Logical Domains (LDoms) 1.1 Administration Guide

Sun Microsystems, Inc.
www.sun.com

Part No. 820-4913-10
December 2008, Revision A

Submit comments about this document at: `http://www.sun.com/hwdocs/feedback`

Contents

Preface

The *Logical Domains (LDoms) 1.1 Administration Guide* provides detailed information and procedures that describe the overview, security considerations, installation, configuration, modification, and execution of common tasks for the Logical Domains Manager 1.1 software on supported servers, blades, and server modules. Refer to "Supported Platforms" in the *Logical Domains (LDoms) 1.1 Release Notes* for a list. This guide is intended for the system administrators on these servers who have a working knowledge of UNIX® systems and the Solaris™ Operating System (Solaris OS).

Related Documentation

The *Logical Domains (LDoms) 1.1 Administration Guide* and *Release Notes* are available at:

`http://docs.sun.com/app/docs/prod/ldoms#hic`

The *Beginners Guide to LDoms: Understanding and Deploying Logical Domains Software* can be found at the Sun BluePrints™ site at:

`http://www.sun.com/blueprints/0207/820-0832.html`

You can find documents relating to your server, software, or Solaris OS at:

`http://docs.sun.com`

Type the name of the server, software, or Solaris OS in the Search box to find the documents you need.

Application	Title	Part Number	Format
Release notes for LDoms	*Logical Domains (LDoms) 1.1 Release Notes*	820-4914-10	HTML PDF
Solaris man pages for LDoms	Solaris 10 Reference Manual Collection: • `drd`(1M) man page • `vntsd`(1M) man page	N/A	HTML
LDoms man page	`ldm`(1M) man page	N/A	SGML
	Logical Domains (LDoms) 1.1 Manager Man Page Guide	820-4915-10	PDF
Basics for Logical Domains software	*Beginners Guide to LDoms: Understanding and Deploying Logical Domains Software*	820-0832	PDF
Administration for LDoms MIB	*Logical Domains (LDoms) MIB 1.0.1 Administration Guide*	820-2319-10	HTML PDF
Release notes for LDoms MIB	*Logical Domains (LDoms) MIB 1.0.1 Release Notes*	820-2320-10	HTML PDF
Administration for Libvirt for LDoms	*Libvirt for LDoms 1.0.1 Administration Guide*	820-3839-10	HTML PDF
Release Notes for Libvirt for LDoms	*Libvirt for LDoms 1.0.1 Release Notes*	820-3838-10	HTML PDF
Solaris OS including installation, using JumpStart™, and using the SMF	Solaris 10 Collection	N/A	HTML PDF
Security	*Solaris Security Toolkit 4.2 Administration Guide*	819-1402-10	HTML PDF
Security	*Solaris Security Toolkit 4.2 Reference Manual*	819-1503-10	HTML PDF
Security	*Solaris Security Toolkit 4.2 Release Notes*	819-1504-10	HTML PDF
Security	*Solaris Security Toolkit 4.2 Man Page Guide*	819-1505-10	HTML PDF

Documentation, Support, and Training

Sun Function	URL
Documentation	`http://docs.sun.com`
Support	`http://www.sun.com/support`
Training	`http://www.sun.com/training`

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. You can submit your comments by going to:

`http://www.sun.com/hwdocs/feedback`

Please include the title and part number of your document with your feedback:

Logical Domains (LDoms) 1.1 Administration Guide, part number 820-4913-10.

CHAPTER 1

Overview of the Logical Domains Software

This chapter contains a brief overview of the Logical Domains software. All of the Solaris OS functionality necessary to use Sun's Logical Domains technology is in the Solaris 10 11/06 release (at a minimum) with the addition of necessary patches. However, system firmware and the Logical Domains Manager are also required to use logical domains. Refer to "Required and Recommended Software" in the *Logical Domains (LDoms) 1.1 Release Notes* for specific details.

Hypervisor and Logical Domains

This section provides a brief overview of the SPARC® hypervisor and the logical domains it supports.

The SPARC hypervisor is a small firmware layer that provides a stable virtualized machine architecture to which an operating system can be written. Sun servers using the hypervisor provide hardware features to support the hypervisor's control over a logical operating system's activities.

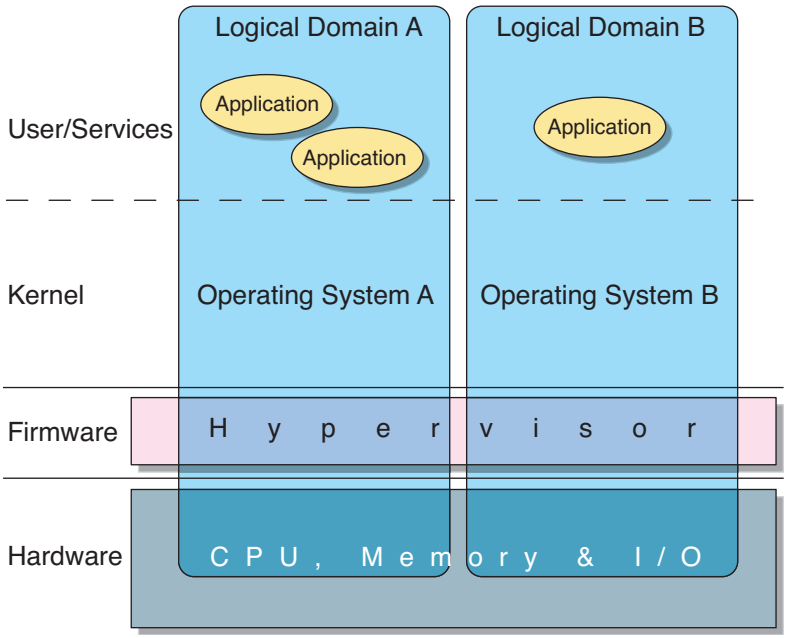
A logical domain is a discrete logical grouping with its own operating system, resources, and identity within a single computer system. Each logical domain can be created, destroyed, reconfigured, and rebooted independently, without requiring a power cycle of the server. You can run a variety of applications software in different logical domains and keep them independent for performance and security purposes.

Each logical domain is allowed to observe and interact with only those server resources made available to it by the hypervisor. Using the Logical Domains Manager, the system administrator specifies what the hypervisor should do through the control domain. Thus, the hypervisor enforces the partitioning of the resources of a server and provides limited subsets to multiple operating system environments.

This is the fundamental mechanism for creating logical domains. The following diagram shows the hypervisor supporting two logical domains. It also shows the layers that make up the Logical Domains functionality:

- Applications, or user/services
- Kernel, or operating systems
- Firmware, or hypervisor
- Hardware, including CPU, memory, and I/O

FIGURE 1-1 Hypervisor Supporting Two Logical Domains



The number and capabilities of each logical domain that a specific SPARC hypervisor supports are server-dependent features. The hypervisor can allocate subsets of the overall CPU, memory, and I/O resources of a server to a given logical domain. This enables support of multiple operating systems simultaneously, each within its own logical domain. Resources can be rearranged between separate logical domains with an arbitrary granularity. For example, memory is assignable to a logical domain with an 8-kilobyte granularity.

Each virtual machine can be managed as an entirely independent machine with its own resources, such as:

- Kernel, patches, and tuning parameters
- User accounts and administrators
- Disks

- Network interfaces, MAC addresses, and IP addresses

Each virtual machine can be stopped, started, and rebooted independently of each other without requiring a power cycle of the server.

The hypervisor software is responsible for maintaining the separation between logical domains. The hypervisor software also provides logical domain channels (LDCs), so that logical domains can communicate with each other. Using logical domain channels, domains can provide services to each other, such as networking or disk services.

The service processor (SP), also known as the system controller (SC), monitors and runs the physical machine, but it does not manage the virtual machines. The Logical Domains Manager runs the virtual machines.

Logical Domains Manager

The Logical Domains Manager is used to create and manage logical domains. There can be only one Logical Domains Manager per server. The Logical Domains Manager maps logical domains to physical resources.

Roles for Logical Domains

All logical domains are the same except for the roles that you specify for them. There are multiple roles that logical domains can perform.

TABLE 1-1 Logical Domain Roles

Domain Role	Description
Control domain	Domain in which the Logical Domains Manager runs allowing you to create and manage other logical domains and allocate virtual resources to other domains. There can be only one control domain per server. The initial domain created when installing Logical Domains software is a control domain and is named `primary`.
Service domain	Domain that provides virtual device services to other domains, such as a virtual switch, a virtual console concentrator, and a virtual disk server.
I/O domain	Domain that has direct ownership of and direct access to physical I/O devices, such as a network card in a PCI Express controller. Shares the devices with other domains in the form of virtual devices when the I/O domain is also the control domain. The number of I/O domains you can have is dependent on your platform architecture. For example, if you are using a Sun UltraSPARC® T1 processor, you can have a maximum of two I/O domains, one of which also must be the control domain.
Guest domain	Domain that is managed by the control domain and uses services from the I/O and service domains.

