug fixes because the 2.5 kernel is still stabilizing. If you are using Linux kernel 2.4, you need to patch the DM in from the [Sistina website](#) and then apply the bug fixes. The DM patches for the kernel are included in the EVMS package on the [EVMS project website](#). The current EVMS package contains a patch for both the 2.4 and 2.5 kernel series.



NOTE

EVMS does not work with Linux kernels 2.3, 2.2, or earlier.

If you do not have a Linux kernel source tree, you can obtain one from [The Linux Kernel Archives](#). The current recommended kernel is 2.4.20, but earlier 2.4 series kernels might work.

2.2. Downloading EVMS

To install EVMS, download the latest version of EVMS (`evms-2.0.0.tar.gz`) from the [EVMS project homepage](#). This file contains source code patches for the Linux kernel and source code for all necessary user-space administration tools.

After downloading the file, use the **tar** command to extract the files in the appropriate place. In the following example, the files are extracted in `/usr/src`. You can substitute another directory.

```
cd /usr/src
tar xvzf evms-2.0.0.tar.gz
```

2.3. Patching the kernel

In addition to the base MD and Device Mapper support, EVMS requires a few patches for the EVMS engine to work correctly with these drivers. These patches are provided in the EVMS package, in the `kernel/2.4.20/` subdirectory. See the INDEX files in the `kernel/2.4.20` directory for descriptions of the patches.

NOTE

If you are currently using the Sistina LVM2 tools, the patches provided here are not compatible with the LVM2 tools. EVMS has made some changes to the kernel ioctl packets so that they work correctly on ppc-64 and sparc-64 architectures. These changes have not yet been accepted by Sistina. Thus, for the time being, you need to use separate kernels for running EVMS and LVM2.

Apply each of the patches for 2.4.20 to your kernel as follows:

```
cd /usr/src/linux-2.4.20
patch -p1 < /src/src/evms-2.0.0/kernel/2.4.20/1-dm-base.patch
patch -p1 < /src/src/evms-2.0.0/kernel/2.4.20/2-syncio.patch
patch -p1 < /src/src/evms-2.0.0/kernel/2.4.20/3-dm-bbr.patch
patch -p1 < /src/src/evms-2.0.0/kernel/2.4.20/4-dm-sparse.patch
patch -p1 < /src/src/evms-2.0.0/kernel/2.4.20/5-md.c.patc
patch -p1 < /src/src/evms-2.0.0/kernel/2.4.20/6-vsprintf.c.patch
patch -p1 < /src/src/evms-2.0.0/kernel/2.4.20/7-vfs-lock.patch
```

2.4. Configuring the kernel

After patching the kernel, configure it with EVMS support. To configure the kernel, complete the following steps:

1. Type the following command:

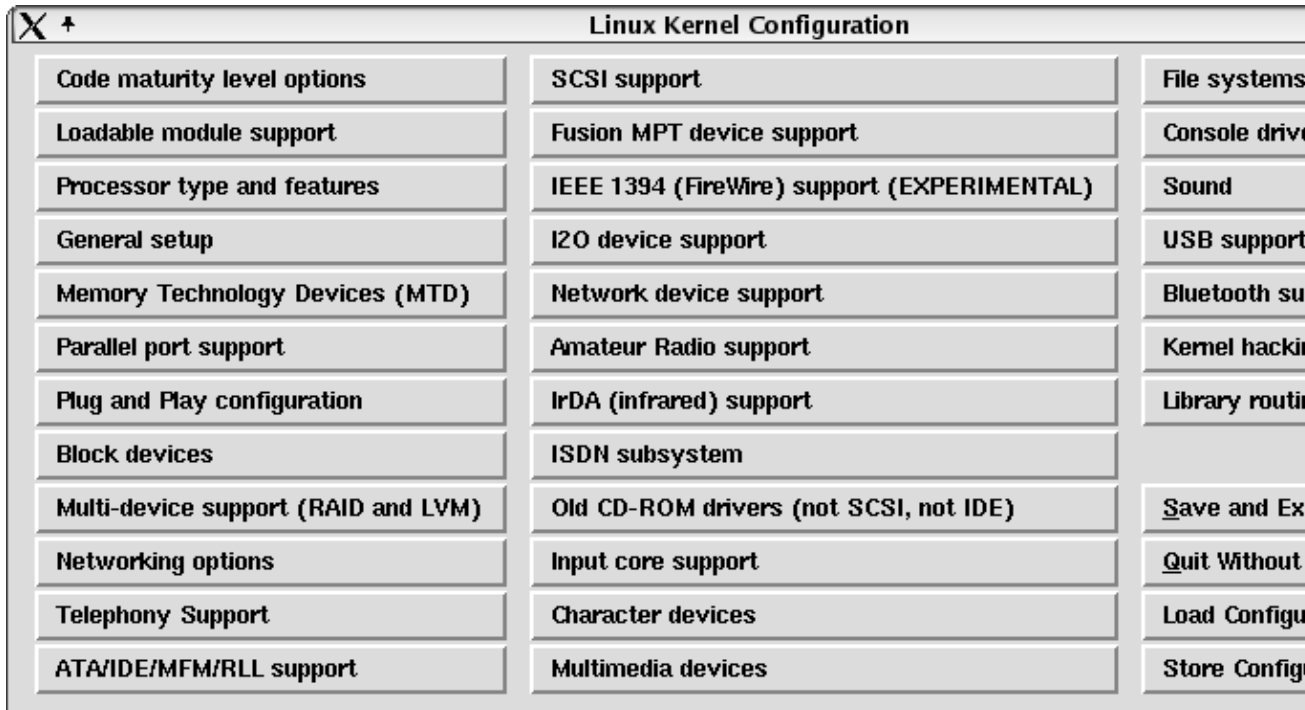
```
make xconfig
```

NOTE

You can also use **config** or **menuconfig** if you don't have an Xwindow system.

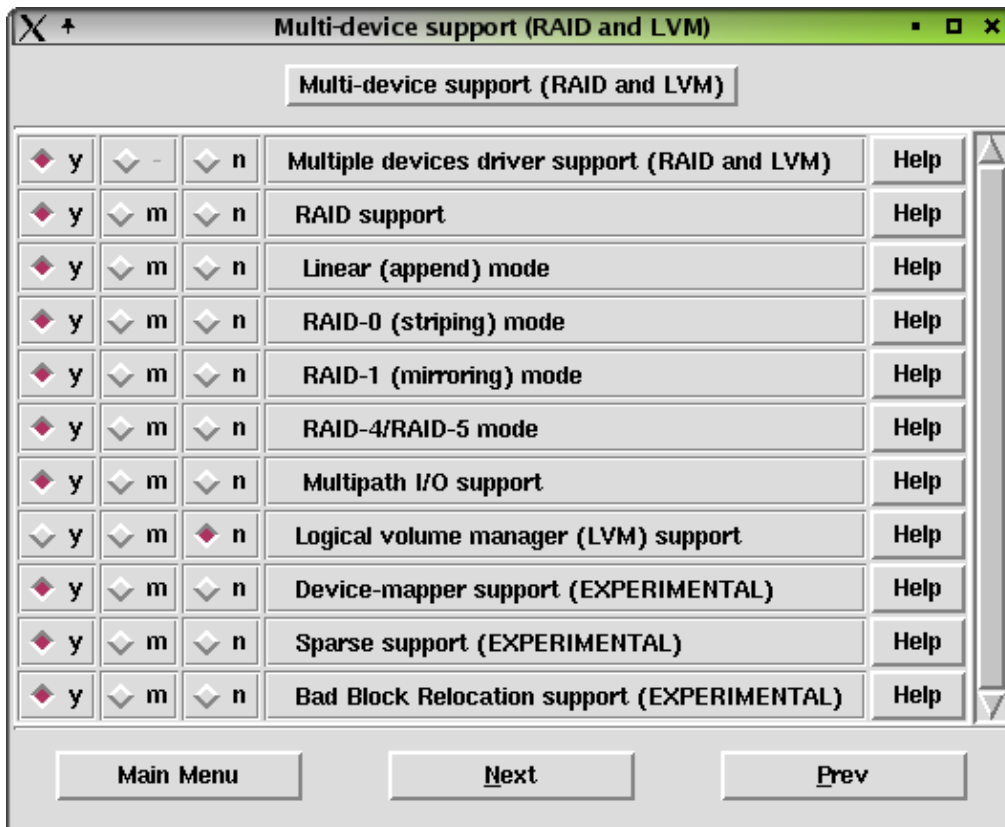
If you are using an Xwindow system, the following window opens.

Figure 2–1. xconfig Window



- To enable the kernel components that EVMS needs, select Multi-device support (RAID and LVM). The following window opens.

Figure 2–2. Multi-device Support (RAID and LVM) Window



NOTE

If you are configuring the components as modules, you can click the Help button beside each module to see the name of each kernel module.

3. Select `y` for Multi-device support (RAID and LVM). Select `y` (to build in the kernel) or `m` (to build modules) for Device Mapper support. These options are the minimum required for EVMS. The remaining options add additional capabilities to EVMS.

**NOTE**

The options you select in this step are the ones you selected in [Section 2.1.1](#).

Continue configuring your kernel as required for your system and hardware. For general instructions on configuring and compiling a Linux kernel, please see [The Kernel HOWTO](#).

When you have finished configuring your kernel, choose Save and Exit to quit the kernel configuration.

2.5. Building and installing the new kernel

After you have configured the kernel, build and install a new kernel image. There are slight variations for different architectures, but the general steps are the same. For example, on Intel® machines you might run **lilo** to install the new kernel image. On S/390 machines, you might run **zipl** to install the new kernel image. The following instructions are based on an installation using an Intel machine.

1. Build the kernel:

```
make dep clean bzImage modules modules_install
```

2. Copy the new kernel to the `/boot` directory.

```
cp /usr/src/linux/arch/i386/boot/bzImage /boot/vmlinuz-evms
```

**NOTE**

The previous command assumes the file to be named `vmlinuz-evms`, you can substitute another name.

3. Open the file for the boot loader. The LILO boot loader information is in `/etc/lilo.conf`. The GRUB boot loader information is in `/boot/menu.list`.
4. Update the boot loader information as follows:

```
image=/boot/vmlinuz-evms
    label=EVMS
```

**NOTE**

Currently, LILO does not support placing the `/boot` filesystem on a Device Mapper device. Thus, `/boot` must be mounted through the traditional partition device.

5. Run your boot loader to install the new kernel image.

**NOTE**

If you installed your kernel as modules, skip to [Section 2.7](#).

6. Reboot your machine and choose the new kernel image to run with EVMS support.
-

2.6. Installing the cluster manager

Skip this section if you do not require high-availability features.

EVMS currently supports the Linux-HA cluster manager, commonly known as Heartbeat. You can download the latest version of the Linux-HA software from <http://www.linux-ha.org/download/>. EVMS works with Heartbeat V1.0.1 and later. For instructions on setting up a two-node cluster, see <http://www.linux-ha.org/download/GettingStarted.html>. For additional information and a list of frequently asked questions, see <http://www.linux-ha.org/download/faqtips.html>.

Linux-HA provides the following sub-packages:

- heartbeat-xxx.arch.rpm
- heartbeat-stonith-xxx.arch.rpm
- heartbeat-pils-xxx.arch.rpm
- heartbeat-lldirectord-xxx.arch.rpm

"xxx" is the version of heartbeat; "arch" is the architecture of the target nodes.

You do not need to install `heartbeat-lldirectord-xxx.arch.rpm` because EVMS does not depend on its service.



NOTE

The download page also contains tarballs and source rpms.

To install the package, execute the following command on each cluster node:

```
rpm -i heartbeat-pils-xxx.arch.rpm heartbeat-stonith-xxx.rpm \
heartbeat-xxx.arch.rpm
```



NOTE

It is highly recommended that you configure the STONITH package on a two-node cluster because data integrity on shared disks is at risk if STONITH is not configured. Although EVMS operates even when STONITH is not configured (EVMS is not restricted to a two-node cluster), EVMS's failover capability is limited by the support provided by the native cluster manager.

Refer to the STONITH device manual for instructions on how to configure the STONITH device.

2.7. Installing the Engine

The EVMS Engine consists of the user-space plug-ins and interfaces as well as a stand-alone library, `dlist`, that the Engine uses for linked-list management.

2.7.1. Installing from source

1. To build and install the Engine from source, first type the following command to change to the `/usr/src/evms-2.0.0` directory:

```
cd /usr/src/evms-2.0.0
```

2. Select the appropriate options for your configuration.

**NOTE**

A list of options can be found by typing:

```
./configure -h
```

3. Configure your engine:

```
./configure [--options]
```

4. Type the following commands:

```
make
make install
ldconfig
```

Unless you specified other directories when you configured the engine, the following list describes where files will be installed on your system:

- ◆ The core Engine library is installed in `/lib`.
 - ◆ The core dlist library is installed in `/lib`.
 - ◆ All plug-in libraries is installed in `/lib/evms`.
 - ◆ All user interface binaries are installed in `/sbin`.
 - ◆ All header files are installed in `/usr/include/evms`.
 - ◆ All manual pages are installed in `/usr/man`.
 - ◆ Add the Engine library path to your `LD_LIBRARY_PATH` environment variable, or to your `/etc/ld.so.conf` file. Do not add the plug-in library path because the Engine dynamically loads these libraries directly.
5. Examine the EVMS configuration file (`evms.conf`). This file contains settings to control how EVMS operates. For example, the logging level, the location of the engine log, and the list of disk devices to examine can all be controlled through settings in the configuration file. The sample file is well commented and advises you of appropriate values for each setting.

The `evms.conf` file is normally installed in `/etc/evms.conf`. However, if you already have a configuration file, it is installed as `/etc/evms.conf.sample`. Examine the new sample to see if your existing file should be updated.

You can now begin using EVMS by typing **evmsgui** to start the GUI, **evmsn** to start the Ncurses UI, or **evms** to start the command line interpreter.

2.7.2. Installing the EVMS Debian packages

Debian packages of EVMS are maintained by Matt Zimmerman at the [Debian](#) website. You can download these packages for the [Woody](#) (testing) release, or for the [Sid](#) (unstable) release.

For the Debian package, patch and recompile your kernel in order to use EVMS. Debian has a packaged version of the EVMS kernel patches that includes instructions for patching and building your kernel.

2.7.3. Installing from CVS

If you would like to use the most recently developed version of EVMS, you can get the code directly from the CVS tree.

1. To download the code, type the following commands:

```
mkdir /usr/src/evms_cvs
cd /usr/src/evms_cvs
cvs -d:pserver:anonymous@cvs.evms.sf.net:/cvsroot/evms login
    (press Enter when prompted for a password)
cvs -z3 -d:pserver:anonymous@cvs.evms.sf.net:/cvsroot/evms co engine2
```

2. Alternatively, if you already have a copy of the CVS code checked out, you can update to the latest code with the following commands:

```
cd /usr/src/evms_cvs/engine2
cvs update -d -P
```

3. The EVMS Engine can be built directly into the CVS tree. To start, change directory to /usr/src/evms_cvs/engine2:

```
cd /usr/src/evms_cvs/engine2
autoconf
```

**NOTE**

The **autoconf** command is necessary only for code taken directly from CVS.

4. Select the appropriate options for your configuration.

**NOTE**

You can find the list of options by typing:

```
./configure -h
```

5. Configure your engine:

```
./configure [--options]
```

6. Type the following commands:

```
make
make install
ldconfig
```

Unless you specified other directories in Step 5, the following list describes where files are installed on your system:

- ◆ The core Engine library is installed in /lib.
- ◆ The core dlist library is installed in /lib.
- ◆ All plug-in libraries is installed in /lib/evms.
- ◆ All user interface binaries are installed in /sbin.
- ◆ All header files are installed in /usr/include/evms.
- ◆ All manual pages are installed in /usr/man.

7. Add the Engine library path to your LD_LIBRARY_PATH environment variable, or to your /etc/ld.so.conf file. Do not add the plug-in library path because the Engine dynamically loads these libraries directly.

2.8. Activating EVMS volumes

In the previous EVMS design (releases 1.2.1 and earlier), volume discovery was performed in the kernel, and all volumes were immediately activated at boot time. With the current EVMS design, volume discovery is performed in user-space, and volumes are activated by communicating with the kernel. Thus, in order to activate your volumes, you must open one of the EVMS user interfaces and perform a save, which will activate all inactive volumes.

For example, start the GUI by running **evmsgui**. You should see all empty checkboxes in the "Active" column. Press Save, which activates all of the volumes and fills in each of the checkboxes.

In addition to manually starting one of the EVMS user interfaces, there is a new utility called **evms_activate**. The **evms_activate** utility simply opens the EVMS Engine and issues a **commit** command. You might want to add a call to **evms_activate** to your boot scripts in order to automatically activate your volumes at boot time. If you have volumes listed in your `/etc/fstab` file, you need to call **evms_activate** before the `fstab` is processed.



NOTE

EVMS requires `/proc` to be mounted in order to find the Device Mapper driver. If you run **evms_activate** before processing the `fstab` file, you might need to manually mount and unmount `/proc` around the call to **evms_activate**.

After the volumes are activated, you can mount them in the normal way with the `dev`-nodes in the `/dev/evms` directory.

2.9. The root filesystem on an EVMS volume

After volume discovery and activation are done in user space, there is an issue with having your system's root file system on an EVMS volume. In order for the root file system's volume to be activated, the EVMS tools must run. However, in order to get to the EVMS tools, the root file system must be mounted. The solution to this dilemma is to use an initial ramdisk (**initrd**). An initial ramdisk is a ram-based device that acts as a temporary root file system at boot time. The initial ramdisk lets EVMS run programs and load modules that are necessary to activate the true root file system.

For instructions on building an `init-ramdisk` for use with EVMS, see [Appendix A](#).

2.10. Setting up clustered EVMS

The following steps show how to set up EVMS clustering. Perform these steps on each cluster node:

1. Because EVMS depends on the CCM services offered by Heartbeat, the Linux-HA CCM service must be configured on the system. Execute the following line to configure CCM:

```
echo 'respawn haclient /usr/lib/heartbeat/ccm' >> /etc/ha.d/ha.cf
```

2. Because Linux-HA expects its consumers to set up a private communication channel, create the following fifo with administrative privileges:

```
mkfifo /var/lib/heartbeat/api/evms.req
mkfifo /var/lib/heartbeat/api/evms.rsp
chgrp haclient /var/lib/heartbeat/api/evms.req
chgrp haclient /var/lib/heartbeat/api/evms.rsp
chmod 200 /var/lib/heartbeat/api/evms.req
chmod 600 /var/lib/heartbeat/api/evms.rsp
```

3. To ensure that clustered EVMS is activated whenever Linux-HA starts, execute the following line:

```
echo 'respawn root /sbin/evmsd' >> /etc/ha.d/ha.cf
```

This step assumes that the **evmsd** daemon is installed in `/sbin/evmsd`. If the **evmsd** daemon is not installed in `/sbin/evmsd`, change `/sbin/evmsd` to the correct location when executing the command.

See [Chapter 15](#) for information about clustering operations.

Chapter 3. Using the EVMS interfaces

This chapter explains how to use the EVMS GUI, Ncurses, and CLI interfaces. This chapter also includes information about basic navigation and commands available through the CLI.

3.1. EVMS GUI

The EVMS GUI is a flexible and easy-to-use interface for administering volumes and storage objects. Many users find the EVMS GUI easy to use because it displays which storage objects, actions, and plug-ins are acceptable for a particular task.

3.1.1. Using context sensitive and action menus

The EVMS GUI lets you accomplish most tasks in one of two ways: context sensitive menus or the Actions menu.

Context sensitive menus are available from any of the main "views." Each view corresponds to a page in a notebook widget located on the EVMS GUI main window. These views are made up of different trees or lists that visually represent the organization of different object types, including volumes, feature objects, regions, containers, segments, or disks.

You can view the context sensitive menu for an object by right-clicking on that object. The actions that are available for that object display on the screen. The GUI will only present actions that are acceptable for the selected object at that point in the process. These actions are not always a complete set.

To use the Actions menu, choose Action-><the action you want to accomplish>-><options>. The Actions menu provides a more guided path for completing a task than do context sensitive menus. The Actions option is similar to the wizard or druid approach used by many GUI applications.

All of the operations you need to perform as an administrator are available through the Actions menu.

3.1.2. Saving changes

All of the changes that you make while in the EVMS GUI are only in memory until you save the changes. In order to make your changes permanent, you must save all changes before exiting. If you forget to save the changes and decide to exit or close the EVMS GUI, you are reminded to save any pending changes.

To explicitly save all the changes you made, select Action->Save, and click the Save button.

3.1.3. Refreshing changes

The Refresh button updates the view and allows you to see changes, like mount points, that might have changed outside of the GUI.

3.1.4. Using the GUI "+"

Along the left hand side of the panel views in the GUI is a "+" that resides beside each item. When you click the "+," the objects that are included in the item are displayed. If any of the objects that display also have a "+" beside them, you can expand them further by clicking on the "+" next to each object name.