If you have an existing system and already have an operating system and other software running on your server, that will be your control domain once you install the Logical Domains Manager. You might want to remove some of your applications from the control domain once it is set up, and balance the load of your applications throughout your domains to make the most efficient use of your system.

Command-Line Interface

The Logical Domains Manager provides a command-line interface (CLI) for the system administrator to create and configure logical domains. The CLI is a single command, `ldm`(1M), with multiple subcommands.

To use the Logical Domains Manager CLI, you must have the Logical Domains Manager daemon, `ldmd`, running. The `ldm`(1M) command and its subcommands are described in detail in the `ldm`(1M) man page and the *Logical Domains (LDoms) Manager Man Page Guide*. The `ldm`(1M) man page is part of the `SUNWldm` package and is installed when the `SUNWldm` package is installed.

To execute the `ldm` command, you must have the `/opt/SUNWldm/bin` directory in your UNIX `$PATH` variable. To access the `ldm`(1M) man page, add the directory path `/opt/SUNWldm/man` to the variable `$MANPATH`. Both are shown as follows:

```
$ PATH=$PATH:/opt/SUNWldm/bin; export PATH (for Bourne or K shell)
$ MANPATH=$MANPATH:/opt/SUNWldm/man; export MANPATH
% set PATH=($PATH /opt/SUNWldm/bin) (for C shell)
% set MANPATH=($MANPATH /opt/SUNWldm/man)
```

Virtual Input/Output

In a Logical Domains environment, an administrator can provision up to 32 domains on a Sun Fire™ or SPARC Enterprise T1000 or T2000 server. Though each domain can be assigned dedicated CPUs and memory, the limited number of I/O buses and physical I/O slots in these systems makes it impossible to provide all domains exclusive access to the disk and network devices. Though some physical devices can be shared by splitting the PCI Express® (PCIe) bus into two (see "Configuring PCI Express Busses Across Multiple Logical Domains" on page 71), it is not sufficient to provide all domains exclusive device access. This lack of direct physical I/O device access is addressed by implementing a virtualized I/O model.

All logical domains with no direct I/O access are configured with virtual I/O devices that communicate with a service domain, which runs a service to provide access to a physical device or its functions. In this client-server model, virtual I/O devices either communicate with each other or a service counterpart through interdomain communication channels called logical domain channels (LDCs). In Logical Domains 1.1 software, the virtualized I/O functionality comprises support for virtual networking, storage, and consoles.

Virtual Network

The virtual network support is implemented using two components: the virtual network and virtual network switch device. The virtual network (`vnet`) device emulates an Ethernet device and communicates with other `vnet` devices in the system using a point-to-point channel. The virtual switch (`vsw`) device mainly functions as a multiplexor of all the virtual network's incoming and outgoing packets. The `vsw` device interfaces directly with a physical network adapter on a service domain, and sends and receives packets on a virtual network's behalf. The `vsw` device also functions as a simple layer-2 switch and switches packets between the `vnet` devices connected to it within the system.

Virtual Storage

The virtual storage infrastructure enables logical domains to access block-level storage that is not directly assigned to them through a client-server model. It consists of two components: a virtual disk client (`vdc`) that exports as a block device interface; and a virtual disk service (`vds`) that processes disk requests on behalf of the virtual disk client and submits them to the physical storage residing on the service domain. Although the virtual disks appear as regular disks on the client domain, all disk operations are forwarded to the physical disk through the virtual disk service.

Virtual Console

In a Logical Domains environment, console I/O from all domains, except the `primary` domain, is redirected to a service domain running the virtual console concentrator (`vcc`) and virtual network terminal server, instead of the systems controller. The virtual console concentrator service functions as a concentrator for all domains' console traffic, and interfaces with the virtual network terminal server daemon (`vntsd`) and provides access to each console through a UNIX socket.

Dynamic Reconfiguration

Dynamic reconfiguration (DR) is the ability to add or remove resources while the operating system is running. The ability to perform dynamic reconfiguration of a particular resource type is dependent on having support in the OS running in the logical domain. Support for dynamic reconfiguration of virtual CPUs is available in all versions of the Solaris 10 OS. Dynamic reconfiguration of virtual I/O devices is supported in logical domains running Solaris 10 10/08 at a minimum. There is no support for dynamic reconfiguration of memory and physical I/O devices. To use the dynamic reconfiguration capability in the Logical Domains Manager CLI, you must have the Logical Domains dynamic reconfiguration daemon, `drd`(1M) running in the domain you want to change.

Delayed Reconfiguration

In contrast to dynamic reconfiguration operations that take place immediately, delayed reconfiguration operations take effect after the next reboot of the OS or stop and start of the logical domain if no OS is running. Any add or remove operations on active logical domains, except `add-vcpu`, `set-vcpu`, and `remove-vcpu` subcommands, are considered delayed reconfiguration operations if you have Solaris 10 5/08 OS or earlier running in the domain. If you have Solaris 10 10/08 OS

running in the domain, the addition and removal of virtual input/output devices do not result in a delayed configuration. The `set-vswitch` subcommand on an active logical domain is considered a delayed reconfiguration operation no matter what Solaris OS is running in the domain.

If you are using a Sun UltraSPARC T1 processor, when the Logical Domains Manager is first installed and enabled (or when the configuration is restored to `factory-default`), the LDoms Manager runs in the configuration mode. In this mode, reconfiguration requests are accepted and queued up, but are not acted upon. This allows a new configuration to be generated and stored to the SC without affecting the state of the running machine, and therefore, without being encumbered by any of the restrictions around things like delayed reconfiguration and reboot of I/O domains.

Once a delayed reconfiguration is in progress for a particular logical domain, any other reconfiguration requests for that logical domain are also deferred until the domain is rebooted or stopped and started. Also, when there is a delayed reconfiguration outstanding for one logical domain, reconfiguration requests for other logical domains are severely restricted and will fail with an appropriate error message.

Even though attempts to remove virtual I/O devices on an active logical domain are handled as a delayed reconfiguration operation if you are running the Solaris 10 5/08 OS or earlier, some configuration change does occur immediately in the domain. This means the device will stop functioning as soon as the associated Logical Domains Manager CLI operation is invoked. This issue does not apply if you are running the Solaris 10 10/08 OS in the domain, since the entire removal happens immediately as part of a virtual I/O dynamic reconfiguration operation.

The Logical Domains Manager subcommand `remove-reconf` cancels delayed reconfiguration operations. You can list delayed reconfiguration operations by using the `ldm list-domain` command. Refer to the `ldm`(1M) man page or the *Logical Domains (LDoms) Manager Man Page Guide* for more information about how to use the delayed reconfiguration feature.

Note – You cannot use the `ldm remove-reconf` command if any other `ldm remove-*` commands have been issued on virtual I/O devices. The `ldm remove-reconf` command fails in these circumstances.

Persistent Configurations

The current configuration of a logical domain can be stored on the system controller (SC) using the Logical Domains Manager CLI commands. You can add a configuration, specify a configuration to be used, remove a configuration, and list the configurations on the system controller. (Refer to the `ldm`(1M) man page or the *Logical Domains (LDoms) Manager Man Page Guide*.) In addition, there is an ALOM CMT Version 1.3 command that enables you to select a configuration to boot (see "Using LDoms With ALOM CMT" on page 171).

CHAPTER 2

Security

This chapter describes the Solaris Security Toolkit software and how you can use it to secure the Solaris OS in your logical domains.

Security Considerations

The Solaris Security Toolkit software, informally known as the JumpStart™ Architecture and Security Scripts (JASS) toolkit, provides an automated, extensible, and scalable mechanism to build and maintain secure Solaris OS systems. The Solaris Security Toolkit provides security for devices critical to the management of your server, including the control domain in the Logical Domains Manager.

The Solaris Security Toolkit 4.2 software package, `SUNWjass`, provides the means to secure the Solaris Operating System on your control domain through the use of the `install-ldm` script by:

- Letting the Solaris Security Toolkit automatically harden your control domain by using the Logical Domains Manager install script (`install-ldm`) and the control driver specific to the Logical Domains Manager (`ldm_control-secure.driver`).
- Selecting an alternative driver when using the install script.
- Selecting no driver when using the install script and applying your own Solaris hardening.

The `SUNWjass` package is located with the Logical Domains (LDoms) Manager 1.1 software package, `SUNWldm`, at Sun's software download web site. You have the option to download and install the Solaris Security Toolkit 4.2 software package at the same time you download and install the Logical Domains Manager 1.1 software. The Solaris Security Toolkit 4.2 software package includes the required patches to enable the Solaris Security Toolkit software to work with the Logical Domains

Manager. Once the software is installed, you can harden your system with Solaris Security Toolkit 4.2 software. Chapter 3 tells you how to install and configure the Solaris Security Toolkit, and harden your control domain.

Following are the security functions available to users of the Logical Domains Manager provided by the Solaris Security Toolkit:

- *Hardening* – Modifying Solaris OS configurations to improve a system's security using the Solaris Security Toolkit 4.2 software with required patches to enable the Solaris Security Toolkit to work with the Logical Domains Manager.
- *Minimizing* – Installing the minimum number of core Solaris OS packages necessary for LDoms and LDoms Management Information Base (MIB) support.
- *Authorization* – Setting up authorization using the Solaris OS Role-Based Access Control (RBAC) adapted for the Logical Domains Manager.
- *Auditing* – Using the Solaris OS Basic Security module (BSM) adapted for the Logical Domains Manager to identify the source of security changes to the system to determine what was done, when it was done, by whom, and what was affected.
- *Compliance* – Determining if a system's configuration is in compliance with a predefined security profile using the Solaris Security Toolkit's auditing feature.

Solaris Security Toolkit and the Logical Domains Manager

Chapter 3 tells you how to install the Solaris Security Toolkit to make it work with the Logical Domains Manager. You would install the Solaris Security Toolkit on the control domain, which is where the Logical Domains Manager runs. You can also install the Solaris Security Toolkit on the other logical domains. The only difference would be that you would use the `ldm_control-secure.driver` to harden the control domain and you would use another driver, such as the `secure.driver`, to harden the other logical domains. This is because the `ldm_control-secure.driver` is specific to the control domain. The `ldm_control-secure.driver` is based on the `secure.driver` and has been customized and tested for use with the Logical Domains Manager. Refer to the *Solaris Security Toolkit 4.2 Reference Manual* for more information about the `secure.driver`.

Hardening

The driver (`ldm_control-secure.driver`) that Solaris Security Toolkit uses to harden the Solaris OS on the control domain is specifically tailored so that the Logical Domains Manager can run with the OS. The `ldm_control-secure.driver` is analogous to the `secure.driver` described in the *Solaris Security Toolkit 4.2 Reference Manual*.

The `ldm_control-secure.driver` provides a baseline configuration for the control domain of a system running the Logical Domains Manager software. It is intended to provide fewer system services than typical for a Solaris OS domain, reserving the control domain for Logical Domains Manager operations, rather than general usage.

The `install-ldm` script installs the Logical Domains Manager software if it is not already installed, and enables the software.

Following is a short summary of the other notable changes from `secure.driver`.

- The Telnet server is disabled from running. You can use Secure Shell (`ssh`) instead. You also can still use the Telnet client to access virtual consoles started by the Logical Domains virtual network terminal server daemon (`vntsd`). For example, if a virtual console is running that is listening to TCP port 5001 on the local system, you can access it as follows.

```
# telnet localhost 5001
```

  See "Enabling the Logical Domains Manager Daemon" on page 36 for instructions on enabling `vntsd`. It is not automatically enabled.

- The following finish scripts have been added. They enable the Logical Domains Manager to install and start. Some of these added scripts must be added to any customized drivers you make and some are optional. The scripts are marked as to whether they are required or optional.
  - `install-ldm.fin` – Installs the `SUNWldm` package. (*Required*)
  - `enable-ldmd.fin` – Enables the Logical Domains Manager daemon (`ldmd`) (*Required*)
  - `enable-ssh-root-login.fin` – Enables the superuser to directly log in through the Secure Shell (`ssh`). (*Optional*)
- The following files have changed. These changes are optional to make in any customized drivers you have and are marked as optional.
  - `/etc/ssh/sshd_config` – Root account access is allowed for the entire network. This file is not used in either driver. (*Optional*)
  - `/etc/ipf/ipf.conf` – UDP port 161 (SNMP) is opened. (*Optional*)
  - `/etc/host.allow` – The Secure Shell daemon (`sshd`) is open for the entire network, not just the local subnet. (*Optional*)

- The following finish scripts are disabled (commented out). You should comment out the `disable-rpc.fin script` in any customized driver you make. The other changes are optional. The scripts are marked as to whether they are required or optional.
  - `enable-ipfilter.fin` – IP Filter, a network packet filter, is not enabled. (*Optional*)
  - `disable-rpc.fin` – Leaves Remote Procedure Call (RPC) service enabled. The RPC service is used by many other system services, such as Network Information Services (NIS) and network file system (NFS). (*Required*)
  - `disable-sma.fin` – Leaves the System Management Agent (NET-SNMP) enabled. (*Optional*)
  - `disable-ssh-root-login.fin` – `ssh` root login cannot be disabled.
  - `set-term-type.fin` – Unneeded legacy script. (*Optional*)

Minimizing Logical Domains

The Solaris OS can be configured with different quantities of packages, depending on your needs. Minimization reduces this set of packages to the bare minimum required to run your desired applications. Minimization is important because it reduces the amount of software containing potential security vulnerabilities and also reduces the level of effort associated with keeping the installed software properly patched. The logical domain minimization activity provides JumpStart™ support for installing a minimized Solaris OS that still fully supports any domain.

The Solaris Security Toolkit provides a JumpStart profile, `minimal-ldm_control.profile`, for minimizing a logical domain for LDoms, which installs all the Solaris OS packages necessary for LDoms and LDoms MIB support. If you want to use the LDoms MIB on the control domain, you need to add that package separately after you install the LDoms and Solaris Security Toolkit packages. It is not installed automatically with the other software. Refer to the *Logical Domains (LDoms) MIB 1.0.1 Administration Guide* for more information about installing and using the LDoms MIB.

Authorization

Authorization for the Logical Domains Manager has two levels:

- Read – allows you to view, but not modify the configuration.
- Read and write – allows you to view and change the configuration.

The changes are not made to the Solaris OS, but are added to the authorization file by the package script `postinstall` when the Logical Domains Manager is installed. Similarly, the authorization entries are removed by the package script `preremove`.

The following table lists the `ldm` subcommands with the corresponding user authorization that is needed to perform the commands.

TABLE 2-1 The `ldm` Subcommands and User Authorizations

ldm Subcommand*	User Authorization
`add-*`	`solaris.ldoms.write`
`bind-domain`	`solaris.ldoms.write`
`list`	`solaris.ldoms.read`
`list-*`	`solaris.ldoms.read`
`panic-domain`	`solaris.ldoms.write`
`remove-*`	`solaris.ldoms.write`
`set-*`	`solaris.ldoms.write`
`start-domain`	`solaris.ldoms.write`
`stop-domain`	`solaris.ldoms.write`
`unbind-domain`	`solaris.ldoms.write`

* Refers to all the resources you can add, list, remove, or set.

Auditing

Auditing the Logical Domains Manager CLI commands is done with Solaris OS Basic Security module (BSM) auditing. Refer to the Solaris 10 *System Administration Guide: Security Services* for detailed information about using Solaris OS BSM auditing.

BSM auditing is not enabled by default for the Logical Domains Manager; however, the infrastructure is provided. You can enable BSM auditing in one of two ways:

- Run the `enable-bsm.fin` finish script in the Solaris Security Toolkit.
- Use the Solaris OS `bsmconv`(1M) command.

For further details about enabling, verifying, disabling, printing output, and rotating logs using BSM auditing with the Logical Domains Manager, see "Enabling and Using BSM Auditing" on page 172.

Compliance

Solaris Security Toolkit does have its own auditing capabilities. The Solaris Security Toolkit software can automatically validate the security posture of any system running the Solaris OS by comparing it with a predefined security profile. Refer to "Auditing System Security" in the *Solaris Security Toolkit 4.2 Administration Guide* for more information about this compliance function.

CHAPTER 3

Installing and Enabling Software

This chapter describes how to install or upgrade the different software components required to enable the Logical Domains 1.1 software. Using the Logical Domains software requires the following components:

- Supported platform, refer to "Supported Platforms" in the *Logical Domains (LDoms) 1.1 Release Notes* for a list of supported platforms.
- Control domain running an operating system at least equivalent to the Solaris 10/08 OS with any patches recommended in "Required Software and Patches" in the *Logical Domains (LDoms) 1.1 Release Notes*. See "Upgrading the Solaris OS" on page 16.
- System firmware version 6.7.*x* for your Sun UltraSPARC T1 platform or system firmware version 7.2.*x* for your Sun UltraSPARC T2 or T2 Plus platform at a minimum. See "Upgrading the System Firmware" on page 21.
- LDoms 1.1 software installed and enabled on the control domain. See "Installing the Logical Domains Manager and Solaris Security Toolkit" on page 24.
- (Optional) the Solaris Security Toolkit 4.2 software. See "Installing the Logical Domains Manager and Solaris Security Toolkit" on page 24
- (Optional) the Logical Domains (LDoms) Management Information Base (MIB) software package. Refer to the *Logical Domains (LDoms) Management Information Base (MIB) 1.0.1 Administration Guide* for more information about using the LDoms MIB.
- (Optional) Libvirt for LDoms 1.0.1 software packages. Refer to the *Libvirt for LDoms 1.0.1 Administration Guide* for more information about using Libvirt for LDoms.