3.1.5. Using the accelerator keys

You can avoid using a mouse for navigating the EVMS GUI by using a series of key strokes, or "accelerator keys," instead. The following sections tell how to use accelerator keys in the EVMS Main Window, the Selection Window, and the Configuration Options Window.

3.1.5.1. Main Window accelerator keys

In the Main Window view, use the following keys to navigate:

Table 3–1. Accelerator keys in the Main Window

Left and right arrow keys	Navigate between the notebook tabs of the different views.
Down arrow and Spacebar	Bring keyboard focus into the view.

While in a view, use the following keys to navigate:

Table 3–2. Accelerator keys in the views

up and down arrows	Allow movement around the window.
"+"	Opens an object tree.
"_"	Collapses an object tree.
ENTER	Brings up the context menu (on a row).
Arrows	Navigate a context menu.
ENTER	Activates an item.
ESC	Dismisses the context menu.
Tab	Gets you out of the view and moves you back up to the notebook tab.

To access the action bar menu, press **Alt** and then the underlined accelerator key for the menu choice (for example, "A" for the Actions dropdown menu).

In a dropdown menu, you can use the up and down arrows to navigate. You could also just type the accelerator key for the menu item, which is the character with the underscore. For example, to initiate a command to delete a container, type **Alt** + "A" + "D" + "C."

Ctrl-S is a shortcut to initiate saving changes. **Ctrl-Q** is a shortcut to initiate quitting the EVMS GUI.

3.1.5.2. Accelerator keys in the selection window

A selection window typically contains a selection list, plus four to five buttons below it. Use the following keys to navigate in the selection window:

Table 3–3. Accelerator keys in the selection window

Tab	Navigates (changes keyboard focus) between the list and the buttons.
Up and down arrows	Navigates within the selection list.
Spacebar	Selects and deselects items in the selection list.
Enter on the button or type the accelerator character (if one exists)	Activates a button

3.1.5.3. Configuration options window accelerator keys

Use the following keys to navigate in the configuration options window:

Table 3–4. Accelerator keys in the configuration options window

Tab	Cycles focus between fields and buttons
Left and right arrows	Navigate the folder tabs if the window has a widget notebook.
Spacebar or the down arrow	Switches focus to a different notebook page.
Enter or type the accelerator character (if one exists)	Activates a button

For widgets, use the following keys to navigate:

Table 3–5. Widget navigation keys in the configuration options window

Tab	Cycles forward through a set of widgets
Shift–Tab	Cycles backward through a set of widgets.

The widget navigation, selection, and activation is the same in all dialog windows.

3.2. EVMS Ncurses interface

The EVMS Ncurses (**evmsn**) user interface is a menu-driven interface with characteristics similar to those of the EVMS GUI. Like the EVMS GUI, **evmsn** can accommodate new plug-ins and features without requiring any code changes.

The EVMS Ncurses user interface allows you to manage volumes on systems that do not have the X and GTK+ libraries that are required by the EVMS GUI.

3.2.1. Navigating through EVMS Ncurses

The EVMS Ncurses user interface initially displays a list of logical volumes similar to the logical volumes view in the EVMS GUI. Ncurses also provides a menu bar similar to the menu bar in the EVMS GUI.

A general guide to navigating through the layout of the Ncurses window is listed below:

- **Tab** cycles you through the available views.
- Status messages and tips are displayed on the last line of the screen.
- Typing the accelerator character (the letter highlighted in red) for any menu item activates that item. For example, typing **A** in any view brings down the Actions menu.
- Typing **A + Q** in a view quits the application.
- Typing **A + S** in a view saves changes made during an **evmsn** session.
- Use the **up** and **down** arrows to highlight an object in a view. Pressing **Enter** while an object in a view is highlighted presents a context popup menu.
- Dismiss a context popup menu by pressing **Esc** or by selecting a menu item with the **up** and **down** arrows and pressing **Enter** to activate the menu item.

Dialog windows are similar in design to the EVMS GUI dialogs, which allow a user to navigate forwards and backwards through a series of dialogs using Next and Previous. A general guide to dialog windows is listed below:

- **Tab** cycles you through the available buttons. Note that some buttons might not be available until a valid selection is made.
- The **left** and **right** arrows can also be used to move to an available button.
- Navigate a selection list with the **up** and **down** arrows.
- Toggle the selection of an item in a list with **spacebar**.
- Activate a button that has the current focus with **Enter**. If the button has an accelerator character (highlighted in red), you can also activate the button by typing the accelerator character regardless of whether the button has the current focus.

The EVMS Ncurses user interface, like the EVMS GUI, provides context menus for actions that are available only to the selected object in a view. Ncurses also provides context menus for items that are available from the Actions menu. These context menus present a list of commands available for a certain object.

3.2.2. Saving changes

All changes you make while in the EVMS Ncurses are only in memory until you save the changes. In order to make the changes permanent, save all changes before exiting. If you forget to save the changes and decide to exit the EVMS Ncurses interface, you will be reminded of the unsaved changes and be given the chance to save or discard the changes before exiting.

To explicitly save all changes, press **A + S** and confirm that you want to save changes.

3.3. EVMS Command Line Interpreter

The EVMS Command Line Interpreter (EVMS CLI) provides a command-driven user interface for EVMS. The EVMS CLI helps automate volume management tasks and provides an interactive mode in situations where the EVMS GUI is not available.

Because the EVMS CLI is an interpreter, it operates differently than command line utilities for the operating system. The options you specify on the EVMS CLI command line to invoke the EVMS CLI control how the EVMS CLI operates. For example, the command line options tell the CLI where to go for commands to interpret and how often the EVMS CLI must commit changes to disk. When invoked, the EVMS CLI prompts for commands.

The volume management commands the EVMS CLI understands are specified in the `/usr/src/evms-1.9.0/engine2/ui/cli/grammar.ps` file that accompanies the EVMS package. These commands are described in detail in the EVMS man page, and help on these commands is available from within the EVMS CLI.

3.3.1. Using the EVMS CLI

Use the **evms** command to start the EVMS CLI. If you do not enter an option with **evms**, the EVMS CLI starts in interactive mode. In interactive mode, the EVMS CLI prompts you for commands. The result of each command is immediately saved to disk. The EVMS CLI exits when you type **exit**. You can modify this behavior by using the following options with **evms**:

-b

This option indicates that you are running in batch mode and anytime there is a prompt for input from the user, the default value is accepted automatically. This is the default behavior with the **-f** option.

-c

This option commits changes to disk only when EVMS CLI exits, not after each command.

-f filename

This option tells the EVMS CLI to use *filename* as the source of commands. The EVMS CLI exits when it reaches the end of *filename*.

-p

This option only parses commands; it does not execute them. When combined with the **-f** option, the **-p** option detects syntax errors in command files.

-h

This option displays help information for options used with the **evms** command.

-rl

This option tells the CLI that all remaining items on the command line are replacement parameters for use with EVMS commands.



NOTE

Replacement parameters are accessed in EVMS commands using the $\$(x)$ notation, where *x* is the number identifying which replacement parameter to use. Replacement parameters are assigned numbers (starting with 1) as they are encountered on the command line. Substitutions are not made within comments or quoted strings.

An example would be:

```
evms -c -f testcase -rl sda sdb
```

sda is the replacement for *parameter1* and *sdb* is the replacement for *parameter2*



NOTE

Information on less commonly used options is available in the EVMS man page.

3.3.2. Notes on commands and command files

The EVMS CLI allows multiple commands to be displayed on a command line. When you specify multiple commands on a single command line, separate the commands with a colon (:). This is important for command files because the EVMS CLI sees a command file as a single long command line. The EVMS CLI has no concept of lines in the file and ignores spaces. These features allow a command in a command file to span several lines and use whatever indentation or margins that are convenient. The only requirement is that the command separator (the colon) be present between commands.

The EVMS CLI ignores spaces unless they occur within quote marks. Place in quotation marks a name that contains spaces or other non-printable or control characters. If the name contains a quotation mark as part of the name, the quotation mark must be "doubled," as shown in the following example:

```
"This is a name containing ""embedded"" quote marks."
```

EVMS CLI keywords are not case sensitive, but EVMS names are case sensitive. Sizes can be input in any units with a unit label, such as KB, MB, GB, or TB.

Finally, C programming language style comments are supported by the EVMS CLI. Comments can begin and end anywhere except within a quoted string, as shown in the following example:

```
/* This is a comment */  
Create:Vo/*This is a silly place for a comment, but it is  
allowed.*/lume,"lvm/Sample Container/My LVM  
Volume",compatibility
```

Chapter 4. The EVMS log file and error data collection

This chapter discusses the EVMS information and error log file and the various logging levels. It also explains how to change the logging level.

4.1. About the EVMS log file

The EVMS Engine creates a log file called `/var/log/evmsEngine.log` every time the Engine is opened. The Engine also saves copies of up to 10 previous Engine sessions in the files `/var/log/evmsEngine.n.log`, where `n` is the number of the session between 1 and 10.

4.2. Log file logging levels

There are several possible logging levels that you can choose to be collected in `/var/log/evmsEngine.log`. The "lowest" logging level, `critical`, collects only messages about serious system problems, whereas the "highest" level, `everything`, collects all logging related messages. When you specify a particular logging level, the Engine collects messages for that level and all the levels below it.

The following table lists the allowable log levels and the information they provide:

Table 4–1. EVMS logging levels

Level name	Description
Critical	The health of the system or the Engine is in jeopardy; for example, an operation has failed because there is not enough memory.
Serious	An operation did not succeed.
Error	The user has caused an error. The error messages are provided to help the user correct the problem.
Warning	An error has occurred that the system might or might not be able to work around.
Default	An error has occurred that the system has already worked around.
Details	Detailed information about the system.
Debug	Information that helps the user debug a problem.
Extra	More information that helps the user debug a problem than the "Debug" level provides.
Entry_Exit	Traces the entries and exits of functions.
Everything	Verbose output.

4.3. Specifying the logging levels

By default, when any of the EVMS interfaces is opened, the Engine logs the Default level of messages into the `/var/log/evmsEngine.log` file. However, if your system is having problems and you want to see more of what is happening, you can change the logging level to be higher; if you want fewer logging messages, you can change the logging level to be lower. To change the logging level, specify the `-d` parameter and the log level on the interface open call. The following examples show how to open the various interfaces with the highest logging level (everything):

```
GUI:          evmsgui -d everything
```

```
Ncurses:      evmsn -d everything
```

```
CLI:          evms -d everything
```



NOTE

If you use the EVMS mailing list for help with a problem, providing to us the log file that is created when you open one of the interfaces (as shown in the previous commands) makes it easier for us to help you.

The EVMS GUI lets you change the logging level during an Engine session. To do so, follow these steps:

1. Select Settings->Log Level->Engine.
2. Click the Level you want.

The CLI command, **probe**, opens and closes the Engine, which causes a new log to start. The log that existed before the **probe** command was issued is renamed `/var/log/evmsEngine.1.log` and the new log is named `/var/log/evmsEngine.log`.

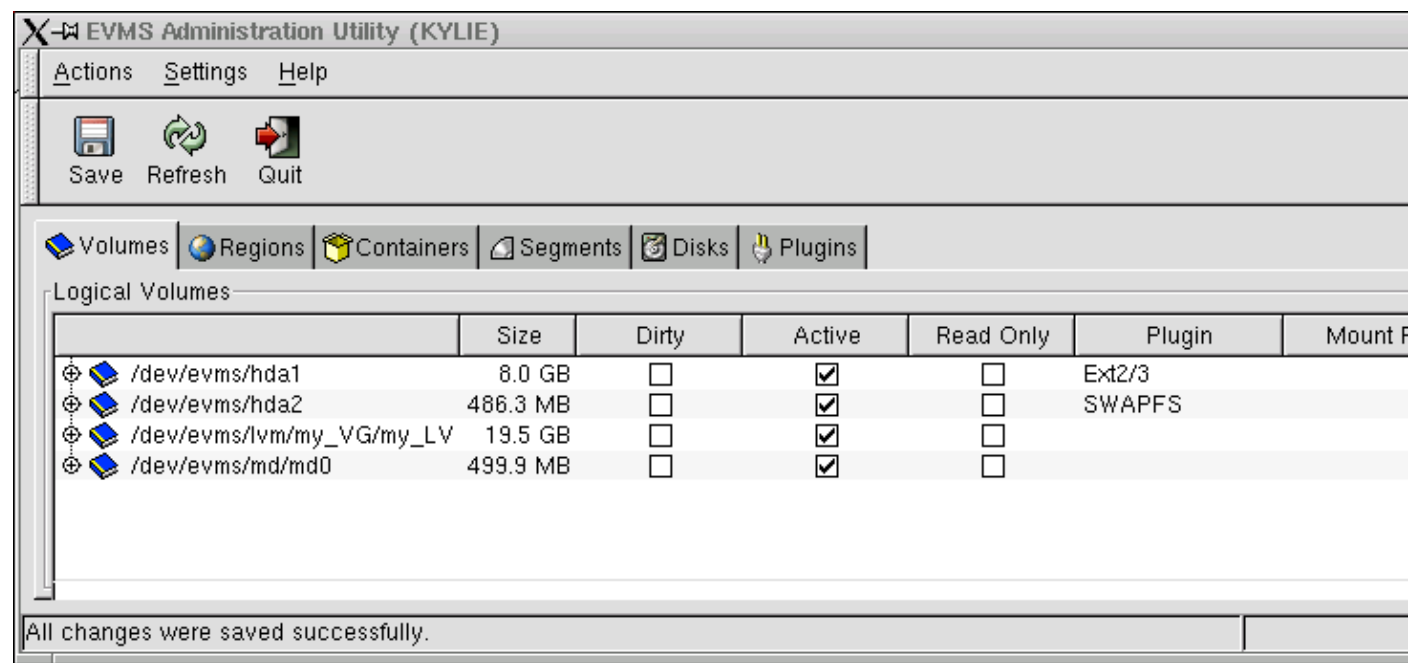
Chapter 5. Viewing compatibility volumes after migrating

Migrating to EVMS allows you to have the flexibility of EVMS without losing the integrity of your existing data. EVMS discovers existing volume management volumes as compatibility volumes. After you have installed EVMS, you can view your existing volumes with the interface of your choice.

5.1. Using the EVMS GUI

If you are using the EVMS GUI as your preferred interface, you can view your migrated volumes by typing **evmsgui** at the command prompt. The following window opens, listing your migrated volumes.

Figure 5–1. GUI start-up window



5.2. Using Ncurses

If you are using the Ncurses interface, you can view your migrated volumes by typing **evmsn** at the command prompt. The following window opens, listing your migrated volumes.

Figure 5–2. Ncurses start-up window

The screenshot shows a terminal window titled 'chavez@localhost: ~'. The menu bar includes 'File', 'Edit', 'View', 'Terminal', 'Go', and 'Help'. Below the menu bar, there are tabs for 'Actions' and 'Settings', with 'Settings' being the active tab. The main content area is titled 'Logical Volumes' and displays a table of logical volumes. The table has columns for 'Name', 'Size', 'Dirty', 'Active', 'R/O', 'Plug-in', and 'Mount'. Three volumes are listed: '/dev/evms/hda1' (101.9 MB, Active), '/dev/evms/hda2' (258.9 MB, Active), and '/dev/evms/hda3' (8.0 GB, Active). The bottom status bar indicates 'Press 'A' to access Actions menu; TAB key to cycle through views'.

Name	Size	Dirty	Active	R/O	Plug-in	Mount
/dev/evms/hda1	101.9 MB		X		Ext2/3	
/dev/evms/hda2	258.9 MB		X		SWAPFS	
/dev/evms/hda3	8.0 GB		X		Ext2/3	

Press 'A' to access Actions menu; TAB key to cycle through views

5.3. Using the CLI

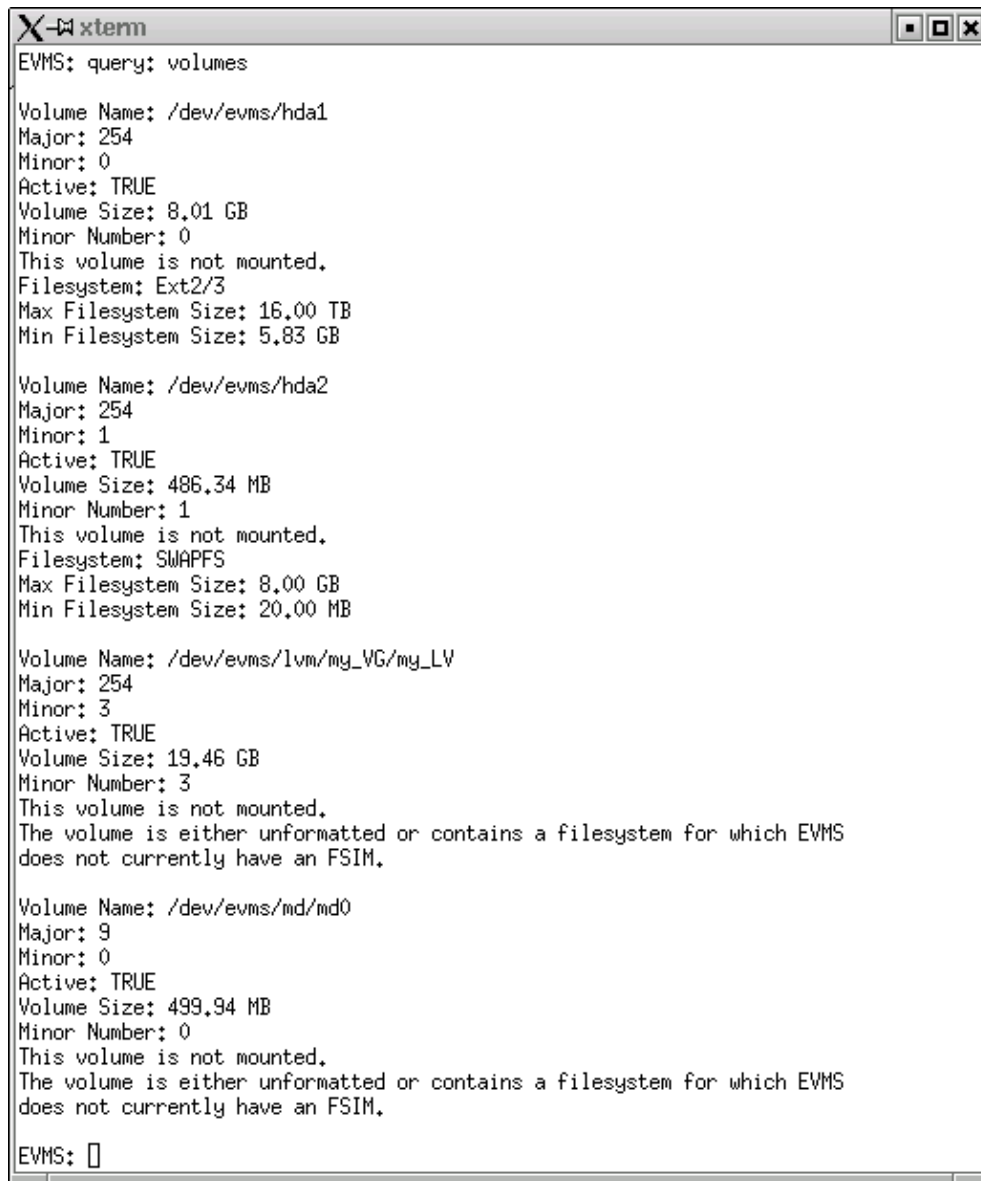
If you are using the Command Line Interpreter (CLI) interface, you can view your migrated volumes by following these steps:

1. Start the Command Line Interpreter by typing **evms** at the command line.
2. Query the volumes by typing the following at the EVMS prompt:

```
query:volumes
```

Your migrated volumes are displayed as results of the query.

Figure 5–3. CLI volume query results

A screenshot of an xterm window titled 'xterm'. The window displays the output of the 'EVMS: query: volumes' command. The output lists four volumes with their respective details: Volume Name, Major/Minor numbers, Active status, Volume Size, Minor Number, and Filesystem. Each volume entry is followed by a status message indicating it is not mounted and providing maximum and minimum filesystem size constraints. The volumes are: /dev/evms/hda1 (8.01 GB, Ext2/3), /dev/evms/hda2 (486.34 MB, SWAPFS), /dev/evms/lvm/my_VG/my_LV (19.46 GB), and /dev/evms/md/md0 (499.94 MB). The prompt 'EVMS: ' is visible at the bottom of the window.

```
EVMS: query: volumes

Volume Name: /dev/evms/hda1
Major: 254
Minor: 0
Active: TRUE
Volume Size: 8.01 GB
Minor Number: 0
This volume is not mounted.
Filesystem: Ext2/3
Max Filesystem Size: 16.00 TB
Min Filesystem Size: 5.83 GB

Volume Name: /dev/evms/hda2
Major: 254
Minor: 1
Active: TRUE
Volume Size: 486.34 MB
Minor Number: 1
This volume is not mounted.
Filesystem: SWAPFS
Max Filesystem Size: 8.00 GB
Min Filesystem Size: 20.00 MB

Volume Name: /dev/evms/lvm/my_VG/my_LV
Major: 254
Minor: 3
Active: TRUE
Volume Size: 19.46 GB
Minor Number: 3
This volume is not mounted.
The volume is either unformatted or contains a filesystem for which EVMS
does not currently have an FSIM.

Volume Name: /dev/evms/md/md0
Major: 9
Minor: 0
Active: TRUE
Volume Size: 499.94 MB
Minor Number: 0
This volume is not mounted.
The volume is either unformatted or contains a filesystem for which EVMS
does not currently have an FSIM.

EVMS: 
```

Chapter 6. Obtaining interface display details

The EVMS interfaces let you view more detailed information about an EVMS object than what is readily available from the main views of the EVMS user interfaces. The type and extent of additional information available is dependent on the interface you use. For example, the EVMS GUI provides more in-depth information than does the CLI.