The Solaris OS and the system firmware must be installed or upgraded on your server before you install or upgrade the Logical Domains Manager. If your system is already using Logical Domains software, see "Upgrading a System Already Using Logical Domains" on page 16. Otherwise, see "Installing Logical Domains Software on a New System" on page 21.

Upgrading a System Already Using Logical Domains

Upgrading the Solaris OS

If your system is already configured with the Logical Domain software, then the control domain has to be upgraded. The other existing domains also have to be upgraded if you want to be able to use all features of the Logical Domains 1.1 software.

Refer to "Required Software and Patches" in the *Logical Domains (LDoms) 1.1 Release Notes* to find the Solaris 10 OS that you should use for this version of the Logical Domains software, and the required and recommended patches for the different domains. Refer to the Solaris 10 10/08 installation guide for complete instructions for upgrading the Solaris OS.

When upgrading the Solaris OS in the control domain, you need to save the Logical Domains constraints database file. This section contains information you need to know about saving and restoring the Logical Domains constraints database file.

Saving and Restoring the Logical Domains Constraints Database File

Whenever you upgrade the operating system on the control domain, you must save and restore the Logical Domains constraints database file that can be found in `/var/opt/SUNWldm/ldom-db.xml`.

Note – You must also save and restore the `/var/opt/SUNWldm/ldom-db.xml` file when you perform any other operation that is destructive to the control domain's file data, such as a disk swap.

Preserving the Logical Domains Constraints Database File When Using Live Upgrade

If you are using live upgrade on the control domain, consider adding the following line to the `/etc/lu/synclist` file:

```
/var/opt/SUNWldm/ldom-db.xml      OVERWRITE
```

This causes the database to be copied automatically from the active boot environment to the new boot environment when switching boot environments. For more information about `/etc/lu/synclist` and synchronizing files between boot environments, refer to "Synchronizing Files Between Boot Environments" in the *Solaris 10 8/07 Installation Guide: Solaris Live Upgrade and Upgrade Planning*.

Upgrading From Solaris 10 OS Older Than Solaris 10 5/08 OS

If the control domain is upgraded from a Solaris 10 OS version older than Solaris 10 5/08 OS (or without patch 127127-11), and if volume manager volumes were exported as virtual disks, then the virtual disk backends must be re-exported with `options=slice` after the Logical Domain Manager has been upgraded. See "Exporting Volumes and Backward Compatibility" on page 88 for more information.

Upgrading the Logical Domains Manager and the System Firmware

This section shows how to upgrade to LDoms 1.1 software.

First download the Logical Domains Manager and the Solaris Security Toolkit on the control domain, see "Downloading Logical Domains Manager and Solaris Security Toolkit" on page 23.

Then stop all domains (except the control domain) running on the platform:

▼ Stop All Domains Running on the Platform, Except the Control Domain

1. Bring down each domain to the `ok` prompt.

2. **Issue the `stop-domain` subcommand from the control domain for each domain.**

```
primary# ldm stop-domain ldom
```

3. **Issue the `unbind-domain` subcommand from the control domain for each domain.**

```
primary# ldm unbind-domain ldom
```

Upgrading to LDoms 1.1 Software

This section shows how to upgrade to LDoms 1.1 software.

Perform the procedure "Upgrade From LDoms 1.0 Software" on page 18 if you want to use your existing LDoms 1.0 configurations with LDoms 1.1 software. Existing LDoms 1.0 configurations do *not* work in LDoms 1.1 software.

If you are upgrading from LDoms 1.0.1, 1.0.2, or 1.0.3 software, perform the procedure "Upgrade From LDoms 1.0.1, 1.0.2, or 1.0.3" on page 20. Existing LDoms 1.0.1, 1.0.2, and 1.0.3 configurations *do* work in LDoms 1.1 software.

▼ Upgrade From LDoms 1.0 Software

Existing LDoms 1.0 configurations do *not* work in LDoms 1.1 software, so you need to save your LDoms 1.0 configurations before the upgrade to use them in LDoms 1.1 software. The following procedure describes a method for saving and rebuilding a configuration using XML constraints files and the `-i` option to the `ldm add-domain` command.

The basic process is to save the constraints information for each domain into an XML file, which can then be re-issued to the Logical Domains Manager after the upgrade to rebuild a desired configuration.

The procedure in this section works for guest domains, not the control domain. Although you can save the control (`primary`) domain's constraints to an XML file, you cannot feed it back into the `ldm add-domain -i` command. However, you can use the resource constraints from the XML file to create the CLI commands to reconfigure your primary domain. See "Rebuilding the Control Domain" on page 63 for instructions on how to translate typical XML output from an `ldm list-constraints -x` primary command into the CLI commands needed to reconfigure a primary domain.

The method that follows does not preserve actual bindings, only the constraints used to create those bindings. This means that, after this procedure, the domains will have the same virtual resources, but will not necessarily be bound to the same physical resources.

1. **For each domain, create an XML file containing the domain's constraints.**

```
# ldm ls-constraints -x ldom > ldom.xml
```

2. **List all the logical domain configurations stored on the system controller.**

```
# ldm ls-config
```

3. **Remove each logical domain configuration stored on the system controller.**

```
# ldm rm-config config_name
```

4. **Disable the Logical Domains Manager daemon (`ldmd`).**

```
# svcadm disable ldmd
```

5. **Remove the Logical Domains Manager package (`SUNWldm`).**

```
# pkgrm SUNWldm
```

6. **Remove the Solaris Security Toolkit package (`SUNWjass`) if you are using that.**

```
# pkgrm SUNWjass
```

7. **Flash update the system firmware. For the entire procedure, see** "Upgrade System Firmware" on page 21 **or** "Upgrade System Firmware Without an FTP Server" on page 23**.**

8. **Reinstall the Logical Domain Manager and the Solaris Security Toolkit. See** "Installing the Logical Domains Manager and Solaris Security Toolkit" on page 24**.**

9. **Reconfigure the `primary` domain manually. For instructions, see** "Set Up the Control Domain" on page 48**.**

10. **Run the following commands for each guest domain's XML file you created in Step 1.**

```
# ldm add-domain -i ldom.xml
# ldm bind-domain ldom
# ldm start-domain ldom
```

▼ Upgrade From LDoms 1.0.1, 1.0.2, or 1.0.3

1. **Flash update the system firmware. For the entire procedure, see** "Upgrade System Firmware" on page 21 **or** "Upgrade System Firmware Without an FTP Server" on page 23**.**

2. **Disable the Logical Domains Manager daemon (`ldmd`).**

```
# svcadm disable ldmd
```

3. **Remove the old `SUNWldm` package.**

```
# pkgrm SUNWldm
```

4. **Add the new `SUNWldm` package.**

   Specifying the `-d` option assumes that the package is in the current directory.

```
# pkgadd -Gd . SUNWldm
```

5. **Refresh the Logical Domains Manager daemon (`ldmd`).**

```
# svcadm refresh ldmd
```

6. **Enable the Logical Domains Manager daemon (`ldmd`).**

```
# svcadm enable ldmd
```

7. **Use the `ldm list` command to verify that the Logical Domains Manager is running.**

   You receive a message similar to the following, which is for the `factory-default` configuration. Note that the `primary` domain is `active`, which means that the Logical Domains Manager is running.

```
# ldm list
NAME             STATE    FLAGS   CONS    VCPU  MEMORY   UTIL  UPTIME
primary          active   ---c-   SP      32    3264M    0.3%  19d 9m
```

Installing Logical Domains Software on a New System

Sun platforms supporting Logical Domains software come preinstalled with the Solaris 10 OS. Initially, the Logical Domains software is not enabled, and the platform appears as a single system hosting only one operating system. After the Solaris OS, system firmware, and Logical Domains Manager have been installed, the original system and instance of the Solaris OS become the control domain. That first domain of the platform is named `primary`, and you cannot change that name or destroy that domain. From there, the platform can be reconfigured to have multiple domains hosting different instances of the Solaris OS.

Updating the Solaris OS

On a brand new system, you may want to reinstall the OS so that it conforms to your installation policy. In that case, refer to "Required and Recommended Solaris OS" in the Logical Domains (LDoms) 1.1 Release Notes to find the Solaris 10 OS that you should use for this version of the Logical Domains software. Refer to your Solaris 10 OS installation guide for complete instructions for installing the Solaris OS. You can tailor your installation to the needs of your system.

If your system is already installed then it needs to be upgraded to the appropriate Solaris 10 OS that should be used for this version of the Logical Domains software. Refer to "Required Software and Patches" in the Logical Domains (LDoms) 1.1 Release Notes to find the Solaris 10 OS that you should use for this version of the Logical Domains software and the required and recommended patches. Refer to the Solaris 10 10/08 Release and Installation Collection for complete instructions for upgrading the Solaris OS.

Upgrading the System Firmware

▼ Upgrade System Firmware

You can find system firmware for your platform at the SunSolve site:

`http://sunsolve.sun.com`

Refer to “Required System Firmware Patches” in the *Logical Domains (LDoms) 1.1 Release Notes* for required system firmware by supported servers.

This procedure describes how to upgrade system firmware using the `flashupdate`(1M) command on your system controller.

- If you do not have access to a local FTP server, see "Upgrade System Firmware Without an FTP Server" on page 23.
- If you want to update the system firmware from the control domain, refer to your system firmware release notes.

Refer to the administration guides or product notes for the supported servers for more information about installing and updating system firmware for these servers.

1. **Shut down and power off the host server from either management port connected to the system controller: serial or network.**

```
# shutdown -i5 -g0 -y
```

2. **Use the `flashupdate`(1M) command to upgrade the system firmware, depending on your server.**

```
sc> flashupdate -s IP-address -f path/Sun_System_Firmware-
x_x_x_build_nn-server-name.bin
username: your-userid
password: your-password
```

   Where:

   - *IP-address* is the IP address of your FTP server.
   - *path* is the location in SunSolve℠ or your own directory where you can obtain the system firmware image.
   - *x_x_x* is the version number of the System Firmware.
   - *nn* is the number of the build that applies to this release.
   - *server-name* is the name of your server. For example, the *server-name* for the Sun Fire T2000 server is `Sun_Fire_T2000`.

3. **Reset the system controller.**

```
sc> resetsc -y
```

4. **Power on and boot the host server.**

```
sc> poweron -c
ok boot disk
```

▼ Upgrade System Firmware Without an FTP Server

If you do not have access to a local FTP server to upload firmware to the system controller, you can use the `sysfwdownload` utility, which is provided with your system firmware upgrade package on the SunSolve site:

`http://sunsolve.sun.com`

1. Run the following commands within the Solaris OS.

```
# cd firmware_location
# sysfwdownload system_firmware_file
```

2. Shut down the Solaris OS instance.

```
# shutdown -i5 -g0 -y
```

3. Power off and update the firmware on the system controller.

```
sc> poweroff -fy
sc> flashupdate -s 127.0.0.1
```

4. Reset and power on the system controller.

```
sc> resetsc -y
sc> poweron
```

Downloading Logical Domains Manager and Solaris Security Toolkit

▼ Download the Software

1. Download the zip file (`LDoms_Manager-1_1.zip`) from the Sun Software Download site. You can find the software from this web site:

`http://www.sun.com/ldoms`

2. **Unzip the zip file.**

```
$ unzip LDoms_Manager-1_1.zip
```

The Logical Domains Manager and the Solaris Security Toolkit are bundled in the same zip file. Refer to "Location of Logical Domains 1.1 Software" in the *Logical Domains (LDoms) 1.1 Release Notes* for details about the structure of the file and what it includes.

Installing the Logical Domains Manager and Solaris Security Toolkit

There are three methods of installing Logical Domains Manager and Solaris Security Toolkit software:

- Using the installation script to install the packages and patches. This automatically installs both the Logical Domains Manager and the Solaris Security Toolkit software. See "Installing the Logical Domains Manager and Solaris Security Toolkit Software Automatically" on page 24.
- Using JumpStart to install the packages. See "Using JumpStart to Install the Logical Domains Manager 1.1 and Solaris Security Toolkit 4.2 Software" on page 31.
- Installing each package manually. See "Installing Logical Domains Manager and Solaris Security Toolkit Software Manually" on page 34.

Note – Remember that you need to manually install the LDoms MIB software package after you install the LDoms and Solaris Security Toolkit packages. It is not automatically installed with the other packages. Refer to the *Logical Domains (LDoms) Management Information Base 1.0.1 Administration Guide* for more information about installing and using the LDoms MIB.

Installing the Logical Domains Manager and Solaris Security Toolkit Software Automatically

If you use the `install-ldm` installation script, you have several choices to specify how you want the script to run. Each choice is described in the procedures that follow.

- **Using the `install-ldm` script with no options does the following automatically:**
  - Checks that the Solaris OS release is Solaris 10 11/06 at a minimum

- Verifies that the package subdirectories `SUNWldm/` and `SUNWjass/` are present
- Verifies that the prerequisite Solaris Logical Domains driver packages, `SUNWldomr` and `SUNWldomu`, are present
- Verifies that the `SUNWldm` and `SUNWjass` packages have not been installed

---

Note – If the script does detect a previous version of `SUNWjass` during installation, you must remove it. You do *not* need to undo any previous hardening of your Solaris OS.

- Installs the Logical Domains Manager 1.1 software (`SUNWldm` package)
- Installs the Solaris Security Toolkit 4.2 software including required patches (`SUNWjass` package)
- Verifies that all packages are installed
- Enables the Logical Domains Manager daemon, `ldmd`
- Hardens the Solaris OS on the control domain with the Solaris Security Toolkit `ldm_control-secure.driver` or one of the other drivers ending in `-secure.driver` that you select.

- **Using the `install-ldm` script with option `-d`** allows you to specify a Solaris Security Toolkit driver other than a driver ending with `-secure.driver`. This option automatically performs all the functions listed in the preceding choice with the added option:
  - Hardens the Solaris OS on the control domain with the Solaris Security Toolkit customized driver that you specify; for example, the `server-secure-myname.driver`.
- **Using the `install-ldm` script with option `-d` and specifying `none`** specifies that you do *not* want to harden the Solaris OS running on your control domain by using the Solaris Security Toolkit. This option automatically performs all the functions except hardening listed in the preceding choices. Bypassing the use of the Solaris Security Toolkit is not suggested and should only be done when you intend to harden your control domain using an alternate process.
- **Using the `install-ldm` script with option `-p`** specifies that you only want to perform the post-installation actions of enabling the Logical Domains Manager daemon (`ldmd`) and running the Solaris Security Toolkit. For example, you would use this option if the `SUNWldm` and `SUNWjass` packages are preinstalled on your server. See "Enable the Logical Domains Manager Daemon and Run the Solaris Security Toolkit Only" on page 31

▼ Install With No Special Options

- **Run the `install-ldm` installation script with no options.**

  The installation script is part of the `SUNWldm` package and is in the `Install` subdirectory.

```
# Install/install-ldm
```

a. If one or more packages are previously installed, you receive this message.

```
# Install/install-ldm
ERROR: One or more packages are already installed: SUNWldm SUNWjass.
If packages SUNWldm.v and SUNWjass are factory pre-installed, run
install-ldm -p to perform post-install actions.  Otherwise remove the
package(s) and restart install-ldm.
```

If you want to perform post-installation actions only, go to "Enable the Logical Domains Manager Daemon and Run the Solaris Security Toolkit Only" on page 31.

b. If the process is successful, you receive messages similar to the following examples.

- Code Example 3-2 shows a successful run of the `install-ldm` script if you choose the following default security profile:

  ```
  a) Hardened Solaris configuration for LDoms (recommended)
  ```

- Code Example 3-3 shows a successful run of the `install-ldm` script if you choose the following security profile:

  ```
  c) Your custom-defined Solaris security configuration
  profile
  ```

  The drivers that are displayed for you to choose are drivers ending with `-secure.driver`. If you write a customized driver that does not end with `-secure.driver`, you must specify your customized driver with the `install-ldm -d` option. (See "Install With a Customized Hardening Driver" on page 29.)

CODE EXAMPLE 3-1 Output From Hardened Solaris Configuration for LDoms

```
# Install/install-ldm
Welcome to the LDoms installer.

You are about to install the domain manager package that will enable
you to create, destroy and control other domains on your system. Given
the capabilities of the domain manager, you can now change the security
configuration of this Solaris instance using the Solaris Security
Toolkit.

Select a security profile from this list:

a) Hardened Solaris configuration for LDoms (recommended)
b) Standard Solaris configuration
c) Your custom-defined Solaris security configuration profile

Enter a, b, or c [a]: a
The changes made by selecting this option can be undone through the
Solaris Security Toolkit's undo feature. This can be done with the
'/opt/SUNWjass/bin/jass-execute -u' command.

Installing LDoms and Solaris Security Toolkit packages.
pkgadd -n -d "/var/tmp/install/Product/Logical_Domain_Manager" -a pkg_admin
SUNWldm.v
Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
Use is subject to license terms.

Installation of <SUNWldm> was successful.
pkgadd -n -d "/var/tmp/install/Product/Solaris_Security_Toolkit" -a pkg_admin
SUNWjass
Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
Use is subject to license terms.

Installation of <SUNWjass> was successful.

Verifying that all packages are fully installed.  OK.
Enabling services: svc:/ldoms/ldmd:default
Running Solaris Security Toolkit 4.2.0 driver ldm_control-secure.driver.
Please wait. . .
/opt/SUNWjass/bin/jass-execute -q -d ldm_control-secure.driver
Executing driver, ldm_control-secure.driver
Solaris Security Toolkit hardening executed successfully; log file
/var/opt/SUNWjass/run/20070208142843/jass-install-log.txt.  It will not
take effect until the next reboot.  Before rebooting, make sure SSH or
the serial line is setup for use after the reboot.
```