The following sections show how to find detailed information on the region `lvm/Sample Container/Sample Region`, which is part of volume `/dev/evms/Sample Volume` (created in section 10.2).

6.1. Using the EVMS GUI

With the EVMS GUI, it is only possible to display additional details on an object through the Context Sensitive Menus, as shown in the following steps:

1. Looking at the volumes view, click the "+" next to volume `/dev/evms/Sample Volume`. Alternatively, look at the regions view.
 2. Right click `lvm/Sample Container/Sample Region`.
 3. Point at Display Details... and left click. A new window opens with additional information about the selected region.
 4. Click More by the Logical Extents box. Another window opens that displays the mappings of logical extents to physical extents.
-

6.2. Using Ncurses

Follow these steps to display additional details on an object with Ncurses:

1. Press **Tab** to reach the Storage Regions view.
 2. Scroll down using the **down** arrow until `lvm/Sample Container/Sample Region` is highlighted.
 3. Press **Enter**.
 4. In the context menu, scroll down using the **down** arrow to highlight "Display Details..."
 5. Press **Enter** to activate the menu item.
 6. In the Detailed Information dialog, use the **down** arrow to highlight the "Logical Extents" item and then use **spacebar** to open another window that displays the mappings of logical extents to physical extents.
-

6.3. Using the CLI

Use the **query** command (abbreviated **q**) with filters to display details about EVMS objects. There are two filters that are especially helpful for navigating within the command line: **list options** (abbreviated **lo**) and **extended info** (abbreviated **ei**).

The **list options** command tells you what can currently be done and what options you can specify. To use this command, first build a traditional query command starting with the command name **query**, followed by a colon (:), and then the type of object you want to query (for example, volumes, objects, plug-ins). Then, you can use filters to narrow the search to only the area you are interested in. For example, to determine the

acceptable actions at the current time on lvm/Sample Container/Sample Region, enter the following command:

```
query: regions, region="lvm/Sample Container/Sample Region", list options
```

The **extended info** filter is the equivalent of Display Details in the EVMS GUI and Ncurses interfaces. The command takes the following form: **query**, followed by a colon (:), the filter (**extended info**), a comma (,), and the object you want more information about. The command returns a list containing the field names, titles, descriptions and values for each field defined for the object. For example, to obtain details on lvm/Sample Container/Sample Region, enter the following command:

```
query: extended info, "lvm/Sample Container/Sample Region"
```

Many of the field names that are returned by the **extended info** filter can be expanded further by specifying the field name or names at the end of the command, separated by commas. For example, if you wanted additional information about logical extents, the query would look like the following:

```
query: extended info, "lvm/Sample Container/Sample Region", Extents
```

Chapter 7. Assigning a segment manager

This chapter discusses when to use a segment manager, what the different types of segment managers are, and how to assign a segment manager to a disk.

7.1. When to assign a segment manager

Assigning a segment manager to a disk allows the disk to be subdivided into smaller storage objects called disk segments. The **assign** command causes a segment manager to create appropriate metadata and expose freespace that the segment manager finds on the disk. You need to assign segment managers when you have a new disk or when you are switching from one partitioning scheme to another.

EVMS displays disk segments as the following types:

- Data: a set of contiguous sectors that has been allocated from a disk and can be used to construct a volume or object.
 - Freespace: a set of contiguous sectors that are unallocated or not in use. Freespace can be used to create a segment.
 - Metadata: a set of contiguous sectors that contain information needed by the segment manager.
-

7.2. Types of segment managers

There are five types of segment managers in EVMS: DOS, GPT, S/390, Cluster, and BSD.

7.2.1. DOS Segment Manager

The most commonly used segment manager is the DOS Segment Manager. This plug-in provides support for traditional DOS disk partitioning. The DOS Segment Manager also recognizes and supports the following variations of the DOS partitioning scheme:

- OS/2: an OS/2 disk has additional metadata sectors that contain information needed to reconstruct disk segments.
 - Embedded partitions: support for BSD, SolarisX86, and UnixWare is sometimes found embedded in primary DOS partitions. The DOS Segment Manager recognizes and supports these slices as disk segments.
-

7.2.2. GUID Partitioning Table (GPT) Segment Manager

The GUID Partitioning Table (GPT) Segment Manager handles the new GPT partitioning scheme on IA-64 machines. The Intel *Extensible Firmware Interface Specification* requires that firmware be able to discover partitions and produce logical devices that correspond to disk partitions. The partitioning scheme described in the specification is called GPT due to the extensive use of Globally Unique Identifier (GUID) tagging. GUID is a 128 bit long identifier, also referred to as a Universally Unique Identifier (UUID). As described in the Intel *Wired For Management Baseline Specification*, a GUID is a combination of time and space fields that produce an identifier that is unique across an entire UUID space. These identifiers are used extensively on GPT partitioned disks for tagging entire disks and individual partitions. GPT partitioned disks serve several functions, such as:

- keeping a primary and backup copy of metadata
- replacing msdos partition nesting by allowing many partitions
- using 64 bit logical block addressing
- tagging partitions and disks with GUID descriptors

The GPT Segment Manager scales better to large disks. It provides more redundancy with added reliability and uses unique names. However, the GPT Segment Manager is not compatible with DOS, OS/2, or Windows®.

7.2.3. S/390 Segment Manager

The S/390 Segment Manager is used exclusively on System/390 mainframes. The S/390 Segment Manager has the ability to recognize various disk layouts found on an S/390 machine, and provide disk segment support for this architecture. The two most common disk layouts are Linux Disk Layout (LDL) and Common Disk Layout (CDL).

The principle difference between LDL and CDL is that an LDL disk cannot be further subdivided. An LDL disk will produce a single metadata disk segment and a single data disk segment. There is no freespace on an LDL disk, and you cannot delete or re-size the data segment. A CDL disk can be subdivided into multiple data disk segments because it contains metadata that is missing from an LDL disk, specifically the Volume Table of Contents (vtoc) information.

The S/390 Segment Manager is the only segment manager plug-in capable of understanding the unique S/390 disk layouts. The S/390 Segment Manager cannot be assigned or unassigned from a disk.

7.2.4. Cluster segment manager

The cluster segment manager (CSM) supports high availability clusters. When the CSM is assigned to a shared storage disk, it writes metadata on the disk that:

- provides a unique disk ID (guid)
- names the EVMS container the disk will reside within
- specifies the cluster node (nodeid) that owns the disk
- specifies the HA cluster (clusterid)

This metadata allows the CSM to build containers for supporting failover situations. It does so by constructing an EVMS container object that consumes all shared objects discovered by the CSM and belonging to the same container. These shared storage objects are consumed by the container. A single segment object is produced by the container for each consumed storage object. A failover of the EVMS resource is then accomplished by simply reassigning the container to the standby cluster node and having that node re-run its discovery process.

7.2.5. BSD segment manager

BSD refers to the Berkeley Software Distribution UNIX® operating system. The EVMS BSD segment manager is responsible for recognizing and producing EVMS segment storage objects that map BSD partitions. A BSD disk may have a slice table in the very first sector on the disk for compatibility purposes with other operating systems. For example, a DOS slice table might be found in the usual MBR sector. The BSD disk would then be found within a disk slice that is located using the compatibility slice table. However,

BSD has no need for the slice table and can fully dedicate the disk to itself by placing the disk label in the very first sector. This is called a "fully dedicated disk" because BSD uses the entire disk and does not provide a compatibility slice table. The BSD segment manager recognizes such "fully dedicated disks" and provides mappings for the BSD partitions.

7.3. Assigning a segment manager to an existing disk

When you assign a segment manager to a disk, the segment manager needs to change the basic layout of the disk. This change means that some sectors are reserved for metadata and the remaining sectors are made available for creating data disk segments. Metadata sectors are written to disk to save information needed by the segment manager; previous information found on the disk is lost. Before assigning a segment manager to an existing disk, you must remove any existing volume management structures, including any previous segment manager.

7.4. Assigning a segment manager to a new disk

When a new disk is added to a system, the disk usually contains no data and has not been partitioned. If this is the case, the disk shows up in EVMS as a compatibility volume because EVMS cannot tell if the disk is being used as a volume. To assign a segment manager to the disk so that it can be subdivided into smaller disk segment objects, tell EVMS that the disk is not a compatibility volume by deleting the volume information.

If the new disk was moved from another system, chances are good that the disk already contains metadata. If the disk does contain metadata, the disk shows up in EVMS with storage objects that were produced from the existing metadata. Deleting these objects will allow you to assign a different segment manager to the disk, and you lose any old data.

7.5. Example: assign a segment manager

This section shows how to assign a segment manager with EVMS.

EVMS initially displays the physical disks it sees as volumes. Assume that you have added a new disk to the system that EVMS sees as `sde`. This disk contains no data and has not been subdivided (no partitions). EVMS assumes that this disk is a compatibility volume known as `/dev/evms/sde`.

Example 7–1. Assign the DOS Segment Manager

Assign the DOS Segment Manager to disk `sde`.



NOTE

In the following example, the DOS Segment Manager creates two segments on the disk: a metadata segment known as `sde_mbr`, and a segment to represent the available space on the drive, `sde_freespace1`. This freespace segment (`sde_freespace1`) can be divided into other segments because it represents space on the drive that is not in use.

7.5.1. Using the EVMS GUI

To assign the DOS Segment Manager to `sde`, first remove the volume, `/dev/evms/sde`:

1. Select Actions->Delete->Volume.
2. Select `/dev/evms/sde`.
3. Click Delete.

Alternatively, you can remove the volume through the GUI context sensitive menu:

1. From the Volumes tab, right click `/dev/evms/sde`.
2. Click Delete.

After the volume is removed, assign the DOS Segment Manager:

1. Select Actions->Add->Segment Manager to Storage Object.
2. Select DOS Segment Manager.
3. Click Next.
4. Select `sde`
5. Click Add

7.5.2. Using Ncurses

To assign the DOS Segment Manager to `sde`, first remove the volume `/dev/evms/sde`:

1. Select Actions->Delete->Segment Manager to Storage Object.
2. Select `/dev/evms/sde`.
3. Activate Delete.

Alternatively, you can remove the volume through the context sensitive menu:

1. From the Logical Volumes view, press **Enter** on `/dev/evms/sde`.
2. Activate Delete.

After the volume is removed, assign the DOS Segment Manager:

1. Select Actions->Add->Segment Manager to Storage Object
2. Select DOS Segment Manager.
3. Activate Next.
4. Select `sde`.
5. Activate Add.

7.5.3. Using the CLI

To assign the DOS Segment Manager to `sde`, first tell EVMS that this disk is not a volume and is available for use:

```
Delete:/dev/evms/sde
```

Next, assign the DOS Segment Manager to `sde` by typing the following:

```
Assign:DosSegMgr={ },sde
```

Chapter 8. Creating segments

This chapter discusses when to use segments and how to create them using different EVMS interfaces.

8.1. When to create a segment

A disk can be subdivided into smaller storage objects called disk segments. A segment manager plug-in provides this capability. Another reason for creating disk segments is to maintain compatibility on a dual boot system where the other operating system requires disk partitions. Before creating a disk segment, you must choose a segment manager plug-in to manage the disk and assign the segment manager to the disk. An explanation of when and how to assign segment managers can be found in [Chapter 7](#).

8.2. Example: create a segment

This section provides a detailed explanation of how to create a segment with EVMS by providing instructions to help you complete the following task:

Example 8–1. Create a 100MB segment

Create a 100MB segment from the freespace segment `sde_freespace1`. This freespace segment lies on a drive controlled by the DOS Segment Manager.

8.2.1. Using the EVMS GUI

To create a segment using the GUI, follow the steps below:

1. Select Actions->Create->Segment to see a list of segment manager plug-ins.
2. Select DOS Segment Manager. Click Next.

The next dialog window lists the free space storage objects suitable for creating a new segment.

3. Select `sde_freespace1`. Click Next.

The last dialog window presents the free space object you selected as well as the available configuration options for that object.

4. Enter 100 MB. Required fields are denoted by the "*" in front of the field description. The DOS Segment Manager provides default values, but you might want to change some of these values.

After you have filled in information for all the required fields, the Create button becomes available.

5. Click Create. A window opens to display the outcome.

Alternatively, you can perform some of the steps to create a segment from the GUI context sensitive menu:

1. From the Segments tab, right click on `sde_freespace1`.
 2. Click Create Segment...
 3. Continue beginning with step 4 of the GUI instructions.
-

8.2.2. Using Ncurses

To create a segment using Ncurses, follow these steps:

1. Select Actions->Create->Segment to see a list of segment manager plug-ins.
2. Select DOS Segment Manager. Activate Next.

The next dialog window lists free space storage objects suitable for creating a new segment.

3. Select `sde_freespacel`. Activate **Next**.
4. Highlight the size field and press **spacebar**.
5. At the "::-" prompt enter **100MB**. Press **Enter**.
6. After all required values have been completed, the Create button becomes available.
7. Activate Create.

Alternatively, you can perform some of the steps to create a segment from the context sensitive menu:

1. From the Segments view, press **Enter** on `sde_frespacel`.
2. Activate Create Segment.
3. Continue beginning with step 4 of the Ncurses instructions.

8.2.3. Using the CLI

To create a data segment from a freespace segment, use the **Create** command. The arguments the **Create** command accepts vary depending on what is being created. The first argument to the **Create** command indicates what is to be created, which in the above example is a segment. The remaining arguments are the freespace segment to allocate from and a list of options to pass to the segment manager. The command to accomplish this is:

```
Create: Segment,sde_freespacel, size=100MB
```



NOTE

The **Allocate** command also works to create a segment.

The previous example accepts the default values for all options you don't specify. To see the options for this command type:

```
query:plugins,plugin=DosSegMgr,list options
```

Chapter 9. Creating a container

This chapter discusses when and how to create a container.

9.1. When to create a container

Segments and disks can be combined to form a container. Containers allow you to combine storage objects and then subdivide those combined storage objects into new storage objects. You can combine storage objects to implement the volume group concept as found in the AIX and Linux logical volume managers.

Containers are the beginning of more flexible volume management. You might want to create a container in order to account for flexibility in your future storage needs. For example, you might need to add additional disks when your applications or users need more storage.

9.2. Example: create a container

This section provides a detailed explanation of how to create a container with EVMS by providing instructions to help you complete the following task.

Example 9–1. Create "Sample Container"

Given a system with three available disk drives (`sdc`, `sdd`, `hdc`), use the EVMS LVM Region Manager to combine these disk drives into a container called "Sample Container" with a PE size of 16 MB.

9.2.1. Using the EVMS GUI

To create a container using the EVMS GUI, follow these steps:

1. Select Actions→Create→Container to see a list plug-ins that support container creation.
2. Select the LVM Region Manager. Click Next.

The next dialog window contains a list of storage objects that the LVM Region Manager can use to create a container.

3. Select `sdc`, `sdd`, and `hdc` from the list. Click Next.
 4. Enter the name **Sample Container** for the container and **16MB** in the PE size field.
 5. Click Create. A window opens to display the outcome.
-

9.2.2. Using Ncurses

To create a container using the Ncurses interface, follow these steps:

1. Select Actions→Create→Container to see a list of plug-ins that support container creation.
2. Select the LVM Region Manager. Activate **Next**.

The next dialog window contains a list of storage objects that the LVM Region Manager can use to create the container.

3. Select `sd`, `sd`, and `hd` from the list. Activate `Next`.
 4. Press **spacebar** to select the field for the container name.
 5. Type **Sample Container** at the ":" prompt. Press **Enter**.
 6. Scroll down until `PE Size` is highlighted. Press **spacebar**.
 7. Scroll down until `16MB` is highlighted. Press **spacebar**.
 8. Activate `OK`.
 9. Activate `Create`.
-

9.2.3. Using the CLI

The **Create** command creates containers. The first argument in the **Create** command is the type of object to produce, in this case a container. The **Create** command then accepts the following arguments: the region manager to use along with any parameters it might need, and the segments or disks to create the container from. The command to complete the previous example is:

```
Create:Container,LvmRegMgr={name="Sample Container",pe_size=16MB},sd,sd,hd
```

The previous example accepts the default values for all options you don't specify. To see the options for this command type:

```
query:plugins,plugin=LvmRegMgr,list options
```

Chapter 10. Creating regions

Regions can be created from containers, but they can also be created from other regions, segments, or disks. Most region managers that support containers create one or more freespace regions to represent the freespace within the container. This function is analogous to the way a segment manager creates a freespace segment to represent unused disk space.

10.1. When to create regions

You can create regions because you want the features provided by a certain region manager or because you want the features provided by that region manager. You can also create regions to be compatible with other volume management technologies, such as MD or LVM. For example, if you wanted to make a volume that is compatible with Linux LVM, you would create a region out of a Linux LVM container and then a compatibility volume from that region.

10.2. Example: create a region

This section tells how to create a region with EVMS by providing instructions to help you complete the following task.

Example 10–1. Create "Sample Region"

Given the container "Sample Container," which has a freespace region of 8799 MB, create a data region 1000 MB in size named "Sample Region."

10.2.1. Using the EVMS GUI

To create a region, follow these steps:

1. Select Actions->Create->Region
2. Select the LVM Region Manager. Click Next.



NOTE

You might see additional region managers that were not in the selection list when you were creating the storage container because not all region managers are required to support containers.

3. Select the freespace region from the container you created in [Chapter 9](#). Verify that the region is named `lvm/Sample Container/Freespace`. Click Next.

The fields in the next window are the options for the LVM Region Manager plug-in, the options marked with an "*" are required.

4. Fill in the name, **Sample Region**.
5. Enter **1000MB** in the size field.
6. Click the Create button to complete the operation. A window opens to display the outcome.

Alternatively, you can perform some of the steps for creating a region with the GUI context sensitive menu:

1. From the Regions tab, right click lvm/Sample Container/Freespace.
2. Click Create Region.
3. Continue beginning with step 4 of the GUI instructions.

10.2.2. Using Ncurses

To create a region, follow these steps:

1. Select Actions->Create->Region.
2. Select the LVM Region Manager. Activate **Next**.
3. Select the freespace region from the container you created earlier in [Chapter 9](#). Verify that the region is named lvm/Sample Container/Freespace.
4. Scroll to the Name field, and press **spacebar**.
5. Type **Sample Region** at the ":" prompt. Press **Enter**.
6. Scroll to the size field, and press **spacebar**.
7. Type **1000MB** at the ":" prompt. Press **Enter**.
8. Activate Create.

Alternatively, you can perform some of the steps for creating a region with the context sensitive menu:

1. From the Storage Regions view, press **Enter** on lvm/Sample Container/Freespace.
2. Activate the Create Region menu item.
3. Continue beginning with step 4 of the Ncurses instructions.

10.2.3. Using the CLI

Create regions with the **Create** command. Arguments to the **Create** command are the following: keyword Region, the name of the region manager to use, the region managers options, and the objects to consume. The form of this command is:

```
Create:region, LvmRegMgr={name="Sample Region", size=1000MB},
"lvm/Sample Container/Freespace"
```

The LVM Region Manager supports many options for creating regions. To see the available options for creating regions and containers, use the following **Query**:

```
query:plugins,plugin=LvmRegMgr,list options
```

Chapter 11. Creating drive links

This chapter discusses the EVMS drive linking feature, which is implemented by the drive link plug-in, and tells how to create, expand, shrink, and delete a drive link.

11.1. What is drive linking?

Drive linking linearly concatenates objects, allowing you to create large storage objects and volumes from smaller individual pieces. For example, say you need a 1 GB volume but do not have contiguous space available of that length. Drive linking lets you link two or more objects together to form the 1 GB volume.

The types of objects that can be drive linked include disks, segments, regions, compatibility volumes, and other feature objects.

Any resizing of an existing drive link, whether to grow it or shrink it, must be coordinated with the appropriate file system operations. EVMS handles these file system operations automatically.

Because drive linking is an EVMS-specific feature that contains EVMS metadata, it is not backward compatible with other volume-management schemes.

11.2. How drive linking is implemented

The drive link plug-in consumes storage objects, called link objects, which produce a larger drive link object whose address space spans the link objects. The drive link plug-in knows how to assemble the link objects so as to create the exact same address space every time. The information required to do this is kept on each link child as persistent drive-link metadata. During discovery, the drive link plug-in inspects each known storage object for this metadata. The presence of this metadata identifies the storage object as a link object. The information contained in the metadata is sufficient to:

- Identify the link object itself.
- Identify the drive link storage object that the link object belongs to.
- Identify all link objects belonging to the drive link storage object
- Establish the order in which to combine the child link objects.