CODE EXAMPLE 3-2 Output From Choosing Customized Configuration Profile

```
# Install/install-ldm
Welcome to the LDoms installer.

You are about to install the domain manager package that will enable
you to create, destroy and control other domains on your system. Given
the capabilities of the domain manager, you can now change the security
configuration of this Solaris instance using the Solaris Security
Toolkit.

Select a security profile from this list:

a) Hardened Solaris configuration for LDoms (recommended)
b) Standard Solaris configuration
c) Your custom-defined Solaris security configuration profile

Enter a, b, or c [a]: c
Choose a Solaris Security Toolkit .driver configuration profile from
this list
1) ldm_control-secure.driver
2) secure.driver
3) server-secure.driver
4) suncluster3x-secure.driver
5) sunfire_15k_sc-secure.driver

Enter a number 1 to 5: 2
The driver you selected may not perform all the LDoms-specific
operations specified in the LDoms Administration Guide.
Is this OK (yes/no)? [no] y
The changes made by selecting this option can be undone through the
Solaris Security Toolkit's undo feature. This can be done with the
'/opt/SUNWjass/bin/jass-execute -u' command.

Installing LDoms and Solaris Security Toolkit packages.
pkgadd -n -d "/var/tmp/install/Product/Logical_Domain_Manager" -a pkg_admin
SUNWldm.v
Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
Use is subject to license terms.

Installation of <SUNWldm> was successful.
pkgadd -n -d "/var/tmp/install/Product/Solaris_Security_Toolkit" -a pkg_admin
SUNWjass
Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
Use is subject to license terms.

Installation of <SUNWjass> was successful.

Verifying that all packages are fully installed.  OK.
```

CODE EXAMPLE 3-2 Output From Choosing Customized Configuration Profile *(Continued)*

```
Enabling services: svc:/ldoms/ldmd:default
Running Solaris Security Toolkit 4.2.0 driver secure.driver.
Please wait. . .
/opt/SUNWjass/bin/jass-execute -q -d secure.driver
Executing driver, secure.driver
Solaris Security Toolkit hardening executed successfully; log file
/var/opt/SUNWjass/run/20070102142843/jass-install-log.txt.  It will not
take effect until the next reboot.  Before rebooting, make sure SSH or
the serial line is setup for use after the reboot.
```

▼ Install With a Customized Hardening Driver

- **Run the** `install-ldm` **installation script with the** `-d` **option to specify a Solaris Security Toolkit customized hardening driver; for example,** `server-secure-myname.driver`**.**

  The installation script is part of the `SUNWldm` package and is in the `Install` subdirectory.

```
# Install/install-ldm -d server-secure-myname.driver
```

  If the process is successful, you receive messages similar to that in Code Example 3-4.

CODE EXAMPLE 3-3 Output From Successful Run of the install-ldm -d Script

```
# Install/install-ldm -d server-secure.driver
The driver you selected may not perform all the LDoms-specific
operations specified in the LDoms Administration Guide.
Installing LDoms and Solaris Security Toolkit packages.
pkgadd -n -d "/var/tmp/install/Product/Logical_Domain_Manager" -a pkg_admin
SUNWldm.v
Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
Use is subject to license terms.

Installation of <SUNWldm> was successful.
pkgadd -n -d "/var/tmp/install/Product/Solaris_Security_Toolkit" -a pkg_admin
SUNWjass
Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
Use is subject to license terms.

Installation of <SUNWjass> was successful.

Verifying that all packages are fully installed.  OK.
Enabling services: svc:/ldoms/ldmd:default
Running Solaris Security Toolkit 4.2.0 driver server-secure-myname.driver.
```

CODE EXAMPLE 3-3 Output From Successful Run of the install-ldm -d Script *(Continued)*

```
Please wait. . .
/opt/SUNWjass/bin/jass-execute -q -d server-secure-myname.driver
Executing driver, server-secure-myname.driver
Solaris Security Toolkit hardening executed successfully; log file
/var/opt/SUNWjass/run/20061114143128/jass-install-log.txt.  It will not
take effect until the next reboot.  Before rebooting, make sure SSH or
the serial line is setup for use after the reboot.
```

▼ Install and Do Not Harden Your System

- **Run the `install-ldm` installation script with the `-d none` option to specify *not* to harden your system using a Solaris Security Toolkit driver.**

  The installation script is part of the `SUNWldm` package and is in the `Install` subdirectory.

```
# Install/install-ldm -d none
```

  If the process is successful, you receive messages similar to the example shown in Code Example 3-5.

CODE EXAMPLE 3-4 Output From Successful Run of the install-ldm -d none Script

```
# Install/install-ldm -d none
Installing LDoms and Solaris Security Toolkit packages.
pkgadd -n -d "/var/tmp/install/Product/Logical_Domain_Manager" -a pkg_admin
SUNWldm.v
Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
Use is subject to license terms.

Installation of <SUNWldm> was successful.
pkgadd -n -d "/var/tmp/install/Product/Solaris_Security_Toolkit" -a pkg_admin
SUNWjass
Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
Use is subject to license terms.

Installation of <SUNWjass> was successful.

Verifying that all packages are fully installed.  OK.
Enabling services: svc:/ldoms/ldmd:default
Solaris Security Toolkit was not applied. Bypassing the use of the
Solaris Security Toolkit is not recommended and should only be
performed when alternative hardening steps are to be taken.
```

▼ Enable the Logical Domains Manager Daemon and Run the Solaris Security Toolkit Only

You might use this option if the `SUNWldm` and `SUNWjass` packages are preinstalled on your server and you want to perform the post-installation actions of enabling the Logical Domains Manager daemon (`ldmd`) and running the Solaris Security Toolkit.

- **Run the `install-ldm` installation script with the `-p` option to perform only the post-installation actions of enabling `ldmd` and running the Solaris Security Toolkit to harden your system.**

```
# Install/install-ldm -p
Verifying that all packages are fully installed.  OK.
Enabling services: svc:/ldoms/ldmd:default
Running Solaris Security Toolkit 4.2.0 driver ldm_control-secure.driver.
Please wait. . .
/opt/SUNWjass/bin/jass-execute -q -d ldm_control-secure.driver
Solaris Security Toolkit hardening executed successfully; log file
var/opt/SUNWjass/run/20070515140944/jass-install-log.txt.  It will not
take effect until the next reboot.  Before rebooting, make sure SSH or
the serial line is setup for use after the reboot.
```

Using JumpStart to Install the Logical Domains Manager 1.1 and Solaris Security Toolkit 4.2 Software

Refer to *JumpStart Technology: Effective Use in the Solaris Operating Environment* for complete information about using JumpStart.

Caution – Do *not* disconnect from the virtual console during a network installation.

▼ Set Up a JumpStart Server

- If you have already set up a JumpStart server, proceed to "Install Using JumpStart Software" on page 32 of this administration guide.
- If you have not already set up a JumpStart server, you must do so.

  Refer to the *Solaris 10 10/08 Installation Guide: Custom JumpStart and Advanced Installations* for complete information about this procedure.

1. **Refer to "Preparing Custom JumpStart Installations (Tasks)" in the *Solaris 10 10/08 Installation Guide: Custom JumpStart and Advanced Installations*, and perform the following steps.**

a. **Read the task map in "Task Map: Preparing Custom JumpStart Installations."**

b. **Set up networked systems with the procedures in "Creating a Profile Server for Network Systems."**

c. **Create the `rules` file with the procedure in "Creating the `rules` File."**

2. **Validate the `rules` file with the procedure in "Validating the `rules` File."**

   The Solaris Security Toolkit provides profiles and finish scripts. Refer to the *Solaris Security Toolkit 4.2 Reference Manual* for more information about profiles and finish scripts.

▼ Install Using JumpStart Software

1. **Change to the directory where you have downloaded the Solaris Security Toolkit package (`SUNWjass`).**

```
# cd /path-to-download
```

2. **Install `SUNWjass` so that it creates the JumpStart (`jumpstart`) directory structure.**

```
# pkgadd -R /jumpstart -d . SUNWjass
```

3. **Use your text editor to modify the `/jumpstart/opt/SUNWjass/Sysidcfg/Solaris_10/sysidcfg` file to reflect your network environment.**

4. **Copy the `/jumpstart/opt/SUNWjass/Drivers/user.init.SAMPLE` file to the `/jumpstart/opt/SUNWjass/Drivers/user.init` file.**

```
# cp user.init.SAMPLE user.init
```

5. **Edit the `user.init` file to reflect your paths.**

6. **To install the Solaris Security Toolkit package (`SUNWjass`) onto the target system during a JumpStart install, you must place the package in the `JASS_PACKAGE_MOUNT` directory defined in your `user.init` file. For example:**

```
# cp -r /path/to/LDoms_Manager-1_0_2/Product/SUNWjass
/jumpstart/opt/SUNWjass/Packages
```

7. **To install the Logical Domains Manager package (`SUNWldm.v`) onto the target system during a JumpStart install, you must place the package from the download area in the `JASS_PACKAGE_MOUNT` directory defined in your `user.init` file. For example:**

```
# cp -r /path/to/LDoms_Manager-1_0_2/Product/SUNWldm.v
/jumpstart/opt/SUNWjass/Packages
```

8. **If you experience problems with a multihomed JumpStart server, modify the two entries in the `user.init` file for `JASS_PACKAGE_MOUNT` and `JASS_PATCH_MOUNT` to the correct path to the `JASS_HOME_DIR/Patches` and `JASS_HOME_DIR/Packages` directories. Refer to the comments in the `user.init.SAMPLE` file for more information.**

9. **Use the `ldm_control-secure.driver` as the basic driver for the Logical Domains Manager control domain.**

   Refer to Chapter 4 in the *Solaris Security Toolkit 4.2 Reference Manual* for information about how to modify the driver for your use. The main driver in the Solaris Security Toolkit that is the counterpart to the `ldm_control-secure.driver` is the `secure.driver`.

10. **After completing the modifications to the `ldm_control-secure.driver`, make the correct entry in the rules file.**

    - If you want to minimize the LDoms control domain, specify the `minimal-ldm-control.profile` in your rules file similar to the following.

```
hostname imbulu - Profiles/minimal-ldm_control.profile
Drivers/ldm_control-secure-abc.driver
```

Note – You must manually install the LDoms MIB software package and Libvirt for LDoms packages after you install the LDoms and Solaris Security Toolkit packages. They are not automatically installed with the other packages.

    - If you do not want to minimize the LDoms control domain, your entry should be similar to the following.

```
hostname imbulu - Profiles/oem.profile Drivers/ldm_control-secure-abc.driver
```

11. **If you undo hardening during a JumpStart install, you must run the following SMF command to restart the Logical Domains Manager.**

```
# svcadm enable svc:/ldoms/ldmd:default
```

Installing Logical Domains Manager and Solaris Security Toolkit Software Manually

Perform the following procedures to install the Logical Domains Manager and Solaris Security Toolkit Software manually:

- "Install the Logical Domains Manager (LDoms) 1.1 Software Manually" on page 34.
- "(Optional) Install the Solaris Security Toolkit 4.2 Software Manually" on page 34.
- "(Optional) Harden the Control Domain Manually" on page 35.

▼ Install the Logical Domains Manager (LDoms) 1.1 Software Manually

Download the Logical Domains Manager 1.1 software, the `SUNWldm` package, from the Sun Software Download site. See "Download the Software" on page 23 for specific instructions.

1. **Use the `pkgadd`(1M) command to install the `SUNWldm.v` package. Use the `-G` option to install the package in the global zone only and the `-d` option to specify the path to the directory that contains the SUNWldm.v package.**

```
# pkgadd -Gd . SUNWldm.v
```

2. **Answer `y` for yes to all questions in the interactive prompts.**

3. **Use the `pkginfo`(1) command to verify that the `SUNWldm` package for Logical Domains Manager 1.1 software is installed.**

   The revision (`REV`) information shown below is an example.

```
# pkginfo -l SUNWldm | grep VERSION
VERSION=1.1,REV=2007.08.23.10.20
```

▼ (*Optional*) Install the Solaris Security Toolkit 4.2 Software Manually

If you want to secure your system, download and install the `SUNWjass` package. The required patches (122608-03 and 125672-01) are included in the `SUNWjass` package. See "Download the Software" on page 23 for specific instructions about downloading the software.

See Chapter 2 in this document for more information about security considerations when using Logical Domains Manager software. For further reference, you can find Solaris Security Toolkit 4.2 documentation at:

`http://docs.sun.com`

1. **Use the `pkgadd`(1M) command to install the `SUNWjass` package.**

```
# pkgadd -d . SUNWjass
```

2. **Use the `pkginfo`(1) command to verify that the `SUNWjass` package for Solaris Security Toolkit 4.2 software is installed.**

```
# pkginfo -l SUNWjass | grep VERSION
VERSION: 4.2.0
```

▼ (*Optional*) Harden the Control Domain Manually

Perform this procedure only if you have installed the Solaris Security Toolkit 4.2 package.

Note – When you use the Solaris Security Toolkit to harden the control domain, you disable many system services and place certain restrictions on network access. Refer to "Related Documentation" on page xv in this document to find Solaris Security Toolkit 4.2 documentation for more information.

---

1. **Harden using the `ldm_control-secure.driver`.**

```
# /opt/SUNWjass/bin/jass-execute -d ldm_control-secure.driver
```

   You can use other drivers to harden your system. You can also customize drivers to tune the security of your environment. Refer to the *Solaris Security Toolkit 4.2 Reference Manual* for more information about drivers and customizing them.

2. **Answer `y` for yes to all questions in the interactive prompts.**

3. **Shut down and reboot your server for the hardening to take place.**

```
# /usr/sbin/shutdown -y -g0 -i6
```

▼ Validate Hardening

- **Check whether the Logical Domains hardening driver (`ldom_control-secure.driver`) applied hardening correctly.**

  If you want to check on another driver, substitute that driver's name in this command example.

```
# /opt/SUNWjass/bin/jass-execute -a ldom_control-secure.driver
```

▼ Undo Hardening

1. **Undo the configuration changes applied by the Solaris Security Toolkit.**

```
# /opt/SUNWjass/bin/jass-execute -u
```

   The Solaris Security Toolkit asks you which hardening runs you want to undo.

2. **Select the hardening runs you want to undo.**
3. **Reboot the system so that the unhardened configuration takes place.**

```
# /usr/sbin/shutdown -y -g0 -i6
```

Note – If you undo hardening that was performed during a JumpStart installation, you must run the following SMF commands to restart the Logical Domains Manager daemon (`ldmd`) and the virtual network terminal server daemon (`vntsd`).

```
# svcadm enable svc:/ldoms/ldmd:default
```

Enabling the Logical Domains Manager Daemon

The installation script `install-ldm` automatically enables the Logical Domains Manager daemon (`ldmd`). If you have installed the Logical Domains Manager software manually, you must enable the Logical Domains Manager daemon, `ldmd`, which allows you to create, modify, and control the logical domains.

▼ Enable the Logical Domains Manager Daemon

1. **Use the `svcadm`(1M) command to enable the Logical Domains Manager daemon, `ldmd`.**

```
# svcadm enable ldmd
```

2. **Use the `ldm list` command to verify that the Logical Domains Manager is running.**

   You receive a message similar to the following, which is for the `factory-default` configuration. Note that the `primary` domain is `active`, which means that the Logical Domains Manager is running.

```
# /opt/SUNWldm/bin/ldm list
NAME             STATE    FLAGS   CONS    VCPU  MEMORY   UTIL  UPTIME
primary          active   ---c-   SP      32    3264M    0.3%  19d 9m
```

Creating Authorization and Profiles and Assigning Roles for User Accounts

You set up authorization and profiles and assign roles for user accounts using the Solaris OS Role-Based Access Control (RBAC) adapted for the Logical Domains Manager. Refer to the Solaris 10 System Administrator Collection for more information about RBAC.

Authorization for the Logical Domains Manager has two levels:

- Read – allows you to view, but not modify the configuration.
- Read and write – allows you to view and change the configuration.

Following are the Logical Domains entries automatically added to the Solaris OS `/etc/security/auth_attr` file:

- `solaris.ldoms.:::LDom administration::`
- `solaris.ldoms.grant:::Delegate LDom configuration::`
- `solaris.ldoms.read:::View LDom configuration::`
- solaris.ldoms.write:::Manage LDom configuration::

Managing User Authorizations

▼ Add an Authorization for a User

Use the following steps as necessary to add authorizations in the `/etc/security/auth_attr` file for Logical Domains Manager users. Because the superuser already has `solaris.*` authorization, the superuser already has permission for `solaris.ldoms.*` authorizations.

1. **Create a local user account for each user who needs authorization to use the `ldm`(1M) subcommands.**

Note – To add Logical Domains Manager authorization for a user, a local (non-LDAP) account must be created for that user. Refer to the Solaris 10 System Administrator Collection for details.

---

2. **Do one of the following depending on which `ldm`(1M) subcommands you want the user to be able to access.**

   See TABLE 2-1 for a list of `ldm`(1M) commands and their user authorizations.

   - Add a read-only authorization for a user using the `usermod`(1M) command.

```
# usermod -A solaris.ldoms.read username
```

   - Add a read and write authorization for a user using the `usermod`(1M) command.

```
# usermod -A solaris.ldoms.write username
```

▼ Delete All Authorizations for a User

- **Delete all authorizations for a local user account (the only possible option).**

```
# usermod -A '' username
```

Managing User Profiles

The `SUNWldm` package adds two system-defined RBAC profiles in the `/etc/security/prof_attr` file for use in authorizing access to the Logical Domains Manager by non-superusers. The two LDoms-specific profiles are:

- `LDoms Review:::Review LDoms configuration:auths=solaris.ldoms.read`
- `LDoms Management:::Manage LDoms domains:auths=solaris.ldoms.*`

One of the preceding profiles can be assigned to a user account using the following procedure.