If any link objects are missing at the conclusion of the discovery process, the drive link storage object contains gaps where the missing link objects occur. In such cases, the drive link plug-in attempts to fill in the gap with a substitute link object and construct the drive link storage object in read-only mode, which allows for recovery action. The missing object might reside on removable storage that has been removed or perhaps a lower layer plug-in failed to produce the missing object. Whatever the reason, a read-only drive link storage object, together logging errors, help you take the appropriate actions to recover the drive link.

11.3. Creating a drive link

The drive link plug-in provides a list of acceptable objects from which it can create a drive-link object. When you create an EVMS storage object and then choose the drive link plug-in, a list of acceptable objects is provided that you can choose from. The ordering of the drive link is implied by the order in which you pick objects from the provided list. After you provide a name for the new drive-link object, the identified link objects are consumed and the new drive-link object is produced. There are no create objects.

Only the last object in a drive link can be expanded, shrunk or removed. Additionally, a new object can be added to the end of an existing drive link only if the file system (if one exists) permits. Any resizing of a drive link, whether to grow it or shrink it, must be coordinated with the appropriate file system operations. EVMS handles these file system operations automatically.

11.4. Example: create a drive link

This section shows how to create a drive link with EVMS:

Example 11–1. Create a drive link

Create a new drive link consisting of `sde4` and `hdc2`, and call it "dl."

11.4.1. Using the EVMS GUI

To create the drive link using the GUI, follow these steps:

1. Select Actions→Create→Feature Object to see a list of EVMS features.
2. Select Drive Linking Feature.
3. Click Next.
4. Click the objects you want to compose the drive link: `sde4` and `hdc2`.
5. Click Next.
6. Type `dl` in the "name" field
7. Click Create.

The last dialog window presents the free space object you selected as well as the available configuration options for that object.

Alternatively, you can perform some of the steps to create a drive link with the GUI context sensitive menu:

1. From the Available Objects tab, right click `sde4`.
 2. Click Create Feature Object...
 3. Continue creating the drive link beginning with step 2 of the GUI instructions. In step 4, `sde4` is selected for you. You can also select `hdc2`.
-

11.4.2. Using Ncurses

To create the drive link, follow these steps:

1. Select Actions→Create→Feature Object to see a list of EVMS features.
2. Select Drive Linking Feature.
3. Activate Next.
4. Use **spacebar** to select the objects you want to compose the drive link from: `sde4` and `hdc2`.
5. Activate Next.
6. Press **spacebar** to edit the Name field.
7. Type `dl` at the ":" prompt. Press **Enter**.
8. Activate Create.

Alternatively, you can perform some of the steps to create a drive link with the context sensitive menu:

1. From the Available Objects view, press **Enter** on `sde4`.
2. Activate the Create Feature Object menu item.
3. Continue creating the drive link beginning with step 4 of the Ncurses instructions. `sde4` will be pre-selected. You can also select `hdc2`.

11.4.3. Using the CLI

Use the **create** command to create a drive link through the CLI. You pass the "object" keyword to the **create** command, followed by the plug-in and its options, and finally the objects.

To determine the options for the plug-in you are going to use, issue the following command:

```
query: plugins, plugin=DriveLink, list options
```

Now construct the **create** command, as follows:

```
create: object, DriveLink={Name=dl}, sde4, hdc2
```

11.5. Expanding a drive link

A drive link is an aggregating storage object that is built by combining a number of storage objects into a larger resulting object. A drive link consumes link objects in order to produce a larger storage object. The ordering of the link objects as well as the number of sectors they each contribute is described by drive link metadata. The metadata allows the drive link plug-in to recreate the drive link, spanning the link objects in a consistent manner. Allowing any of these link objects to expand would corrupt the size and ordering of link objects; the ordering of link objects is vital to the correct operation of the drive link. However, expanding a drive link can be controlled by only allowing sectors to be added at the end of the drive link storage object. This does not disturb the ordering of link objects in any manner and, because sectors are only added at the end of the drive link, existing sectors have the same address (logical sector number) as before the expansion. Therefore, a drive link can be expanded by adding additional sectors in two different ways:

- By adding an additional storage object to the end of the drive link.
- By expanding the last storage object in the drive link.

If the expansion point is the drive link storage object, you can perform the expansion by adding an additional storage object to the drive link. This is done by choosing from a list of acceptable objects during the expand operation. Multiple objects can be selected and added to the drive link.

If the expansion point is the last storage object in the drive link, then you expand the drive link by interacting with the plug-in that produced the object. For example, if the link was a segment, then the segment manager plug-in that produced the storage object expands the link object. Afterwards, the drive link plug-in notices the size difference and updates the drive link metadata to reflect the resize of the child object.

There are no expand options.

11.6. Shrinking a drive link

Shrinking a drive link has the same restrictions as expanding a drive link. A drive link object can only be shrunk by removing sectors from the end of the drive link. This can be done in the following ways:

- By removing link objects from the end of the drive link.
- By shrinking the last storage object in the drive link.

The drive link plug-in attempts to orchestrate the shrinking of a drive-link storage object by only listing the last link object. If you select this object, the drive link plug-in then lists the next-to-last link object, and so forth, moving backward through the link objects to satisfy the shrink command.

If the shrink point is the last storage object in the drive link, then you shrink the drive link by interacting with the plug-in that produced the object.

There are no shrink options.

11.7. Deleting a drive link

A drive link can be deleted as long as it is not currently a compatibility volume, an EVMS volume, or consumed by another EVMS plug-in.

No options are available for deleting a drive link storage object.

Chapter 12. Creating snapshots

This chapter discusses snapshotting and tells how to create a snapshot.

12.1. What is a snapshot?

A snapshot represents a frozen image of a volume. The source of a snapshot is called an "original." When a snapshot is created, it looks exactly like the original at that point in time. As changes are made to the original, the snapshot remains the same and looks exactly like the original at the time the snapshot was created.

The snapshot feature is very useful for performing data backups. In order to perform a consistent backup, the volume that is being backed up should not change while the backup is running. This often means the volume must be taken offline during the backup, which can be a significant inconvenience to users. With snapshots, the volume can be kept online. A snapshot of the volume is created and the backup is taken from the snapshot, while the original remains in active use.

12.2. Creating and activating snapshot objects

Creating and activating a snapshot is a two-step process. The first step is to create the snapshot object. The snapshot object specifies where the saved data will be stored when changes are made to the original. The second step is to activate the object, which is to make an EVMS volume from the object.

12.2.1. Creating a snapshot

You can create a snapshot object from any unused storage object in EVMS (disks, segments, regions, or feature objects). The size of this consumed object is the size available to the snapshot object. The snapshot object can be smaller or larger than the original volume. If the object is smaller, the original volume could fill up as data is copied from the original to the snapshot, given sufficient activity on the original. In this situation, the snapshot is deactivated and additional I/O to the snapshot fails.

Base the size of the snapshot object on the amount of activity that is likely to take place on the original during the lifetime of the snapshot. The more changes that occur on the original and the longer the snapshot is expected to remain active, the larger the snapshot should be. Clearly, determining this calculation is not simple and requires trial and error to determine the correct snapshot object size to use for a particular situation. The goal is to create a snapshot object large enough to prevent the snapshot from being deactivated if it fills up, yet small enough to not waste disk space. If the snapshot object is the same size as the original volume (actually, a little larger, to account for the snapshot mapping tables), the snapshot is never deactivated.

12.2.2. Activating a snapshot

After you create a snapshot, activate it by making an EVMS volume from the object. After you create the volume and save the changes, the snapshot is active. The only option for activating snapshots is the name to give the EVMS volume. This name can be the same as or different than the name of the snapshot object.

12.3. Example: create a snapshot

This section shows how to create a snapshot with EVMS:

Example 12–1. Create a snapshot of a volume

Create a new snapshot of `/dev/evms/vol` on `lvm/Sample Container/Sample Region`, and call it "snap."

12.3.1. Using the EVMS GUI

To create the snapshot using the GUI, follow these steps:

1. Select Actions→Create→Feature Object to see a list of EVMS feature objects.
2. Select Snapshot Feature.
3. Click Next.
4. Select `lvm/Sample Container/Sample Region`.
5. Click Next.
6. Select `/dev/evms/vol` from the list in the "Volume to be Snapshotted" field.
7. Type `snap` in the "Snapshot Object Name" field.
8. Click Create.

Alternatively, you can perform some of the steps to create a snapshot with the GUI context sensitive menu:

1. From the Available Objects tab, right click `lvm/Sample Container/Sample Region`.
2. Click Create Feature Object...
3. Continue creating the snapshot beginning with step 2 of the GUI instructions. You can skip steps 4 and 5 of the GUI instructions.

12.3.2. Using Ncurses

To create the snapshot, follow these steps:

1. Select Actions→Create→Feature Object to see a list of EVMS feature objects.
2. Select Snapshot Feature.
3. Activate Next.
4. Select `lvm/Sample Container/Sample Region`.
5. Activate Next.
6. Press **spacebar** to edit the "Volume to be Snapshotted" field.
7. Highlight `/dev/evms/vol` and press **spacebar** to select.
8. Activate OK.
9. Highlight "Snapshot Object Name" and press **spacebar** to edit.
10. Type **snap** at the "::<" prompt. Press **Enter**.
11. Activate Create.

Alternatively, you can perform some of the steps to create a snapshot with the context sensitive menu:

1. From the Available Objects view, press **Enter** on lvm/Sample Container/Sample Region.
 2. Activate the Create Feature Object menu item.
 3. Continue creating the snapshot beginning with step 6 of the Ncurses instructions.
-

12.3.3. Using the CLI

Use the **create** command to create a snapshot through the CLI. You pass the "Object" keyword to the **create** command, followed by the plug-in and its options, and finally the objects.

To determine the options for the plug-in you are going to use, issue the following command:

```
query: plugins, plugin=Snapshot, list options
```

Now construct the **create** command, as follows:

```
create: object, Snapshot={original=/dev/evms/vol, snapshot=snap},  
"lvm/Sample Container/Sample Region"
```

12.4. Reinitializing a snapshot

Snapshots can be reinitialized, which causes all of the saved data to be erased and starts the snapshot from the current point in time. A reinitialized snapshot has the same original, chunk size, and writeable flags as the original snapshot.

To reinitialize a snapshot, delete the EVMS volume from the snapshot without deleting the snapshot object. Then create a new EVMS volume from the snapshot object.

12.5. Deleting a snapshot

When a snapshot is no longer needed, you can remove it by deleting the EVMS volume from the snapshot object, and then deleting the snapshot object. Because the snapshot saved the initial state of the original volume (and not the changed state), the original is always up-to-date and does not need any modifications when a snapshot is deleted.

No options are available for deleting snapshots.

12.6. Rolling back a snapshot

Situations can arise where a user wants to restore the original volume to the saved state of the snapshot. This action is called a rollback. One such scenario is if the data on the original is lost or corrupted. Snapshot rollback acts as a quick backup and restore mechanism, and allows the user to avoid a more lengthy restore operation from tapes or other archives.

Another situation where rollback can be particularly useful is when you are testing new software. Before you install a new software package, create a writeable snapshot of the target volume. You can then install the software to the snapshot volume, instead of to the original, and then test and verify the new software on the snapshot. If the testing is successful, you can then roll back the snapshot to the original and effectively install the software on the regular system. If there is a problem during the testing, you can simply delete the snapshot

without harming the original volume.

Rollback can only be performed when both the snapshot and the original volumes are unmounted and otherwise not in use. Rollback can also be performed only when there is only a single snapshot of an original. If an original has multiple snapshots, all but the desired snapshot must be deleted before rollback can take place.

No options are available for rolling back snapshots.

Chapter 13. Creating volumes

This chapter discusses when and how to create volumes.

13.1. When to create a volume

EVMS treats volumes and storage objects separately. A storage object does not automatically become a volume; it must be made into a volume.

Volumes are created from storage objects. Volumes are either EVMS native volumes or compatibility volumes. Compatibility volumes are intended to be compatible with a volume manager other than EVMS, such as the Linux LVM, MD, OS/2 or AIX. Compatibility volumes might have restrictions on what EVMS can do with them. EVMS native volumes have no such restrictions, but they can be used only by an EVMS equipped system. Volumes are mountable and can contain file systems.

EVMS native volumes contain EVMS-specific information to identify the name and minor number. After this volume information is applied, the volume is no longer fully backwards compatible with existing volume types.

Instead of adding EVMS metadata to an existing object, you can tell EVMS to make an object directly available as a volume. This type of volume is known as a compatibility volume. Using this method, the final product is fully backwards-compatible with the desired system.

13.2. Example: create an EVMS native volume

This section provides a detailed explanation of how to create an EVMS native volume with EVMS by providing instructions to help you complete the following task.

Example 13–1. Create an EVMS native volume

Create an EVMS native volume called "Sample Volume" from the region, /lvm/Sample Container/Region, you created in [Chapter 10](#).

13.2.1. Using the EVMS GUI

Follow these instructions to create an EVMS volume:

1. Select Actions->Create->EVMS Volume.
2. Choose lvm/Sample Container/Sample Region.
3. Type **sample volume** in the name field.
4. Click Create.

Alternatively, you can perform some of the steps to create an EVMS volume from the GUI context sensitive menu:

1. From the Available Options tab, right click lvm/Sample Container/Sample Region.

2. Click Create EVMS Volume...
3. Continue beginning with step 3 of the GUI instructions.

13.2.2. Using Ncurses

To create a volume, follow these steps:

1. Select Actions->Create->EVMS Volume.
2. Enter **sample volume** at the "name" prompt. Press **Enter**.
3. Activate Create.

Alternatively, you can perform some of the steps to create an EVMS volume from the context sensitive menu:

1. From the Available Objects view, press **Enter** on lvm/Sample Container/Sample Region.
2. Activate the Create EVMS Volume menu item.
3. Continue beginning with step 3 of the Ncurses instructions.

13.2.3. Using the CLI

To create a volume, use the **Create** command. The arguments the **Create** command accepts vary depending on what is being created. In the case of the example, the first argument is the key word `volume` that specifies what is being created. The second argument is the object being made into a volume, in this case `lvm/Sample Container/Sample Region`. The third argument is type specific for an EVMS volume, `Name=`, followed by what you want to call the volume, in this case `Sample Volume`. The following command creates the volume from the example.

```
Create: Volume, "lvm/Sample Container/Sample Region", Name="Sample Volume"
```

13.3. Example: create a compatibility volume

This section provides a detailed explanation of how to create a compatibility volume with EVMS by providing instructions to help you complete the following task.

Example 13–2. Create a compatibility volume

Create a compatibility volume called "Sample Volume" from the region, `/lvm/Sample Container/Region`, you created in [Chapter 10](#).

13.3.1. Using the GUI

To create a compatibility volume, follow these steps:

1. Select Actions->Create->Compatibility Volume.
2. Choose the region `lvm/Sample Container/Sample Region` from the list.
3. Click the Create button.
4. Click the Volume tab in the GUI to see a volume named `/dev/evms/lvm/Sample Container/Sample Region`. This volume is your compatibility volume.

Alternatively, you can perform some of the steps to create a compatibility volume from the GUI context sensitive menu:

1. From the Available Objects tab, right click lvm/Sample Container/Sample Region.
 2. Click Create Compatibility Volume...
 3. Continue beginning with step 3 of the GUI instructions.
-

13.3.2. Using Ncurses

To create a compatibility volume, follow these steps:

1. Select Actions->Create->Compatibility Volume.
2. Choose the region lvm/Sample Container/Storage Region from the list..
3. Activate Create.

Alternatively, you can perform some of the steps to create a compatibility volume from the context sensitive menu:

1. From the Available Objects view, press **Enter** on lvm/Sample Container/Sample Region.
 2. Activate the Create Compatibility Volume menu item.
 3. Continue beginning with step 3 of the Ncurses instructions.
-

13.3.3. Using the CLI

To create a volume, use the **Create** command. The arguments the **Create** command accepts vary depending on what is being created. In the case of the example, the first argument is the key word `volume` that specifies what is being created. The second argument is the object being made into a volume, in this case `lvm/Sample Container/Sample Region`. The third argument, `compatibility`, indicates that this is a compatibility volume and should be named as such.

```
Create:Volume,"lvm/Sample Container/Sample Region",compatibility
```

Chapter 14. FSIMs and file system operations

This chapter discusses the five File System Interface Modules (FSIMs) shipped with EVMS, and then provides examples of adding file systems and coordinating file system checks with the FSIMs.

14.1. The FSIMs supported by EVMS

EVMS currently ships with five FSIMs. These file system modules allow EVMS to interact with file system utilities such as **mkfs** and **fsck**. Additionally, the FSIMs ensure that EVMS safely performs operations, such as expanding and shrinking file systems, by coordinating these actions with the file system.

You can invoke operations such as **mkfs** and **fsck** through the various EVMS user interfaces. Any actions you initiate through an FSIM are not committed to disk until the changes are saved in the user interface. Later in this chapter we provide examples of creating a new file system and coordinating file system checks through the EVMS GUI, Ncurses, and command-line interfaces.

The FSIMs supported by EVMS are:

- JFS
 - XFS
 - ReiserFS
 - Ext2/3
 - SWAPFS
-

14.1.1. JFS

The JFS module supports the IBM journaling file system (JFS). Current support includes **mkfs**, **unmkfs**, **fsck**, and online file system expansion. Support for external logging will be added in a future release of EVMS. You must have at least version 1.0.9 of the JFS utilities for your system to work with this EVMS FSIM. You can download the latest utilities from the [JFS for Linux](#) site.

For more information on the JFS FSIM, refer to [Appendix F](#).

14.1.2. XFS

The XFS FSIM supports the XFS file system from SGI. Command support includes **mkfs**, **unmkfs**, **fsck**, and online expansion. Use version 1.2 or higher, which you can download from [the SGI open source FTP directory](#).

For more information on the XFS FSIM, refer to [Appendix G](#).

14.1.3. ReiserFS

The ReiserFS module supports the ReiserFS journaling file system. This module supports **mkfs**, **unmkfs**, **fsck**, online and offline expansion and offline shrinkage. You need version 3.x.1a or higher of the ReiserFS utilities for use with the EVMS FSIM modules. You can download the ReiserFS utilities from [The Naming System Venture \(Namesys\)](#) web site.

For more information on the ReiserFS FSIM, refer to [Appendix H](#).

14.1.4. Ext2/3

The EXT2/EXT3 FSIM supports both the ext2 and ext3 file system formats. The FSIM supports **mkfs**, **unmkfs**, **fsck**, and offline shrinkage and expansion.

For more information on the Ext2/3 FSIM, refer to [Appendix I](#).

14.1.5. SWAPFS

The SWAPFS FSIM supports Linux swap devices. The FSIM lets you create and delete swap devices, and supports **mkfs**, **unmkfs**, shrinkage and expansion. Currently, you are responsible for issuing the **swapon** and **swapoff** commands either in the startup scripts or manually. You can resize swap device with the SWAPFS FSIM as long as the device is not in use.

14.2. Example: add a file system to a volume

After you have made an EVMS or compatibility volume, add a file system to the volume before mounting it. You can add a file system to a volume through the EVMS interface of your choice.

Example 14–1. Add a JFS File System to a Volume

This example creates a new JFS file system, named `jfs_vol`, on volume `/dev/evms/my_vol`.

14.2.1. Using the EVMS GUI

Follow these steps to create a JFS file system with the EVMS GUI:

1. Select Actions->File Systems->Make.
2. Select JFS File System Interface Module.
3. Click Next.
4. Select `/dev/evms/my_vol`.
5. Click Next.
6. Type **jfs_vol** in the "Volume Label" field. Customize any other options you are interested in.
7. Click Make.
8. The operation is completed when you save.

Alternatively, you can perform some of the steps to create a file system with the GUI context sensitive menu:

1. From the Volumes tab, right click `/dev/evms/my_vol`.
 2. Click Make Filesystem...
 3. Continue creating the file system beginning with step 2 of the GUI instructions. You can skip steps 4 and 5 of the GUI instructions.
-

14.2.2. Using Ncurses

Follow these steps to create a JFS file system with Ncurses:

1. Select Actions->File Systems->Make.
2. Select JFS File System Interface Module.
3. Activate Next.
4. Select `/dev/evms/my_vol`.
5. Activate Next.
6. Scroll down using the **down** arrow until Volume Label is highlighted.
7. Press **Spacebar**.
8. At the "::<" prompt enter `jfs_vol`.
9. Press **Enter**.
10. Activate Make.

Alternatively, you can perform some of the steps to create a file system with the context sensitive menu:

1. From the Volumes view, press **Enter** on `/dev/evms/my_vol`.
2. Activate the Make Filesystem menu item.
3. Continue creating the file system beginning with step 2 of the Ncurses instructions.

14.2.3. Using the CLI

Use the **mkfs** command to create the new file system. The arguments to **mkfs** include the FSIM type (in our example, JFS), followed by any option pairs, and then the volume name. The command to accomplish this is:

```
mkfs: JFS={vollabel=jfs_vol}, /dev/evms/my_vol
```

The command is completed upon saving.

If you are interested in other options that **mkfs** can use, look at the results of the following query:

```
query: plugins, plugin=JFS, list options
```

14.3. Example: check a file system

You can also coordinate file system checks from the EVMS user interfaces.

Example 14–2. Check a JFS File System

This example shows how to perform a file system check on a JFS file system, named `jfs_vol`, on volume `/dev/evms/my_vol`, with verbose output.

14.3.1. Using the EVMS GUI

Follow these steps to check a JFS file system with the EVMS GUI:

1. Select Actions->File Systems->Check/Repair.
2. Select `/dev/evms/my_vol`.
3. Click Next.
4. Click the **Yes** button by Verbose Output. Customize any other options you are interested in.
5. Click Check.
6. The operation is completed when you save.

Alternatively, you can perform some of the steps to check a file system with the GUI context sensitive menu:

1. From the Volumes tab, right click `/dev/evms/my_vol`.
2. Click Check/Repair File System...
3. Continue checking the file system beginning with step 3 of the GUI instructions.

14.3.2. Using Ncurses

Follow these steps to check a JFS file system with Ncurses:

1. Select Actions->File System->Check/Repair
2. Select `/dev/evms/my_vol`.
3. Activate Next.
4. Scroll down using the **down** arrow until Vebose Output is highlighted.
5. Press **Spacebar** to change Verbose Output to **Yes**.
6. Activate Check.

Alternatively, you can perform some of the steps to check a file system with the context sensitive menu:

1. From the Volumes view, press **Enter** on `/dev/evms/my_vol`.
2. Activate the Check/Repair File System menu item.
3. Continue checking the file system beginning with step 3 of the Ncurses instructions.

14.3.3. Using the CLI

The CLI **check** command takes a volume name and options as input. The command to check the file system on `/dev/evms/my_vol` is the following:

```
check: /dev/evms/my_vol, verbose=TRUE
```

Currently, a query command for viewing additional options is not available.

Chapter 15. Clustering operations

This chapter discusses how to configure cluster storage containers (referred to throughout this chapter as "cluster containers"), a feature provided by the EVMS Cluster Segment Manager (CSM).

Disks that are physically accessible from all of the nodes of the cluster can be grouped together as a single manageable entity. EVMS storage objects can then be created using storage from these containers.

Ownership is assigned to a container to make the container either private or shared. A container that is owned by any one node of the cluster is called a private container. EVMS storage objects and storage volumes created using space from a private container are accessible from only the owning node.

A container that is owned by all the nodes in a cluster is called a shared container. EVMS storage objects and storage volumes created using space from a shared container are accessible from all nodes of the cluster simultaneously.

EVMS provides the tools to convert a private container to a shared container, and a shared container to a private container. EVMS also provides the flexibility to change the ownership of a private container from one cluster node to another cluster node.

15.1. Rules and restrictions for creating cluster containers

Note the following rules and limitations for creating cluster containers:

- Do not assign non-shared disks to a cluster container.
- Storage objects created on a cluster container must not span across multiple cluster containers.
Currently, the EVMS Engine cannot enforce this rule, so you must ensure that objects and volumes created from cluster storage manager segments do not span multiple containers.
- A disk should not span cluster containers.
- Do not assign RAID, snapshot, and BBR features to storage objects on a cluster container.

15.2. Example: create a private cluster container

This section tells how to create a sample private container and provides instructions for completing the following task:

Example 15–1. Create a private cluster container

Given a system with three available shared disks (`sdd`, `sde`, and `sdf`), use the EVMS Cluster Segment Manager to combine these disk drives into a container called `Priv1` owned by `node1`.

15.2.1. Using the EVMS GUI

To create a container with the EVMS GUI, follow these steps:

1. Select Actions->Create->Container to see a list of plug-ins that support container creation.
2. Select the Cluster Segment Manager.
3. Click Next.

The next dialog window contains a list of storage objects that the CSM can use to create a container.

4. Select `sdd`, `sde`, and `sdf` from the list.
5. Click Next.
6. In the first pull-down menu, select the "Node Id" of the cluster node that owns this container (node1).
Select "Storage Type" as private from the second pull-down menu.
7. Enter the name **Priv1** for the Container Name.
8. Click Create.

A window opens that displays the outcome.

9. Commit the changes.

15.2.2. Using Ncurses

To create the private container with the Ncurses interface, follow these steps:

1. Select Actions->Create->Container to see a list of plug-ins that support container creation.
2. Scroll down with the **down** arrow and select Cluster Segment Manager by pressing **spacebar**. The plug-in you selected is marked with an "x."
3. Press **Enter**.

The next submenu contains a list of disks that the Cluster Segment Manager finds acceptable to use for the creation of a container.

4. Use **spacebar** to select `sdd`, `sde`, and `sdf` from the list. The disks you select are marked with an "x."
5. Press **Enter**.
6. On the Create Storage Container – Configuration Options menu, press **spacebar** on the Node Id, which will provide a list of nodes from which to select.
7. Press **spacebar** on the node `node1` and then press **Enter**.
8. Scroll down with the **down** arrow and press **spacebar** on the Storage Type. A list of storage types opens.
9. Scroll down with the **down** arrow to private entry and press **spacebar**.
10. Press **Enter**.
11. Scroll down with the **down** arrow to Container Name and press **spacebar**.

The Change Option Value menu opens and asks for the Container Name. Type in the name of the container as **Priv1**, and press **Enter**.

12. Press **Enter** to complete the operation.

15.2.3. Using the CLI

An operation to create a private cluster container with the CLI takes three parameters: the name of the container, the type of the container, and the nodeid to which the container belongs.

On the CLI, type the following command to create the private container `Priv1`:

```
create: container,CSM={name="Priv1",type="private",nodeid="node1"},sdd,sde,sdf
```

15.3. Example: create a shared cluster container

This section tells how to create a sample shared container and provides instructions to help you complete the following task:

Example 15–2. Create a shared cluster container

Given a system with three available shared disks (`sdd`, `sde`, and `sdf`), use the EVMS Cluster Segment Manager to combine these disk drives into a shared container called `Shar1`.

15.3.1. Using the EVMS GUI

To create a shared cluster container with the EVMS GUI, follow these steps:

1. Select Actions->Create->Container to see a list of plug-ins that support container creation.
2. Select the Cluster Segment Manager.
3. Click Next.

The next dialog window contains a list of storage objects that the CSM can use to create a container.

4. Select `sdd`, `sde`, and `sdf` from the list.
5. Click Next.
6. You do not need to change the "Node Id" field. Select Storage Type as shared from the second pull-down menu.
7. Enter the name **Shar1** for the Container Name.
8. Click Create. A window opens to display the outcome.
9. Commit the changes.
10. Quit the GUI and run **evms_activate** on each of the cluster nodes so that the nodes discover the volume. This process will be automated in future releases of EVMS.

15.3.2. Using Ncurses

To create a shared cluster contained with the Ncurses interface, follow these steps:

1. Select Actions->Create->Container to see a list of plug-ins that support container creation.
2. Scroll down with the **down** arrow and select Cluster Segment Manager by pressing **spacebar**. The plug-in you selected is marked with an "x."
3. Press **Enter**.

The next submenu contains a list of disks that the Cluster Segment Manager finds acceptable to use for the creation of a container.

4. Use **spacebar** to select `sdd`, `sde`, and `sdf` from the list. The disks you select are marked with an "x."
5. Press **Enter**.
6. The Create Storage Container – Configuration Options menu open; ignore the "Node Id" menu.
7. Scroll down with the **down** arrow and press **spacebar** on the Storage Type. A list of storage types opens.
8. Scroll down with the **down** arrow to `shared` entry and press **spacebar**.
9. Press **Enter**.
10. Scroll down with the **down** arrow to Container Name and press **spacebar**.

The Change Option Value menu opens and asks for the Container Name. Type in the name of the container as `Shar1`, and press **Enter**.

11. Press **Enter** to complete the operation.
12. Quit `Ncurses` and run `evms_activate` on each of the cluster nodes. This process will be automated in future releases of EVMS.

15.3.3. Using the CLI

An operation to create a shared cluster container with the CLI takes two parameters: the name of the container and the type of the container.

On the CLI, type the following command to create shared container `Shar1`:

```
create: container,CSM={name="Shar1",type="shared"},sdd,sde,sdf
```

Run `evms_activate` on each node of the cluster. This process will be automated in future releases of EVMS.

15.4. Example: convert a private container to a shared container

This section tells how to convert a sample private container to a shared container and provides instructions for completing the following task:

Example 15–3. Convert a private container to shared

Given a system with a private storage container `Priv1` owned by `evms1`, convert `Priv1` to a shared storage container with the same name.



CAUTION

Ensure that no application is using the volumes on the container on any node of the cluster.

15.4.1. Using the EVMS GUI

Follow these steps to convert a private cluster container to a shared cluster container with the EVMS GUI:

1. Select Actions->Modify->Container to see a list of containers.
2. Select the container `csn/Priv1` and press **Next**.

A Modify Properties dialog box opens.

3. Change "Type Field" to "shared" and click **Modify**.

A window opens that displays the outcome.

4. Commit the changes.
5. Quit the GUI and run **evms_activate** on all the cluster nodes so that the nodes discover all the volumes on the `csn/Priv1` container. This process will be automated in a future release of EVMS.

15.4.2. Using Ncurses

Follow these steps to convert a private cluster container to a shared cluster container with the Ncurses interface:

1. Select Actions->Modify->Container to see a list of containers.
2. The Modify Container Properties dialog opens. Select the container `csn/Priv1` by pressing **spacebar**. The container you selected is marked with an "x."

Press **Enter**.

3. Use **spacebar** to select `sdd`, `sde`, and `sdf` from the list. The disks you select are marked with an "x."
4. Press **Enter**.
5. The Modify Container Properties – Configuration Options" dialog opens. Scroll down with the **down** arrow and press **spacebar** on the "Type field".
6. Press **spacebar**.
7. The Change Option Value dialog opens. Type **shared** and press **Enter**.

The changed value is now displays in the Modify Container Properties – Configuration Options dialog.

8. Press **Enter**.

The outcome of the command is displayed at the bottom of the screen.

9. Save the changes by clicking **Save** in the Actions pulldown menu.
10. Quit Ncurses and run **evms_activate** on all the cluster nodes so that the nodes discover all the volumes on the `csn/Priv1` container. This process will be automated in a future release of EVMS.

15.4.3. Using the CLI

The **modify** command modifies the properties of a container. The first argument of the command is the object to modify, followed by its new properties. The command to convert the private container to a shared container in the example is:

```
modify: csn/Priv1,type=shared
```

Run **evms_activate** on all the cluster nodes so that the nodes discover all the volumes on the `csn/Priv1` container. This process will be automated in a future release of EVMS.

15.5. Example: convert a shared container to a private container

This section tells how to convert a sample shared container to a private container and provides instructions for completing the following task:

Example 15–4. Convert a shared container to private

Given a system with a shared storage container `Shar1`, convert `Shar1` to a private storage container owned by node `node1` (where `node1` is the nodeid of one of the cluster nodes).



CAUTION

Ensure that no application is using the volumes on the container of any node in the cluster.

15.5.1. Using the EVMS GUI

Follow these steps to convert a shared cluster container to a private cluster container with the EVMS GUI:

1. Select **Actions**→**Modify**→**Container** to see a list of containers.
2. Select the container `csn/Shar1` and press **Next**.

A **Modify Properties** dialog opens.

3. Change "Type Field" to "private" and the "NodeID" field to `node1`. Click **Modify**.

A window opens that displays the outcome.

4. Commit the changes.
5. Quit the GUI and run **evms_activate** on the other nodes to deactivate the volumes of the shared container on the other nodes. This process will be automated in a future release of EVMS.

15.5.2. Using Ncurses

Follow these steps to convert a shared cluster container to a private cluster container with the Ncurses interface:

1. Select **Actions**→**Modify**→**Container**
2. The **Modify Container Properties** dialog opens. Select the container `csn/Shar1` by pressing **spacebar**. The container you selected is marked with an "x."

Press **Enter**.

3. The **Modify Container Properties – Configuration Options** dialog opens. Scroll down with the **down** arrow and press **spacebar** on the "Type" field.
4. Press **spacebar**.
5. The **Change Option Value** dialog opens. Type **private** and press **Enter**.

6. The Modify Container Properties – Configuration Options dialog opens. Scroll down the list to NodeId with the **down** arrow and press **spacebar**.
7. The Change Option Value dialog opens. Enter **node1** and press **Enter**.
8. The changed values now display in the Modify Container Properties – Configuration Options dialog. Press **Enter**.

The outcome of the command is displayed at the bottom of the screen.

9. Save the changes by clicking **Save** in the Actions pulldown.
10. Quit Ncurses and run **evms_activate** on all the cluster nodes to deactivate the volumes of the shared container on all the other nodes. This process will be automated in a future release of EVMS.

15.5.3. Using the CLI

The **modify** command modifies the properties of a container. The first argument of the command is the object to modify, followed by its new properties. The command to convert the shared container to a private container in the example is:

```
modify: csm/Shar1,type=private,nodeid=node1
```

Run **evms_activate** on all the cluster nodes to deactivate the volumes of the shared container on all the other nodes. This process will be automated in a future release of EVMS.

15.6. Example: deport a private or shared container

When a container is deported, the node disowns the container and deletes all the objects created in memory that belong to that container. No node in the cluster can discover objects residing on a deported container or create objects for a deported container. This section explains how to deport a private or shared container.

Example 15–5. Deport a cluster container

Given a system with a private or shared storage container named **c1**, deport **c1**.

15.6.1. Using the EVMS GUI

To deport a container with the EVMS GUI, follow these steps:

1. Select Actions->Modify->Container.
 2. Select the container **csm/c1** and press **Next**.
- A Modify Properties dialog opens.
3. Change "Type Field" to "deported." Click **Modify**.
- A window opens that displays the outcome.
4. Commit the changes.



NOTE

If the deported container was a shared container, quit the GUI and then run **evms_activate** on each cluster node. This operation will be automated in a future release of EVMS.

15.6.2. Using Ncurses

To deport a container with Ncurses, follow these steps:

1. Scroll down the list with the **down** arrow to Modify. Press **Enter**.

A submeny is displayed.

2. Scroll down until Container is highlighted. Press **Enter**.

The Modify Container Properties dialog opens.

3. Select the container `csn/c1` by pressing **spacebar**. The container you selected is marked with an "x."
4. Press **Enter**.

The Modify Container Properties – Configuration Options dialog opens.

5. Scroll down and press **spacebar** on the "Type" field.
6. Press **spacebar**.

The Change Option Value dialog opens.

7. Type **deported** and press **Enter**.

The changed value is displayed in the Modify Container Properties – Configuration Options dialog.

8. Press **Enter**.

The outcome of the command is displayed at the bottom of the screen.

9. Commit the changes by clicking **Save** in the Actions pulldown.



NOTE

If the deported container was a shared container, quit Ncurses and then run **evms_activate** on each cluster node. This operation will be automated in a future release of EVMS.

15.6.3. Using the CLI

To deport a container from the CLI, execute the following command at the CLI prompt:

```
modify: csm/c1,type="deported"
```



NOTE

If the deported container was a shared container, run **evms_activate** on each cluster node. This operation will be automated in a future release of EVMS.

15.7. Deleting a cluster container

The procedure for deleting a cluster container is the same for deleting any container. See [Section 20.2](#)

15.8. Failover and Failback of a private container on Linux-HA

EVMS supports the Linux-HA cluster manager in EVMS V2.0 and later; support for the RSCT cluster manager will be made available in a future release of EVMS.



NOTE

Ensure that **evms_activate** is called in one of the startup scripts before the **heartbeat** startup script is called. If **evms_activate** is not called, failover might not work correctly.

Follow these steps to set up failover and failback of a private container:

1. Add an entry in `/etc/ha.d/haresources` for each private container to be failed over. For example, if `container1` and `container2` are to be failed over together to the same node with `node1` as the owning node, add the following entry to `/etc/ha.d/haresources`:

```
node1 evms_failover::container1 evms_failover::container2
```

`node1` is the cluster node that owns this resource. The resource is failed over to the other node when `node1` dies.

Similarly, if `container3` and `container4` are to be failed over together to the same node with `node2` as the owning node, then add the following entry to `/etc/ha.d/haresources`:

```
node2 evms_failover::container3 evms_failover::container4
```

Refer to the following source for more details on the semantics of resource groups:
<http://www.linux-ha.org/download/GettingStarted.html>.

2. Validate that the `/etc/ha.d`, `/etc/ha.cf` and `/etc/ha.d/haresources` files are the same on all the nodes of the cluster.
3. The heartbeat cluster manager must be restarted, as follows, after the `/etc/ha.d/haresources` file has been changed:

```
/etc/init.d/heartbeat restart
```



NOTE

Do not add shared containers to the list of failover resources; doing so causes EVMS to respond unpredictably.

15.9. Remote configuration management

EVMS supports the administration of cluster nodes by any node in the cluster. For example, storage on remote cluster node `node1` can be administered from cluster node `node2`. The following sections show how to set up remote administration through the various EVMS user interfaces.

15.9.1. Using the EVMS GUI

To designate node2 as the node to administer from the GUI, follow these steps:

1. Select Settings->Node Administered...
2. Select node2.
3. Click **Administer** to switch to the new node.

The discovery of the remote configuration is initiated and the status bar displays the message "Now administering node node2," which indicates that the GUI is switched over and node node.

15.9.2. Using Ncurses

To designate node2 as the node to administer from Ncurses, follow these steps:

1. Go to the Settings pulldown menu.
2. Scroll down with the **down** arrow to the "Node Administered" option and press **Enter**.
3. The Administer Remote Node dialog opens. Select node2 and press **spacebar**.

The node you selected is marked with an "x."

4. Press **Enter**.
5. The "EVMS is examining your system. Please wait" dialog opens. After a while you will be switched over to the node node2.

15.9.3. Using the CLI

To designate node2 as a node administrator from the CLI, issue this command:

```
evms -n node2
```

15.10. Forcing a cluster container to be imported

A private container and its objects are made active on a node if:

- the private container is owned by the node
- the container is not deported
- the node currently has quorum

Similarly, a shared container and its objects are made active on a node if the node currently has quorum. However, the administrator can force the importation of private and shared containers by overriding these rules, as follows:



NOTE

Use extreme caution when performing this operation by ensuring that the node on which the cluster container resides is the only active node in the cluster. Otherwise, the data in volumes on shared and private containers on the node can get corrupted.

1. Enabling the maintenance mode in the **evms.conf** file. The option to modify in the **evms.conf** file is the following:

```
# cluster segment manager section
csm {
#     admin_mode=yes  # values are: yes or no
                        # The default is no. Set this key to
                        # yes when you wish to force the CSM
                        # to discover objects from all cluster
                        # containers, allowing you to perform
                        # configuration and maintenance. Setting
                        # admin_mode to yes will cause the CSM
                        # to ignore container ownership, which
                        # will allow you to configure storage
                        # in a maintenance mode.
```

2. Running **evms_activate** on the node.
-

Chapter 16. Converting volumes

This chapter discusses converting compatibility volumes to EVMS volumes and converting EVMS volumes to compatibility volumes. For a discussion of the differences between compatibility and EVMS volumes, see [Chapter 13](#).

16.1. When to convert volumes

There are several different scenarios that might help you determine what type of volumes you need. For example, if you wanted persistent names or to make full use of EVMS features, such as BBR, Drive Linking, or Snapshotting, you would convert your compatibility volumes to EVMS volumes. In another situation, you might decide that a volume needs to be read by a system that understands the underlying volume management scheme. In this case, you would convert your EVMS volume to a compatibility volume.

A volume can only be converted when it is offline. This means the volume must be unmounted and otherwise not in use. The volume must be unmounted because the conversion operation changes both the name and the device number of the volume. Once the volume is converted, you can remount it using its new name.

16.2. Example: convert compatibility volumes to EVMS volumes

A compatibility volume can be converted to an EVMS volume in the following situations:

- The compatibility volume has no file system (FSIM) on it.
- The compatibility volume has a file system, but the file system can be shrunk (if necessary) to make room for the EMVS metadata.

This section provides a detailed explanation of how to convert compatibility volumes to EVMS volumes and provides instructions to help you complete the following task.

Example 16–1. Convert a compatibility volume

You have a compatibility volume `/dev/evms/hda3` that you want to make into an EVMS volume named `my_vol`.

16.2.1. Using the EVMS GUI

Follow these steps to convert a compatibility volume with the EVMS GUI:

1. Choose Action→Convert →Compatibility Volume to EVMS.
2. Select `/dev/evms/hda3` from the list of available volumes.
3. Type `my_vol` in the name field.
4. Click the Convert button to convert the volume.

Alternatively, you can perform some of the steps to convert the volume from the GUI context sensitive menu:

1. From the Volumes tab, right click on `/dev/evms/hda3`.
 2. Click Convert to EVMS Volume...
 3. Continue to convert the volume beginning with step 3 of the GUI instructions.
-

16.2.2. Using Ncurses

Follow these instructions to convert a compatibility volume to an EVMS volume with the Ncurses interface:

1. Choose Actions→Convert→Compatibility Volume to EVMS Volume
2. Select `/dev/evms/hda3` from the list of available volumes.
3. Type **my_vol** when prompted for the name. Press **Enter**.
4. Activate Convert.

Alternatively, you can perform some of the steps to convert the volume from the context sensitive menu:

1. From the Volumes view, press **Enter** on `/dev/evms/hda3`.
 2. Activate the Convert to EVMS Volume menu item.
 3. Continue to convert the volume beginning with step 3 of the Ncurses instructions.
-

16.2.3. Using the CLI

To convert a volume, use the **Convert** command. The **Convert** command takes the name of a volume as its first argument, and then `name=` for what you want to name the new volume as the second argument. To complete the example and convert a volume, type the following command at the EVMS : prompt:

```
convert: /dev/evms/hda3, Name=my_vol
```

16.3. Example: convert EVMS volumes to compatibility volumes

An EVMS volume can be converted to a compatibility volume only if the volume does not have EVMS features on it. This section provides a detailed explanation of how to convert EVMS volumes to compatibility volumes by providing instructions to help you complete the following task.

Example 16–2. Convert an EVMS volume

You have an EVMS volume, `/dev/evms/my_vol`, that you want to make a compatibility volume.

16.3.1. Using the EVMS GUI

Follow these instructions to convert an EVMS volume to a compatibility volume with the EVMS GUI:

1. Choose Action→Convert→EVMS Volume to Compatibility Volume.
2. Select `/dev/evms/my_vol` from the list of available volumes.
3. Click the Convert button to convert the volume.

Alternatively, you can perform some of the steps to convert the volume through the GUI context sensitive menu:

1. From the Volumes tab, right click `/dev/evms/my_vol`.
 2. Click Convert to Compatibility Volume...
 3. Continue converting the volume beginning with step 3 of the GUI instructions.
-

16.3.2. Using Ncurses

Follow these instructions to convert an EVMS volume to a compatibility volume with the Ncurses interface:

1. Choose Actions->Convert->EVMS Volume to Compatibility Volume
2. Select `/dev/evms/my_vol` from the list of available volumes.
3. Activate Convert.

Alternatively, you can perform some of the steps to convert the volume through the context sensitive menu:

1. From the Volumes view, press **Enter** on `/dev/evms/my_vol`.
 2. Activate the Convert to Compatibility Volume menu item.
 3. Continue to convert the volume beginning with step 3 of the Ncurses instructions.
-

16.3.3. Using the CLI

To convert a volume use the **Convert** command. The **Convert** command takes the name of a volume as its first argument, and the keyword `compatibility` to indicate a change to a compatibility volume as the second argument. To complete the example and convert a volume, type the following command at the EVMS : prompt:

```
convert: /dev/evms/my_vol, compatibility
```

Chapter 17. Expanding and shrinking volumes

This chapter tells how to expand and shrink EVMS volumes with the EVMS GUI, Nurses, and CLI interfaces. Note that you can also expand and shrink compatibility volumes and EVMS objects.

17.1. Why expand and shrink volumes?

Expanding and shrinking volumes are common volume operations on most systems. For example, it might be necessary to shrink a particular volume to create free space for another volume to expand into or to create a new volume.

EVMS simplifies the process for expanding and shrinking volumes, and protects the integrity of your data, by coordinating expand and shrink operations with the volume's file system. For example, when shrinking a volume, EVMS first shrinks the underlying file system appropriately to protect the data. When expanding a volume, EVMS expands the file system automatically when new space becomes available.

Not all file system interface modules (FSIM) types supported by EVMS allow shrink and expand operations, and some only perform the operations when the file system is mounted ("online"). The following table details the shrink and expand options available for each type of FSIM.

Table 17–1. FSIM support for expand and shrink operations

FSIM type	Shrinks	Expands
JFS	No	Online only
XFS	No	Online only
ReiserFS	Offline only	Offline and online
ext2/3	Offline only	Offline only
SWAPFS	Offline only	Offline only

You can perform all of the supported shrink and expand operations with each of the EVMS user interfaces.

17.2. Example: shrink a volume

This section tells how to shrink a compatibility volume by 500 MB.

Example 17–1. Shrink a volume

Shrink the volume `/dev/evms/lvm/Sample Container/Sample Region`, which is the compatibility volume that was created in the chapter entitled "Creating Volumes," by 500 MB.

17.2.1. Using the EVMS GUI

Follow these steps to shrink the volume with the EVMS GUI:

1. Select Actions->Shrink->Volume...
2. Select /dev/evms/lvm/Sample Container/Sample Region from the list of volumes.
3. Click Next.
4. Select /lvm/Sample Container/Sample Region from the list of volumes.
5. Click Next.
6. Enter **500MB** in the "Shrink by Size" field.
7. Click Shrink.

Alternatively, you can perform some of the steps to shrink the volume with the GUI context sensitive menu:

1. From the Volumes tab, right click /dev/evms/lvm/Sample Container/Sample Region
2. Click Shrink...
3. Continue the operation beginning with step 3 of the GUI instructions.

17.2.2. Using Ncurses

Follow these steps to shrink a volume with Ncurses:

1. Select Actions->Shrink->Volume.
2. Select /dev/evms/lvm/Sample Container/Sample Region from the list of volumes.
3. Activate Next.
4. Select lvm/Sample Container/Sample Region from the shrink point selection list.
5. Activate Next.
6. Scroll down using the **down** arrow until Shrink by Size is highlighted.
7. Press **spacebar**.
8. Press **Enter**.
9. At the "::<" prompt enter **500MB**.
10. Press **Enter**.
11. Activate Shrink.

Alternatively, you can perform some of the steps to shrink the volume with the context sensitive menu:

1. From the Volumes view, press **Enter** on /dev/evms/lvm/Sample Container/Sample Region.
2. Activate the Shrink menu item.
3. Continue the operation beginning with step 3 of the Ncurses instructions.

17.2.3. Using the CLI

The **shrink** command takes a shrink point followed by an optional name value pair or an optional shrink object. To find the shrink point, use the **query** command with the shrink points filter on the object or volume you plan to shrink. For example:

```
query: shrink points, "/dev/evms/lvm/Sample Container/Sample Region"
```

Use a list options filter on the object of the shrink point to determine the name-value pair to use, as follows:


```
query: objects, object="lvm/Sample Container/Sample Region", list options
```

With the option information that is returned, you can construct the command, as follows:

```
shrink: "lvm/Sample Container/Sample Region", remove_size=500MB
```

17.3. Example: expand a volume

This section tells how to expand a volume a compatibility volume by 500 MB.

Example 17–2. Expand a volume

Expand the volume `/dev/evms/lvm/Sample Container/Sample Region`, which is the compatibility volume that was created in the chapter entitled "Creating Volumes," by 500 MB.

17.3.1. Using the EVMS GUI

Follow these steps to expand the volume with the EVMS GUI:

1. Select Actions→Expand→Volume...
2. Select `/dev/evms/lvm/Sample Container/Sample Region` from the list of volumes.
3. Click Next.
4. Select `lvm/Sample Container/Sample Region` from the list as the expand point.
5. Click Next.
6. Enter **500MB** in the "Additional Size" field.
7. Click Expand.

Alternatively, you can perform some of the steps to expand the volume with the GUI context sensitive menu:

1. From the Volumes tab, right click `/dev/evms/lvm/Sample Container/Sample Region`.
2. Click Expand...
3. Continue the operation to expand the volume beginning with step 3 of the GUI instructions.

17.3.2. Using Ncurses

Follow these steps to expand a volume with Ncurses:

1. Select Actions→Expand→Volume.
2. Select `/dev/evms/lvm/Sample Container/Sample Region` from the list of volumes.
3. Activate Next.
4. Select `lvm/Sample Container/Sample Region` from the list of expand points.
5. Activate Next.
6. Press **spacebar** on the Additional Size field.
7. At the "::<" prompt enter **500MB**.
8. Press **Enter**.
9. Activate Expand.

Alternatively, you can perform some of the steps to shrink the volume with the context sensitive menu:

1. From the Volumes view, press **Enter** on /dev/evms/lvm/Sample Container/Sample Region.
2. Activate the Expand menu item.
3. Continue the operation beginning with step 3 of the Ncurses instructions.

17.3.3. Using the CLI

The **expand** command takes an expand point followed by an optional name value pair and an expandable object. To find the expand point, use the **query** command with the Expand Points filter on the object or volume you plan to expand. For example:

```
query: expand points, "/dev/evms/lvm/Sample Container/Sample Region"
```

Use a list options filter on the object of the expand point to determine the name–value pair to use, as follows:

```
query: objects, object="lvm/Sample Container/Sample Region", list options
```

The free space in your container is the container name plus /Freespace.

With the option information that is returned, you can construct the command, as follows:

```
expand: "lvm/Sample Container/Sample Region", add_size=500MB,  
"lvm/Sample Container/Freespace"
```

Chapter 18. Adding features to an existing volume

This chapter tells how to add additional EVMS features to an already existing EVMS volume.

18.1. Why add features to a volume?

EVMS lets you add features such as drive linking or bad block relocation to a volume that already exists. By adding features, you avoid having to potentially destroy the volume and recreate it from scratch. For example, take the scenario of a volume that contains important data but is almost full. If you wanted to add more data to that volume but no free space existed on the disk immediately after the segment, you could add a drive link to the volume. The drive link concatenates another object to the end of the volume and continues seamlessly.

18.2. Example: add drive linking to an existing volume

The following example shows how to add drive linking to a volume with the EVMS GUI, Ncurses, and CLI interfaces.

Example 18–1. Add drive linking to an existing volume

The following sections show how to add a drive link to volume `/dev/evms/vol` and call the drive link "DL."



NOTE

Drive linking can be done only on EVMS volumes; therefore, `/dev/evms/vol` must be converted to an EVMS volume if it is not already.

18.2.1. Using the EVMS GUI

Follow these steps to add a drive link to the volume with the EVMS GUI:

1. Select Actions→Add→Feature to Volume.
2. Select `/dev/evms/vol`
3. Click Next.
4. Select Drive Linking Feature.
5. Click Next.
6. Type **DL** in the Name Field.
7. Click Add.

Alternatively, you can perform some of the steps to add a drive link with the GUI context sensitive menu:

1. From the Volumes tab, right click `/dev/evms/vol`.
 2. Click Add feature...
 3. Continue adding the drive link beginning with step 3 of the GUI instructions.
-

18.2.2. Using Ncurses

Follow these steps to add a drive link to a volume with Ncurses:

1. Select Actions->Add->Feature to Volume.
2. Select /dev/evms/vol.
3. Activate Next.
4. Select Drive Linking Feature.
5. Activate Next.
6. Press **Spacebar** to edit the Name field.
7. At the "::-" prompt enter **DL**.
8. Press **Enter**.
9. Activate Add.

Alternatively, you can perform some of the steps to add a drive link with the context sensitive menu:

1. From the Volumes view, press **Enter** on /dev/evms/vol.
2. Activate the Add feature menu item.
3. Continue adding the drive link beginning with step 3 of the Ncurses instructions.

18.2.3. Using the CLI

Use the **add feature** to add a feature to an existing volume. Specify the command name followed by a colon, followed by any options and the volume to operate on. To determine the options for a given feature, use the following query:

```
query: plugins, plugin=DriveLink, list options
```

The option names and descriptions are listed to help you construct your command. For our example, the command would look like the following:

```
add feature: DriveLink={ Name="DL }, /dev/evms/vol
```

Chapter 19. Plug-in operations tasks

This chapter discusses plug-in operations tasks and shows how to complete a plug-in task with the EVMS GUI, Ncurses, and CLI interfaces.

19.1. What are plug-in tasks?

Plug-in tasks are functions that are available only within the context of a particular plug-in. These functions are not common to all plug-ins. For example, tasks to add spare disks to a RAID array make sense only in the context of the MD plug-in, and tasks to reset a snapshot make sense only in the context of the Snapshot plug-in.

19.2. Example: complete a plug-in operations task

This section shows how to complete a plug-in operations task with the EVMS GUI, Ncurses, and CLI interfaces.

Example 19–1. Add a spare disk to a compatibility volume made from an MD RAID5 region

This example adds disk `sde` as a spare disk onto volume `/dev/evms/md/md0`, which is a compatibility volume that was created from an MD RAID5 region.

19.2.1. Using the EVMS GUI

Follow these steps to add `sde` to `/dev/evms/md/md0` with the EVMS GUI:

1. Select Other->Storage Object Tasks...
2. Select md/md0.
3. Click Next.
4. Select Add spare object.
5. Click Next.
6. Select sde.
7. Click Add.
8. The operation is completed when you save.

Alternatively, you could use context-sensitive menus to complete the task, as follows:

1. View the region md/md0. You can view the region either by clicking on the small plus sign beside the volume name (`/dev/evms/md/md0`) on the volumes tab, or by selecting the regions tab.
 2. Right click the region (md/md0). A list of acceptable Actions and Navigational shortcuts displays. The last items on the list are the tasks that are acceptable at this time.
 3. Point to Add spare object and left click.
 4. Select sde.
 5. Click Add.
-

19.2.2. Using Ncurses

Follow these steps to add `sde` to `/dev/evms/md/md0` with Ncurses:

1. Select Other->Storage Object Tasks
2. Select `md/md0`.
3. Activate Next.
4. Select Add spare object.
5. Activate Next.
6. Select `sde`.
7. Activate Add.

Alternatively, you can use the context sensitive menu to complete the task:

1. From the Regions view, press **Enter** on `md/md0`.
2. Activate the Add spare object menu item.
3. Select `sde`.
4. Activate Add.

19.2.3. Using the CLI

With the EVMS CLI, all plug-in tasks must be accomplished with the **task** command. Follow these steps to add `sde` to `/dev/evms/md/md0` with the CLI:

1. The following query command with the list options filter to determines the acceptable tasks for a particular object and the name-value pairs it supports. The command returns information about which plug-in tasks are available at the current time and provides the information necessary for you to complete the command.

```
query: objects, object=md/md0, list options
```

2. The command takes the name of the task (returned from the previous query), the object to operate on (in this case, `md/md0`), any required options (none in this case) and, if necessary, another object to be manipulated (in our example, `sde`, which is the spare disk we want to add):

```
task: addspare, md/md0, sde
```

The command is completed upon saving.

Chapter 20. Destroying EVMS objects

This chapter tells how to destroy EVMS objects through the delete and delete recursive operations.

20.1. How to delete objects: delete and delete recursive

There are two ways in EVMS that you can destroy objects that you no longer want: Delete and Delete Recursive. The Delete option destroys only the specific object you specify. The Delete Recursive option destroys the object you specify and its underlying objects, down to the container, if one exists, or else down to the disk. In order for an object to be deleted, it must not be mounted. EVMS verifies that the object you are attempting to delete is not mounted and does not perform the deletion if the object is mounted.

20.2. Example: perform a delete recursive operation

The following example shows how to destroy a volume and the objects below it with the EVMS GUI, Ncurses, and CLI interfaces.

Example 20–1. Destroy a volume and the region and container below it

This example uses the delete recursive operation to destroy volume `/dev/evms/Sample Volume` and the region and container below it. Volume `/dev/evms/Sample Volume` is the volume that was created in earlier. Although we could also use the delete option on each of the objects, the delete recursive option takes fewer steps. Note that because we intend to delete the container as well as the volume, the operation needs to be performed in two steps: one to delete the volume and its contents, and one to delete the container and its contents.

20.2.1. Using the EVMS GUI

Follow these steps to delete the volume and the container with the EVMS GUI:

1. Select Actions->Delete->Volume.
2. Select volume `/dev/evms/Sample Volume` from the list.
3. Click Recursive Delete. This step deletes the volume and the region `lvm/Sample Container/Sample Region`. If you want to keep the underlying pieces or want to delete each piece separately, you would click Delete instead of Delete Recursive.
4. Assuming you chose Delete Recursive (if not, delete the region before continuing with these steps), select Actions->Delete->Container.
5. Select container `lvm/Sample Container` from the list.
6. Click Recursive Delete to destroy the container and anything under it. Alternatively, click Delete to destroy only the container (if you built the container on disks as in the example, either command has the same effect).

Alternatively, you can perform some of the volume deletion steps with the GUI context sensitive menu:

1. From the Volumes tab, right click `/dev/evms/Sample Volume`.
2. Click Delete...

3. Continue with the operation beginning with step 3 of the GUI instructions.

20.2.2. Using Ncurses

Follow these steps to delete the volume and the container with Ncurses:

1. Select Actions->Delete->Volume.
2. Select volume /dev/evms/Sample Volume from the list.
3. Activate Delete Volume Recursively. This step deletes the volume and the region lvm/Sample Container/Sample Region. If you want to keep the underlying pieces or want to delete each piece separately, activate Delete instead of Delete Recursive.
4. Assuming you chose Delete Volume Recursively (if not, delete the region before continuing with these steps), select Actions->Delete->Container.
5. Select container lvm/Sample Container from the list.
6. Click Recursive Delete to destroy the container and everything under it. Alternatively, activate Delete to delete only the container (if you built the container on disks as in the example, either command has the same effect).
7. Press **Enter**.

Alternatively, you can perform some of the volume deletion steps with the context sensitive menu:

1. From the Volumes view, press **Enter** on /dev/evms/Sample Volume.
2. Activate Delete.
3. Continue with the operation beginning with step 3 of the Ncurses instructions.

20.2.3. Using the CLI

Use the **delete** and **delete recursive** commands to destroy EVMS objects. Specify the command name followed by a colon, and then specify the volume, object, or container name. For example:

1. Enter this command to perform the delete recursive operation:

```
delete recursive: "/dev/evms/Sample Volume"
```

This step deletes the volume and the region /lvm/Sample Container/Sample Region. If you wanted to keep the underlying pieces or wanted to delete each piece separately, use the **delete** command, as follows:

```
delete: "/dev/evms/Sample Volume"
```

2. Assuming you chose Delete Volume Recursively (if not, delete the region before continuing with these steps) enter the following to destroy the container and everything under it:

```
delete recursive: "lvm/Sample Container"
```

To destroy only the container, enter the following:

```
delete: "lvm/Sample Container"
```


Appendix A. Building an init–ramdisk to use with EVMS

EVMS versions 1.9.0 and later perform volume discovery in user space and communicate with kernel drivers to activate the volumes. This process presents a problem with having the root file system on an EVMS volume. In order for the root file system volume to be activated, the EVMS tools must be running. However, in order to access the EVMS tools, the root file system must be mounted.

The solution to this dilemma is to use an initial ramdisk (initrd). An initrd is a ram–based device that acts as a temporary root file system at boot time and provides the ability to run programs and load modules that are necessary to activate the true root file system. Thus, in order to have your root file system on an EVMS volume, you need to create and use an initrd.

The following sections provide instructions for creating a new initrd image for use with EVMS.

A.1. Build and install EVMS

Follow the normal EVMS installation instructions for configuring your kernel and building the EVMS tools. See [Chapter 3](#) or the INSTALL file in the EVMS package for more information.

A.2. Kernel support for initrd

Before you can start creating the initrd, make sure your kernel supports ramdisks. You can verify that your kernel supports ramdisks when the kernel is being configured; if it does not support ramdisks, you change the settings so that the kernel does support ramdisks.

Start the kernel configuration:

```
cd /usr/src/linux
make xconfig
```

On the Main Menu screen, under the section for Block Devices, check to see if "RAM disk support" and "Initial RAM disk (initrd) support" are supported (a "Y" should be in the field before each entry). If they are not supported, enter a "Y" for "RAM disk support" and "Initial RAM disk (initrd) support."

```
Main Menu
-Block Devices
<Y>RAM disk support
<Y>Initial RAM disk (initrd) support
```

This support must be built statically into the kernel. Building RAM disk support as a module will not work. The "Default RAM disk size" option is not important at this time, because it can be modified with a command–line option to the kernel.

Save your kernel configuration and rebuild and reinstall your kernel.

A.3. Build the initrd image

The next step is to build the actual ramdisk image, which is described in the following subsections. The important thing to remember is that any program that needs to run from the initrd needs to be copied to the initrd. In addition, any shared libraries that are needed by programs that run from the initrd need to be copied to the initrd as well.

A.3.1. Create a new, blank initrd

Start by creating a new initrd image with an ext2 file system. The following example creates the initrd image at `/boot/initrd-evms`. You can choose to use a different file name if you wish.

The size of the initrd in the following example is 16 MB. You can make the initrd larger or smaller by adjusting the "count" value. The minimum size needed for all the required EVMS tools and supporting libraries is about 11 MB. If you are installing kernel modules to your initrd (see step 3–H below) or other non-EVMS programs, the size might need to be increased.

```
dd if=/dev/zero of=/boot/initrd-evms bs=1M count=16
mke2fs -F -m 0 -b 1024 /boot/initrd-evms
```

A.3.2. Mount the initrd

In order to copy all the required files to the initrd, the initrd must be mounted through a loopback device. To mount the initrd through a loopback device requires that you have loopback support compiled in your kernel (or as a kernel module). See the "Block Devices" menu in the kernel configuration for more information about loopback.

```
mkdir /mnt/initrd
mount -t ext2 -o loop /boot/initrd-evms /mnt/initrd
```

A.3.3. Set up the basic directory structure

Use the following commands to create several basic directories that are required on the initrd:

```
cd /mnt/initrd
mkdir bin dev etc lib proc sbin var
cd var
mkdir lock log
```

A.3.4. Copy helpful utilities

The script that runs in the initrd requires a few common command-line utilities, which you can create with the following commands:

```
cd /bin
cp -a bash cat echo expr grep mount sh umount /mnt/initrd/bin
cd /etc
cp fstab /mnt/initrd/etc
```

A.3.5. Copy supporting libraries

The utilities from the previous step, along with the EVMS tools, require a few common shared libraries. You can create these shared libraries with the following commands:

```
cd /lib
cp -a ld-* /mnt/initrd/lib
cp -a libc-* libc.* /mnt/initrd/lib
cp -a libdl-* libdl.* /mnt/initrd/lib
cp -a libpthread* /mnt/initrd/lib
cp -a libtermcap* /mnt/initrd/lib
```

It is possible that some of the utilities (bash in particular) require additional libraries. Use the **ldd** command to determine if you need additional libraries copied to your initrd. For example, output from the **ldd /bin/bash** command provides a list similar to the following:

```
libtermcap.so.2 => /lib/libtermcap.so.2 (0x40020000)
libdl.so.2 => /lib/libdl.so.2 (0x40024000)
libc.so.6 => /lib/libc.so.6 (0x40027000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

All libraries listed by **ldd** need to be copied to the `/lib` directory on the initrd.

A.3.6. Copy the EVMS tools

Several EVMS libraries and a couple of executables need to be copied to the initrd. First, you need to locate where the EVMS libraries were installed. By default, these libraries are installed in `/lib`. If you specified a different prefix or `libdir` when you configured EVMS, the libraries might be located in a different directory. It is important that these libraries be installed in the same location on the initrd as they are on your real system. For example, if the libraries installed in `/lib`, the libraries need to be copied to `/lib` on the initrd; If the libraries are installed in `/usr/lib`, they need to be copied to `/usr/lib` on the initrd.

The following example assumes the libraries are installed in `/lib`. Only the shared libraries (`.so`) need to be copied. The static versions (`.a`) are not needed on the initrd.

```
cd /lib
cp -a libevms*so* /mnt/initrd/lib
cp -a libdlist*so* /mnt/initrd/lib
```

Next, copy the plug-in libraries to the initrd. The plug-ins are always installed in the `evms` subdirectory of the directory where `libevms` is installed.

Not all of the plug-ins need to be copied to the initrd. Several plug-ins are only for interfacing with the file system utilities and are not necessary at boot time. Other plug-ins are only for interfacing with clustering packages, which cannot be started until the regular boot process.

The following is a list of the specific plug-ins that do not need to be installed:

- csm
- ext2
- ha
- jfs

- reiser
- replace
- rsct
- swap
- xfs

Create and change directory to `/lib/evms`:

```
mkdir /mnt/initrd/lib/evms
cd /lib/evms
```

Copy the contents of the `/lib/evms` directory, minus the plug-ins listed earlier that do not need to be installed, to `/mnt/initrd/lib/evms`:

```
for foo in aix bbr bsd disk dos drivelinek gpt lvm md os2 s390 snapshot sparse
do
    cp -a *$foo* /mnt/initrd/lib/evms
done
```

Next, copy the activation program to the `initrd`. The full user interfaces are not needed, because the only thing the `initrd` does is activate the volumes. Unlike the EVMS libraries, the exact location of this program in the `initrd` is not important, so it can simply go in `sbin`:

```
cd /sbin
cp evms_activate /mnt/initrd/sbin
cp get_dev_num /mnt/initrd/sbin
```

Finally, if you have an `/etc/evms.conf` file installed, you should copy it to the `initrd` so that EVMS uses the correct options during activation. (However, if you have an `/etc/evms.conf` file but have never modified it for your system, it should still have all the default values and does not necessarily need to be installed on the `initrd`.)

```
cd /etc
cp evms.conf /mnt/initrd/etc
```

A.3.7. Set up disk devices

The `initrd` needs to be set up to reflect the disk devices that are on your system. EVMS needs to find the disks in order to activate the volumes.

Before setting up the disk devices on the `initrd`, determine if you are using `devfs`. If you are not sure, you can quickly check for the file `/dev/.devfsd`. If `/dev/.devfsd` exists, you are running `devfs`. You can also check your kernel configuration in the "Filesystems" menu. If "/dev file system support" and "Automatically mount at boot" are enabled, you are running `devfs`.

A.3.7.1. devfs users

Because `devfs` runs automatically within the `initrd`, you do not need to manually copy the device files to the `initrd`. However, `devfs` does need to be mounted within the `initrd` for it to work properly. There are two ways to accomplish this:

- In the kernel configuration, in the "Filesystems" menu, set the "Automatically mount at boot time" option. With this option set, `devfs` will be automatically mounted on `/dev` when the `initrd` is loaded.
- Manually mount `devfs` from the `linuxrc` script before running `evms_activate`. See [Section A.3.9](#) for more details.

A.3.7.2. devfsd users

EVMS does not require `devfs` users to run `devfsd`. However, if you do run `devfsd`, you also need to run it on the `initrd` to ensure that all disks and segments are discovered with the same names on both the `initrd` and the real system. Thus, if you run `devfsd`, you need to copy the `devfsd` program and `config` file to the `initrd`, as follows:

```
cd /sbin
cp devfsd /mnt/initrd/sbin
cd /etc
cp devfsd.conf /mnt/initrd/etc
```

Next, examine the `devfsd.conf` file (the one you just copied to the ramdisk) with a text editor. First look for a line like this:

```
LOOKUP.* MODLOAD
```

Also in the `devfsd` file, look for a line that begins with `RESTORE`. This line specifies a directory where `devfsd` stores changes to the `/dev` file system. Create this directory in your `initrd`. For example, if your `devfsd.conf` file contains the line "`RESTORE /dev-state`," issue the following commands to prevent error messages from being generated when `devfsd` starts within the `initrd`:

```
cd /mnt/initrd
mkdir dev-state
```

A.3.7.3. Non-devfs users

Because `devfs` is not running and mounted within the `initrd`, you need to manually copy the necessary device node files to the `initrd`. If you only have IDE or SCSI disks, the following commands should be sufficient. If you specifically know which disks are on your system, you can copy only those device files.

```
cd /dev
cp -a hd[a-z] /mnt/initrd/dev
```

In addition to the disk devices, you also need a console device:

```
cp -a console /mnt/initrd/dev
```

A.3.8. Copy kernel modules

If you have any kernel modules that need to be loaded in order for EVMS to run, those modules need to be copied to the `initrd`. In particular, if you build your IDE or SCSI drivers, the Device Mapper or MD/Software-RAID drivers, or any required file systems as modules, they need to be present on the `initrd` so they can be loaded before you run EVMS and try to mount the root file system.

If you build all of the necessary drivers and file systems statically into the kernel, you can skip this step. Skipping this step is the recommended approach so that you avoid any possible problems that might be caused by required modules missing from the `initrd`.

When copying the kernel modules, it is probably safest to copy the entire module directory so as not to miss any modules that might be needed on the `initrd`:

```
mkdir /mnt/initrd/lib/modules
cd /lib/modules
cp -a x.y.z /mnt/initrd/lib/modules
```

`x.y.z` is the version of the kernel that will be running EVMS and the `initrd`.

In addition, you will need the module-loading utilities, and probably the module configuration file:

```
cd /sbin
cp modprobe /mnt/initrd/sbin
cd /etc
cp modules.conf /mnt/initrd/etc
```

A.3.9. Write the `linuxrc` script

At this point, all of the necessary files, programs, and libraries should be on the `initrd`. The only thing remaining is the `linuxrc` script. When the kernel mounts the `initrd`, it tries to run a script called `linuxrc`, in the root of the `initrd`. This script performs all the actions necessary for the `initrd`, and prepares the root device so that it can be mounted when the `initrd` exits.

A sample `linuxrc` script is provided in the `doc` directory of the EVMS source package. You can use this script as a starting point.

Copy the **`linuxrc`** sample to your `initrd`:

```
cd /usr/src/evms-2.0.0/doc
cp linuxrc /mnt/initrd
```

Open the `linuxrc` sample script in your favorite text editor. The following paragraphs provide a brief explanation of what the **`linuxrc`** does at boot time and offer suggestions for modifying the script for your system.

- The first section tries to mount `devfs`. You only need to uncomment this section if you are running `devfs` and do not automatically mount `devfs` on `/dev` (see [Section A.3.7](#) for more details).
- The next section tries to start the `devfs` daemon. If `devfs` is not running or `devfsd` is not present, this section is skipped.
- The next section mounts the `proc` file system. EVMS looks in the `/proc` file system to find the location of the Device Mapper driver. Also, later parts of the `linuxrc` script try to access `/proc` in order to properly set the root file system device.
- The next section loads the kernel modules. If you did not copy any kernel modules to your `initrd` ([Section A.3.8](#)), you can leave this section commented out. If you need to load kernel modules from the `initrd`, this is the place to do it. Use the **`modprobe`** command for each module that needs to be loaded. A few examples have been provided within the section.
- The next section is where EVMS runs and activates all of the volumes.

- The next section examines the kernel command line for a parameter that specifies the root volume. More information about how to set up this parameter is included in [Section A.4](#). Device Mapper dynamically allocates all device numbers, which means it is possible that the root volume specified to LILO or GRUB might have a different number when the `initrd` runs than when the system was last running. In order to make sure the correct volume is mounted as root, the `linuxrc` script must determine what the desired root volume name is, determine the number for that device, and set that value in the appropriate file in `/proc`.
- Finally, the `/proc` file system can be unmounted. Also, `devfs` can be unmounted (but only if it was mounted at the start of the script).

When the `linuxrc` script completes, the kernel automatically tries to mount the root file system, and the `initrd` is removed from memory.

A.3.10. Unmount the `initrd` image

The contents of the `initrd` should now be complete and you can unmount it.

A.3.11. Compress the `initrd` image

To conserve space, compress the `initrd` image:

```
gzip /boot/initrd-evms
```

A.4. Set up the boot loader

In order to actually use the `initrd` at boot time, the boot-loader must know the location of the `initrd` so it can tell the kernel where to load it from. There are also some other changes that the boot loader needs to know about in order to successfully mount your EVMS volume as the root file system. The procedure is slightly different for LILO and GRUB, the two main boot loaders used with Linux.

A.4.1. LILO procedure

LILO uses the file `/etc/lilo.conf` as its configuration file. Edit the file with a text editor. If you have not already done so, add an image section for the kernel you will be using with EVMS. The section should look something like this:

```
image = /boot/vmlinuz-2.4.20 # Replace with your kernel image
label = 2.4.20 #Any label you'd like to use
read-only
initrd = /boot/initrd-evms.gz # The compressed initrd you just created
append = "ramdisk=16384 root=/dev/evms/Root"
```

The last line (beginning with "append") in this section is very important. The line specifies parameters that are passed to the kernel command line. The "ramdisk" option overrides the default ramdisk size. This value is in kilobytes and needs to be at least as big as the `initrd` image you created in step [Section A.3.1](#). Thus, if your `initrd` image is 20 MB, you need to set this value to $20 * 1024 = 20480$.

The "root" option in the "append" line is not only a parameter to the kernel but also an indicator to the `linuxrc` script ([Section A.3.9](#)) so it can determine the name of your root file system and use it to tell the

kernel the actual root device after the volumes have been activated. Obviously, you should set this option to the actual name of your root volume.

After updating `/etc/lilo.conf`, run **lilo** to install the boot-loader.

A.4.2. GRUB procedure

GRUB uses the file `/boot/grub/menu.lst` as its configuration file. Edit this file with a text editor. If you have not already, add a menu item for the kernel you will be using with EVMS. The menu item should look something like this:

```
title 2.4.20      # Any label you'd like to use
  kernel (hd0,0)/vmlinuz-2.4.20 root=/dev/evms/Root ramdisk=16384
                # Replace with the name of your kernel image.
                # See the Grub documentation for which (hdx,y)
                # value to use.
  initrd (hd0,0)/initrd-evms.gz # The compressed initrd image you just created
```

The extra information after the kernel image name is very important. These are parameters that are passed to the kernel command line. The "ramdisk" option overrides the default ramdisk size. This value is in kilobytes and needs to be at least as big as the initrd image you created in [Section A.3.1](#). Thus, if your initrd image is 20 MB, you need to set this value to $20 * 1024 = 20480$.

The "root" option in the "kernel" line is not only a parameter to the kernel but also an indicator to the `linuxrc` script ([Section A.3.9](#)) so it can determine the name of your root file system and use it to tell the kernel the actual root device after the volumes have been activated. Obviously, you should set this option to the actual name of your root volume.

A.5. Update the file system configuration

Because the goal of creating the initrd is to use an EVMS volume as your root file system, you also need to update the `fstab` file. Edit `/etc/fstab` with a text editor. There should be an entry in the file similar to the following:

```
/dev/evms/RootVolume  /  ext2  defaults  1  1
```

Replace `RootVolume` with the actual name of your root volume, and `ext2` with the appropriate type for the root file system.

A.6. Reboot the system

The kernel has been built with the appropriate support, the initrd image has been constructed, and the boot-loader has been configured. You are now ready to reboot your system using your EVMS volume as the root file system.

In general, you should still run **evms_activate** during your regular boot scripts. Even though the volumes will already be activated, running `evms_activate` makes sure the device files in `/dev/evms` correctly reflect the device numbers assigned by Device Mapper.

Appendix B. The DOS link plug-in

The DOS plug-in is the most commonly used EVMS segment manager plug-in. The DOS plug-in supports DOS disk partitioning as well as:

- OS/2 partitions that require extra metadata sectors.
- Embedded partition tables: SolarisX86, BSD, and UnixWare.

The DOS plug-in reads metadata and constructs segment storage objects that provide mappings to disk partitions.

B.1. How the DOS plug-in is implemented

The DOS plug-in provides compatibility with DOS partition tables. The plug-in produces EVMS segment storage objects that map primary partitions described by the MBR partition table and logical partitions described by EBR partition tables.

DOS partitions have names that are constructed from two pieces of information:

- The device they are found on.
- The partition table entry that provided the information.

Take, for example, partition name `hda1`, which describes a partition that is found on device `hda` in the MBR partition table. DOS partition tables can hold four entries. Partition numbers 1–4 refer to MBR partition records. Therefore, our example is telling us that partition `hda1` is described by the very first partition record entry in the MBR partition table. Logical partitions, however, are different than primary partitions. EBR partition tables are scattered across a disk but are linked together in a chain that is first located using an extended partition record found in the MBR partition table. Each EBR partition table contains a partition record that describes a logical partition on the disk. The name of the logical partition reflects its position in the EBR chain. Because the MBR partition table reserves numerical names 1–4, the very first logical partition is always named 5. The next logical partition, found by following the EBR chain, is called 6, and so forth. So, the partition `hda5` is a logical partition that is described by a partition record in the very first EBR partition table.

While discovering DOS partitions, the DOS plug-in also looks for OS/2 DLAT metadata to further determine if the disk is an OS/2 disk. An OS/2 disk has additional metadata and the metadata is validated during recovery. This information is important for the DOS plug-in to know because an OS/2 disk must maintain additional partition information. (This is why the DOS plug-in asks, when being assigned to a disk, if the disk is a Linux disk or an OS/2 disk.) The DOS plug-in needs to know how much information must be kept on the disk and what kind of questions it should ask the user when obtaining the information.

An OS/2 disk can contain compatibility volumes as well as logical volumes. A compatibility volume is a single partition with an assigned drive letter that can be mounted. An OS/2 logical volume is a drive link of 1 or more partitions that have software bad-block relocation at the partition level.

Embedded partitions, like those found on a SolarisX86 disk or a BSD compatibility disk, are found within a primary partition. Therefore, the DOS plug-in inspects primary partitions that it has just discovered to further determine if any embedded partitions exist. Primary partitions that hold embedded partition tables have partition type fields that indicate this. For example, a primary partition of type `0xA9` probably has a BSD

partition table that subdivides the primary partition into BSD partitions. The DOS plug-in looks for a BSD disk label and BSD data partitions in the primary partition. If the DOS plug-in finds a BSD disk label, it exports the BSD partitions. Because this primary partition is actually just a container that holds the BSD partitions, and not a data partition itself, it is not exported by the DOS plug-in. Embedded partitions are named after the primary partition they were discovered within. As an example, `hda3 . 1` is the name of the first embedded partition found within primary partition `hda3`.

B.2. Assigning the DOS plug-in

Assigning a segment manager to a disk means that you want the plug-in to manage partitions on the disk. In order to assign a segment manager to a disk, the plug-in needs to create and maintain the appropriate metadata, which is accomplished through the "disk type" option. When you specify the "disk type" option and choose Linux or OS/2, the plug-in knows what sort of metadata it needs to keep and what sort of questions it should ask when creating partitions.

An additional OS/2 option is the "disk name" option, by which you can provide a name for the disk that will be saved in OS/2 metadata and that will be persistent across reboots.

B.3. Creating DOS partitions

There are two basic DOS partition types:

1. A primary partition, which is described by a partition record in the MBR partition table.
2. A logical partition, which is described by a partition record in the EBR partition table.

Every partition table has room for four partition records; however, there are a few rules that impose limits on this.

An MBR partition table can hold four primary partition records unless you also have logical partitions. In this case, one partition record is used to describe an extended partition and the start of the EBR chain that in turn describes logical partitions.

Because all logical partitions must reside in the extended partition, you cannot allocate room for a primary partition within the extended partition and you cannot allocate room for a logical partition outside or adjacent to this area.

Lastly, an EBR partition table performs two functions:

1. It describes a logical partition and therefore uses a partition record for this purpose.
2. It uses a partition record to locate the next EBR partition table.

EBR partition tables use at most two entries.

When creating a DOS partition, the options you are presented with depend on the kind of disk you are working with. However, both OS/2 disks and Linux disks require that you choose a freespace segment on the disk within which to create the new data segment. The create options are:

size

The size of the partition you are creating. Any adjustments that are needed for alignment are

performed by the DOS plug-in and the resulting size might differ slightly from the value you enter.

offset

Lets you skip sectors and start the new partition within the freespace area by specifying a sector offset.

type

Lets you enter a partition type or choose from a list of partition types; for example, native Linux.

primary

Lets you choose between creating a primary or logical partition. Due to the rules outlined above, you might or might not have a choice. The DOS plug-in can determine if a primary or logical partition can be created in the freespace area you chose and disable this choice.

bootable

Lets you enable the sys_ind flag field in a primary partition and disable it when creating a logical partition. The sys_ind flag field identifies the active primary partition for booting.

Additional OS/2 options are the following:

partition name

An OS/2 partition can have a name, like Fred or Part1.

volume name

OS/2 partitions belong to volumes, either compatibility or logical, and volumes have names.

However, because the DOS plug-in is not a logical volume manager, it cannot actually create OS/2 logical volumes.

drive letter

You can specify the drive letter for an OS/2 partition, but it is not a required field. Valid drive letters are: C,D...Z.

B.4. Expanding DOS partitions

A partition is a physically contiguous run of sectors on a disk. You can expand a partition by adding unallocated sectors to the initial run of sectors on the disk. Because the partition must remain physically contiguous, a partition can only be expanded by growing into an unused area on the disk. These unused areas are exposed by the DOS plug-in as freespace segments. Therefore, a data segment is only expandable if a freespace segment immediately follows it. Lastly, because a DOS partition must end on a cylinder boundary, DOS segments are expanded in cylinder size increments. This means that if the DOS segment you want to expand is followed by a freespace segment, you might be unable to expand the DOS segment if the freespace segment is less than a cylinder in size.

There is one expand option, as follows:

size

This is the amount by which you want to expand the data segment. The amount must be a multiple of the disk's cylinder size.

B.5. Shrinking DOS partitions

A partition is shrunk when sectors are removed from the end of the partition. Because a partition must end on a cylinder boundary, a partition is shrunk by removing cylinder amounts from the end of the segment.

There is one shrink option, as follows:

size

The amount by which you want to reduce the size of the segment. Because a segment ends on a cylinder boundary, this value must be some multiple of the disk's cylinder size.

B.6. Deleting partitions

You can delete an existing DOS data segment as long as it is not currently a compatibility volume, an EVMS volume, or consumed by another EVMS plug-in. No options are available for deleting partitions.

Appendix C. The MD region manager

Multiple disks (MD) support in Linux is a software implementation of RAID (Redundant Array of Independent Disks). The basic idea of software RAID is to combine multiple inexpensive hard disks into an array of disks to obtain performance, capacity, and reliability that exceeds that of a single large disk.

Linux software RAID works on most block devices. A Linux RAID device can be composed of a mixture of IDE or SCSI devices. Furthermore, because a Linux RAID device is itself a block device, it can be a member of another Linux RAID device.

Whereas there are six standard types of RAID arrays (RAID-0 through RAID-5) in the hardware implementation, the Linux implementation of software RAID has RAID-0, RAID-1, RAID-4, and RAID-5 levels. In addition to these four levels, Linux also has support for another non-redundant array called "Linear Mode."

All levels of Linux software RAID are discussed in greater detail in the Software RAID HOWTO of The Linux Documentation Project ([TLDP](#)). One important thing to remember is RAID is not a substitute for backups.

C.1. Creating an MD region

There are four EVMS MD region plug-ins: Linear, RAID-0, RAID-1, and RAID-4/5. The RAID-4/5 region plug-in provides support for both RAID-4 and RAID-5 arrays. After an MD region manager is selected, the software provides a list of acceptable objects. The ordering of the MD array is implied by the order in which you pick objects from the provided list. The following are MD region configuration options:

chunk size

The smallest chunk size is 4 KB and the largest is 4096 KB. The chunk size is a power of 2 of the previous value. Consider the intended use of the MD region when selecting chunk size. For example, if the MD region contains mostly large files, you might see better performance by having a larger chunk size. The block size of the file system being used is also an important factor when selecting chunk size.

This option is available for use with RAID-0 and RAID-4/5.

spare disk

The benefit of having a spare disk is that when an active disk fails, the kernel MD code automatically replaces the failed disk with the spare disk. Otherwise, the MD array operates in a degraded mode.

This option is available for use with RAID-1 and RAID-4/5.

RAID-5 algorithms

There are four RAID-5 parity algorithms: left asymmetric, right asymmetric, left symmetric, and right symmetric. The [ACCS](#) web page provides examples of what the different parity algorithms do.

This option is available for use with the RAID-5 algorithm.

C.2. Adding and removing a spare object (RAID–1 and RAID–4/5)

When adding a spare disk to an existing MD region, select an available object that has the same size as the disks that are currently active in the MD region. If the MD region consists of objects with different sizes, use the smallest size.

Note that after adding a spare to a degraded MD region, the kernel MD code automatically starts the reconstruction of the MD array. When reconstruction finishes, the spare disks becomes an active disk.

If you want to reorganize disks and segments, you can remove an existing spare disk from the MD region. This is a safe operation because the spare disk does not contain any data.

C.3. Removing an active object (RAID–1 only)

Only the MD RAID–1 region plug-in lets you remove an active disk of the MD region. This option is available for those MD RAID–1 regions that have at least two active members.

C.4. Removing a faulty object (RAID–1 and RAID–4/5)

When an I/O error occurs on a disk, the disk is marked faulty by the kernel MD code. Use this function to permanently remove the faulty disk from the MD region.

C.5. Marking an object faulty (RAID–4/5 only)

This option is available when the RAID–4/5 array has at least one spare disk. Use this option to swap an active disk with a spare disk. The active disk is marked faulty and can be later removed, as described in [Section C.4](#).

C.6. Replacing an object

In EVMS 2.0 and later, you can replace a member of an MD region with an available storage object. The new object must be the same size as the replaced object. This option is currently only supported for volumes that are offline.

C.7. Characteristics of Linux RAID levels

The following subsections describe the characteristics of each Linux RAID level.

C.7.1. Linear mode

Characteristics:

- Two or more disks are combined into one virtual MD device.
- The disks are appended to each other, so writing linearly to the MD device fills up disk 0 first, then 1, and so on.

- The disks do not have to be of the same size.

Advantages:

- Can be used to build a very large MD device.
- No parity calculation overhead is involved.

Disadvantages:

- Not a "true" RAID because it is not fault-tolerant.
 - One disk crash will probably result in loss of most or all data.
 - Should never be used in mission-critical environments.
-

C.7.2. RAID-0

Characteristics:

- Two or more disks are combined into one virtual MD device.
- Also called "stripe" mode.
- Stripe size determines how data is written to disk. For example, writing 16 K bytes to a RAID-0 array of three disks with stripe size of 4 K bytes is broken down into:
 - ◆ 4 K bytes of disk 0
 - ◆ 4 K bytes to disk 1
 - ◆ 4 K bytes to disk 2
 - ◆ 4 K bytes to disk 0
- The disks should be the same size but they do not have to be the same size.

Advantages:

- Can be used to build a very large MD device.
- I/O performance is greatly improved by spreading I/O load across many controllers and disks.
- No parity calculation overhead is involved.

Disadvantages:

- Not a "true" RAID because it is not fault-tolerant.
 - One disk crash is liable to result in the loss of the whole array.
 - Should never be used in mission-critical environments.
-

C.7.3. RAID-1

Characteristics:

- Consists of two or more disks to provide a two-way or N-way mirrored MD device.
- Writes result in writing identical data to all active disks in the array.
- Reads can be performed on any active disk of the array.
- Data is intact as long as there is at least one "good" active disk in the array.
- The disks should be the same size. If they are different sizes, the size of the RAID-1 array is determined by the smallest disk.

Advantages:

- 100% redundancy of data.
- Under certain circumstances, a RAID–1 array can sustain multiple simultaneous disk failures.
- Kernel MD code provides good read–balancing algorithm.
- No parity calculation overhead is involved.

Disadvantages:

- Write performance is often worse than on a single device.
-

C.7.4. RAID–4

Characteristics:

- Consists of three or more striped disks.
- Parity information is kept on one disk. When a disk fails, parity information is used to reconstruct all data.
- The disks should be the same size. If they are different sizes, the size of the RAID–4 array is determined by the smallest disk.

Advantages:

- Like RAID–0, I/O performance is greatly improved by spreading the I/O load across many controllers and disks.

Disadvantages:

- The parity disk becomes a bottleneck. Therefore, a slow parity disk degrades I/O performance of the whole array.
 - Cannot sustain a two–disk simultaneous failure.
-

C.7.5. RAID–5

Characteristics:

- Consists of three or more striped disks.
- Parity information is distributed evenly among the participating disks.
- The disks should be the same size. If they are different sizes, the size of the RAID–5 array is determined by the smallest disk.

Advantages:

- Like RAID–0, I/O performance is greatly improved by spreading the I/O load across many controllers and disks.
- Read performance is similar to RAID–0.

Disadvantages:

- Writes can be expensive when required read–in blocks for parity calculations are not in the cache.

- Cannot sustain a two-disk simultaneous failure.
-

Appendix D. The LVM plug-in

The LVM plug-in combines storage objects into groups called containers. From these containers, new storage objects can be created, with a variety of mappings to the consumed objects. Containers allow the storage capacity of several objects to be combined, allow additional storage to be added in the future, and allow for easy resizing of the produced objects.

D.1. How LVM is implemented

The Linux LVM plug-in is compatible with volumes and volume groups from the original Linux LVM tools from Sistina Software. The original LVM is based on the concept of volume groups. A volume group (VG) is a grouping of physical volumes (PVs), which are usually disks or disk partitions. The volume group is not directly usable as storage space; instead, it represents a pool of available storage. You create logical volumes (LVs) to use this storage. The storage space of the LV can map to one or more of the group's PVs.

The Linux LVM concepts are represented by similar concepts in the EVMS LVM plug-in. A volume group is called a container, and the logical volumes that are produced are called regions. The physical volumes can be disks, segments, or other regions. Just as in the original LVM, regions can map to the consumed objects in a variety of ways.

D.2. Container operations

D.2.1. Creating LVM containers

Containers are created with an initial set of objects. In the LVM plug-in, the objects can be disks, segments, or regions. LVM has two options for creating containers. The value of these options cannot be changed after the container has been created. The options are:

name

The name of the new container.

pe_size

The physical extent (PE) size, which is the granularity with which regions can be created. The default is 16 MB. Each region must have a whole number of extents. Also, each region can have only up to 65534 extents. Thus, the PE size for the container limits the maximum size of a region in that container. With the default PE size, an LVM region can be, at most 1 TB. In addition, each object consumed by the container must be big enough to hold at least five extents. Thus, the PE size cannot be arbitrarily large. Choose wisely.

D.2.2. Adding objects to LVM containers

You can add objects to existing LVM containers in order to increase the pool of storage that is available for creating regions. A single container can consume up to 256 objects. Because the name and PE size of the containers are set when the container is created, no options are available when you add new objects to a container. Each object must be large enough to hold five physical extents. If an object is not large enough to satisfy this requirement, the LVM plug-in will not allow the object to be added to the container.

D.2.3. Removing objects from LVM containers

You can remove a consumed object from its container as long as no regions are mapped to that object. The LVM plug-in does not allow objects that are in use to be removed their their container. If an object must be removed, you can delete or shrink regions, or move extents, in order to free the object from use.

No options are available for removing objects from LVM containers.

D.2.4. Deleting LVM containers

You can delete a container as long as the container does not have any produced regions. The LVM plug-in does not allow containers to be deleted if they have any regions. No options are available for deleting LVM containers.

D.3. Region operations

D.3.1. Creating LVM regions

You create LVM regions from the freespace in LVM containers. If there is at least one extent of freespace in the container, you can create a new region.

The following options are available for creating LVM regions:

name

The name of the new region.

extents

The number of extents to allocate to the new region. A new region must have at least one extent and no more than the total available free extents in the container, or 65534 (whichever is smaller). If you use the *extents* option, the appropriate value for the *size* option is automatically calculated. By default, a new region uses all available extents in the container.

size

The size of the new region. This size must be a multiple of the container's PE size. If you use the *size* option, the appropriate value for the *extents* options is automatically calculated. By default, a new region uses all available freespace in the container.

stripes

If the container consumes two or more objects, and each object has unallocated extents, then the new region can be striped across multiple objects. This is similar to RAID-0 striping and achieves an increased amount of I/O throughput across multiple objects. This option specifies how many objects the new region should be striped across. By default, new regions are not striped, and this value is set to 1.

stripe_size

The granularity of striping. The default value is 16 KB. Use this option only if the *stripes* option is greater than 1.

contiguous

This option specifies that the new region must be allocated on a single object, and that the extents on that object must be physically contiguous. By default, this is set to false, which allows regions to span objects. This option cannot be used if the *stripes* option is greater than 1.

pv_names

A list of names of the objects the new region should map to. By default, this list is empty, which means all available objects will be used to allocate space to the new region.

D.3.2. Expanding LVM regions

You can expand an existing LVM region if there are unused extents in the container. If a region is striped, you can expand it only by using free space on the objects it is striped across. If a region was created with the contiguous option, you can only expand it if there is physically contiguous space following the currently allocated space.

The following options are available for expanding LVM regions:

add_extents

The number of extents to add to the region. If you specify this option, the appropriate value for the *add_size* option is automatically calculated. By default, the region will expand to use all free extents in the container.

add_size

The amount of space to add to the region. If you specify this option, the appropriate value for the *add_extents* option is automatically calculated. By default, the region will expand to use all freespace in the container.

pv_names

A list of names of the objects to allocate the additional space from. By default, this list is empty, which means all available objects will be used to allocate new space to the region.

D.3.3. Shrinking LVM regions

You can shrink an existing LVM region by removing extents from the end of the region. Regions must have at least one extent, so regions cannot be shrunk to zero.

The following options are available when shrinking LVM regions. Because regions are always shrunk by removing space from the end of the region, a list of objects cannot be specified in this command.

remove_extents

The number of extents to remove from the region. If you specify this option, the appropriate value for the *remove_size* option is automatically calculated. By default, one extent is removed from the region.

remove_size

The amount of space to shrink the region by. If you specify this option, the appropriate value for the *remove_extents* option is automatically calculated.

D.3.4. Deleting LVM regions

You can delete an existing LVM region as long as it is not currently a compatibility volume, an EVMS volume, or consumed by another EVMS plug-in. No options are available for deleting LVM regions.

Appendix E. The CSM plug-in

The Cluster Segment Manager (CSM) is the EVMS plug-in that identifies and manages cluster storage. The CSM protects disk storage objects by writing metadata at the start and end of the disk, which prevents other plug-ins from attempting to use the disk. Other plug-ins can look at the disk, but they cannot see their own metadata signatures and cannot consume the disk. The protection that CSM provides allows the CSM to discover cluster storage and present it in an appropriate fashion to the system.

All cluster storage disk objects must be placed in containers that have the following attributes:

- cluster ID that identifies the cluster management software
- node ID that identifies the owner of the disk objects
- storage type: private, shared, or deported

The CSM plug-in reads metadata and constructs containers that consume the disk object. Each disk provides a usable area, mapped as an EVMS data segment, but only if the disk is accessible to the node viewing the storage.

The CSM plug-in performs these operations:

- examines disk objects
- creates containers
- uses the containers to consume disk objects
- produces data segment objects if the disk is accessible to the node

E.1. Assigning the CSM plug-in

Assigning a segment manager to a disk means that you want the plug-in to manage partitions on the disk. In order to do this, the plug-in needs to create and maintain appropriate metadata. The CSM creates the following three segments on the disk:

- primary metadata segment
- usable area data segment
- secondary metadata segment

The CSM collects the information it needs to perform the assign operation with the following options:

NodeId

Choose only from a list of configured node IDs that have been provided to the CSM by clustering software. The default selection is the node from which you are running the EVMS user interface.

Container Name

The name for the container. Currently, you need to keep this name unique across the cluster to prevent name-in-conflict errors should the container fail over to another node that has a container with the same name. Later releases of EVMS will assist you with this.

Storage Type

Can be either: share, private, or deported.

Note that you would typically assign the CSM to a disk when you want to add a disk to an existing CSM container. If you are creating a new container, you have a choice of using either:

Actions->Create->Container or Actions->Add->Segment Manager.

If the container doesn't exist, it will be created for it. If the container already exists, the disk will be added to it.

E.2. Unassigning the CSM plug-in

Unassigning a CSM plug-in results in the CSM removing its metadata from the specified disk storage object. The result is that the disk has no segments mapped and appears as a raw disk object. The disk is removed from the container that consumed it and the data segment is removed as well.

E.3. Deleting a CSM container

An existing CSM container cannot be deleted if it is producing any data segments, because other EVMS plug-ins might be building higher-level objects on the CSM objects. To delete a CSM container, first remove disk objects from the container. When the last disk is removed, the container is also removed.

Appendix F. JFS file system interface module

The JFS FSIM lets EVMS users create and manage JFS file systems from within the EVMS interfaces. In order to use the JFS FSIM, version 1.0.9 or later of the JFS utilities must be installed on your system. The latest version of JFS can be found at <http://oss.software.ibm.com/jfs/>.

F.1. Creating JFS file systems

JFS file systems can be created with **mkfs** on any EVMS or compatibility volume (at least 16 MB in size) that does not already have a file system. The following options are available for creating JFS file systems:

badblocks

Perform a read-only check for bad blocks on the volume before creating the file system. The default is false.

caseinsensitive

Mark the file system as case-insensitive (for OS/2 compatibility). The default is false.

vollabel

Specify a volume label for the file system. The default is none.

journalvol

Specify the volume to use for an external journal. This option is only available with version 1.0.20 or later of the JFS utilities. The default is none.

logsize

Specify the inline log size (in MB). This option is only available if the *journalvol* option is not set. The default is 0.4% of the size of the volume up to 32 MB.

F.2. Checking JFS file systems

The following options are available for checking JFS file systems with **fsck**:

force

Force a complete file system check, even if the file system is already marked clean. The default is false.

readonly

Check the file system is in read-only mode. Report but do not fix errors. If the file system is mounted, this option is automatically selected. The default is false.

omitlog

Omit replaying the transaction log. This option should only be specified if the log is corrupt. The default is false.

verbose

Display details and debugging information during the check. The default is false.

version

Display the version of **fsck**, **jfs** and exit without checking the file system. The default is false.

F.3. Removing JFS file systems

A JFS file system can be removed from its volume if the file system is unmounted. This operation involves erasing the superblock from the volume so the file system will not be recognized in the future. There are no options available for removing file systems.

F.4. Expanding JFS file systems

A JFS file system is automatically expanded when its volume is expanded. However, JFS only allows the volume to be expanded if it is mounted, because JFS performs all of its expansions online. In addition, JFS only allows expansions if version 1.0.21 or later of the JFS utilities are installed.

F.5. Shrinking JFS file systems

At this time, JFS does not support shrinking its file systems. Hence, volumes with JFS file systems cannot be shrunk.

Appendix G. XFS file system interface module

The XFS FSIM lets EVMS users create and manage XFS file systems from within the EVMS interfaces. In order to use the XFS FSIM, version 2.0.0 or later of the XFS utilities must be installed on your system. The latest version of XFS can be found at <http://oss.sgi.com/projects/xfs/>.

G.1. Creating XFS file systems

XFS file systems can be created with **mkfs** on any EVMS or compatibility volume that does not already have a file system. The following options are available for creating XFS file systems:

vollabel

Specify a volume label for the file system. The default is none.

journalvol

Specify the volume to use for an external journal. The default is none.

logsize

Specify the inline log size (in MB). This option is only available if the *journalvol* option is not set. The default is 4 MB; the allowed range is 2 to 256 MB.

G.2. Checking XFS file systems

The following options are available for checking XFS file systems with **fsck**:

readonly

Check the file system is in read-only mode. Report but do not fix errors. The default is false.

verbose

Display details and debugging information during the check. The default is false.

G.3. Removing XFS file systems

An XFS file system can be removed from its volume if the file system is unmounted. This operation involves erasing the superblock from the volume so the file system will not be recognized in the future. There are no options available for removing file systems.

G.4. Expanding XFS file systems

An XFS file system is automatically expanded when its volume is expanded. However, XFS only allows the volume to be expanded if it is mounted, because XFS performs all of its expansions online.

G.5. Shrinking XFS file systems

At this time, XFS does not support shrinking its file systems. Hence, volumes with XFS file systems cannot be shrunk.

Appendix H. ReiserFS file system interface module

The ReiserFS FSIM lets EVMS users create and manage ReiserFS file systems from within the EVMS interfaces. In order to use the ReiserFS FSIM, version 3.x.0 or later of the ReiserFS utilities must be installed on your system. In order to get full functionality from the ReiserFS FSIM, use version 3.x.1b or later. The latest version of ReiserFS can be found at <http://www.namesys.com/>.

H.1. Creating ReiserFS file systems

ReiserFS file systems can be created with **mkfs** on any EVMS or compatibility volume that does not already have a file system. The following option is available for creating ReiserFS file systems:

vollabel

Specify a volume label for the file system. The default is none.

H.2. Checking ReiserFS file systems

The following option is available for checking XFS file systems with **fsck**:

mode

There are three possible modes for checking a ReiserFS file system: Check Read-Only, Fix, and Rebuild Tree."

H.3. Removing ReiserFS file systems

A ReiserFS file system can be removed from its volume if the file system is unmounted. This operation involves erasing the superblock from the volume so the file system will not be recognized in the future. There are no options available for removing file systems.

H.4. Expanding ReiserFS file systems

A ReiserFS file system is automatically expanded when its volume is expanded. ReiserFS file systems can be expanded if the volume is mounted or unmounted.

H.5. Shrinking ReiserFS file systems

A ReiserFS file system is automatically shrunk if the volume is shrunk. ReiserFS file systems can only be shrunk if the volume is unmounted.

Appendix I. Ext-2/3 file system interface module

The Ext-2/3 FSIM lets EVMS users create and manage Ext2 and Ext3 file systems from within the EVMS interfaces. In order to use the Ext-2/3 FSIM, the e2fsprogs package must be installed on your system. The e2fsprogs package can be found at <http://e2fsprogs.sourceforge.net/>.

I.1. Creating Ext-2/3 file systems

Ext-2/3 file systems can be created with **mkfs** on any EVMS or compatibility volume that does not already have a file system. The following options are available for creating Ext-2/3 file systems:

badblocks

Perform a read-only check for bad blocks on the volume before creating the file system. The default is false.

badblocks_rw

Perform a read/write check for bad blocks on the volume before creating the file system. The default is false.

vollabel

Specify a volume label for the file system. The default is none.

journal

Create a journal for use with the Ext2 file system. The default is true.

I.2. Checking Ext-2/3 file systems

The following options are available for checking Ext-2/3 file systems with **fsck**:

force

Force a complete file system check, even if the file system is already marked clean. The default is false.

readonly

Check the file system is in read-only mode. Report but do not fix errors. If the file system is mounted, this option is automatically selected. The default is false.

badblocks

Check for bad blocks on the volume and mark them as busy. The default is false.

badblocks_rw

Perform a read-write check for bad blocks on the volume and mark them as busy. The default is false.

I.3. Removing Ext-2/3 file systems

An Ext-2/3 file system can be removed from its volume if the file system is unmounted. This operation involves erasing the superblock from the volume so the file system will not be recognized in the future. There are no options available for removing file systems.

I.4. Expanding and shrinking Ext-2/3 file systems

An Ext-2/3 file system is automatically expanded or shrunk when its volume is expanded or shrunk. However, Ext-2/3 only allows these operations if the volume is unmounted, because online expansion and

shrinkage is not yet supported.