▼ Add a Profile for a User

- **Add an administrative profile for a local user account; for example, LDoms Management.**

```
# usermod -P "LDoms Management" username
```

▼ Delete All Profiles for a User

- **Delete all profiles for a local user account (the only possible option).**

```
# usermod -P '' username
```

Assigning Roles to Users

The advantage of using this procedure is that only a user who has been assigned a specific role can assume the role. In assuming a role, a password is required if the role is given a password. This provides two layers of security. If a user has not been assigned a role, then the user cannot assume the role (by doing the `su` *role_name* command) even if the user has the correct password.

▼ Create a Role and Assign the Role to a User

1. Create a role.

```
# roleadd -A solaris.ldoms.read ldm_read
```

2. Assign a password to the role.

```
# passwd ldm_read
```

3. Assign the role to a user; for example, `user_1`**.**

```
# useradd -R ldm_read user_1
```

4. **Assign a password to the user (`user_1`).**

```
# passwd user_1
```

5. **Assign access only to the `user_1` account to become the `ldm_read` account.**

```
# su user_1
```

6. **Type the user password when or if prompted.**

7. **Verify the user ID and access to the `ldm_read` role.**

```
$ id
uid=nn(user_1) gid=nn(<group name>)
$ roles
ldm_read
```

8. **Provide access to the user for `ldm` subcommands that have read authorization.**

```
# su ldm_read
```

9. **Type the user password when or if prompted.**

10. **Type the `id` command to show the user.**

```
$ id
uid=nn(ldm_read) gid=nn(<group name>)
```

Factory Default Configuration and Disabling Logical Domains

The initial configuration where the platform appears as a single system hosting only one operating system is called the factory default configuration. If you want to disable logical domains, you probably also want to restore this configuration so that the system regains access to all resources (CPUs, memory, I/O), which might have been assigned to other domains.

This section describes how to remove all guest domains, remove all Logical Domains configurations, and revert the configuration to the factory default.

▼ Remove All Guest Logical Domains

1. **List all the logical domain configurations on the system controller.**

```
primary# ldm ls-config
```

2. **Remove all configurations (*config_name*) previously saved to the system controller (SC). Use the following command for each such configuration.**

```
primary# ldm rm-config config_name
```

   Once you remove all the configurations previously saved to the SC, the `factory-default` domain would be the next one to use when the control domain (`primary`) is rebooted.

3. **Stop all guest domains using the `-a` option.**

```
primary# ldm stop-domain -a
```

4. **Unbind all guest domains.**

```
primary# ldm unbind-domain ldom
```

Note – You might not be able to unbind an I/O domain in a split-PCI configuration if it is providing services required by the control domain. In this situation, skip this step.

▼ Restore the Factory Default Configuration

1. **Select the factory default configuration.**

```
primary# ldm set-config factory-default
```

2. **Stop the control domain.**

```
primary# shutdown -i1 -g0 -y
```

3. **Power cycle the system controller so that the `factory-default` configuration is reloaded.**

```
sc> poweroff
sc> poweron
```

▼ Disable the Logical Domains Manager

- **Disable the Logical Domains Manager from the control domain.**

```
primary# svcadm disable ldmd
```

Note – Disabling the Logical Domains Manager does not stop any running domains, but does disable the ability to create a new domains, change the configuration of existing domains, or monitor the state of the domains.

Caution – If you disable the Logical Domains Manager, this disables some services, such as error reporting or power management. In the case of error reporting, if you are in the `factory-default` configuration, you can reboot the sole domain to restore error reporting. However, this is not the case with power management. In addition, some system management or monitoring tools rely on the Logical Domains Manager.

▼ Removing the Logical Domains Manager

After restoring the factory default configuration and disabling the Logical Domains Manager, you can remove the Logical Domains Manager software.

- **Remove the Logical Domains Manager software.**

```
primary# pkgrm SUNWldm
```

Note – If you remove the Logical Domains Manager before restoring the factory default configuration, you can restore the factory default configuration from the system controller as shown in the following procedure.

▼ Restore the Factory Default Configuration From the System Controller

If you remove the Logical Domains Manager before restoring the factory default configuration, you can restore the factory default configuration from the system controller.

1. **Restore the factory default configuration from the system controller.**

```
sc> bootmode config=factory-default
```

2. **Power cycle the system to load the factory default configuration.**

CHAPTER 4

Setting Up Services and Logical Domains

This chapter describes the procedures necessary to set up default services, your control domain, and guest domains.

Output Messages

You receive different output messages from the commands you use to create default services and to set up the control (`primary`) domain depending on your platform:

- Sun UltraSPARC T1 processors
- Sun UltraSPARC T2 and T2 Plus processors

Sun UltraSPARC T1 Processors

You receive the following notice after the setup commands for the `primary` domain if you are using a server with a Sun UltraSPARC T1 processor:

```
Notice: the LDom Manager is running in configuration mode. Any
configuration changes made will only take effect after the machine
configuration is downloaded to the system controller and the host
is reset.
```

Sun UltraSPARC T2 and T2 Plus Processors

You receive the following message after the first operation that cannot be performed dynamically on any device or for any service on the `primary` domain if you are using a server with a Sun UltraSPARC T2 or T2 Plus processor:

```
Initiating delayed reconfigure operation on LDom primary. All
configuration changes for other LDoms are disabled until the
LDom reboots, at which time the new configuration for LDom
primary will also take effect.
```

You receive the following notice after every subsequent operation on the `primary` domain until reboot if you are using a server with a Sun UltraSPARC T2 or T2 Plus processor:

```
Notice: LDom primary is in the process of a delayed
reconfiguration. Any changes made to this LDom will only take
effect after it reboots.
```

Creating Default Services

You must create the following virtual default services initially to be able to use them later:

- `vdiskserver` – virtual disk server
- `vswitch` – virtual switch service
- `vconscon` – virtual console concentrator service

▼ Create Default Services

1. **Create a virtual disk server (`vds`) to allow importing virtual disks into a logical domain.**

   For example, the following command adds a virtual disk server (`primary-vds0`) to the control domain (`primary`).

```
primary$ ldm add-vds primary-vds0 primary
```

2. **Create a virtual console concentrator (`vcc`) service for use by the virtual network terminal server daemon (`vntsd`) and as a concentrator for all logical domain consoles.**

   For example, the following command would add a virtual console concentrator service (`primary-vcc0`) with a port range from 5000 to 5100 to the control domain (`primary`).

```
primary$ ldm add-vcc port-range=5000-5100 primary-vcc0 primary
```

3. **Create a virtual switch service (`vsw`) to enable networking between virtual network (`vnet`) devices in logical domains. Assign a GLDv3-compliant network adapter to the virtual switch if each of the logical domains needs to communicate outside the box through the virtual switch.**

   For example, the following command would add a virtual switch service (`primary-vsw0`) on network adapter driver `e1000g0` to the control domain (`primary`).

```
primary$ ldm add-vsw net-dev=e1000g0 primary-vsw0 primary
```

   This command automatically allocates a MAC address to the virtual switch. You can specify your own MAC address as an option to the `ldm add-vsw` command. However, in that case, it is your responsibility to ensure that the MAC address specified does not conflict with an already existing MAC address.

   If the virtual switch being added replaces the underlying physical adapter as the primary network interface, it must be assigned the MAC address of the physical adapter, so that the Dynamic Host Configuration Protocol (DHCP) server assigns the domain the same IP address. See "Enabling Networking Between the Control/Service Domain and Other Domains" on page 50.

```
primary$ ldm add-vsw mac-addr=2:04:4f:fb:9f:0d net-dev=e1000g0 primary-vsw0
primary
```

4. **Verify the services have been created by using the `list-services` subcommand. Your output should look similar to the following.**

```
primary$ ldm list-services primary
VDS
    NAME             VOLUME         OPTIONS          DEVICE
    primary-vds0

VCC
    NAME             PORT-RANGE
    primary-vcc0     5000-5100

VSW
    NAME             MAC               NET-DEV   DEVICE     MODE
    primary-vsw0     02:04:4f:fb:9f:0d e1000g0   switch@0   prog,promisc
```

Initial Configuration of the Control Domain

Initially, all system resources are allocated to the control domain. To allow the creation of other logical domains, you must release some of these resources.

▼ Set Up the Control Domain

Note – This procedure contains examples of resources to set for your control domain. These numbers are examples only, and the values used might not be appropriate for your control domain.

---

1. **Assign cryptographic resources to the control domain.**

Note – If you have any cryptographic devices in the control domain, you cannot dynamically reconfigure CPUs. So if you are not using cryptographic devices, `set-mau` to `0`.

The following example would assign one cryptographic resource to the control domain, `primary`. This leaves the remainder of the cryptographic resources available to a guest domain.

```
primary$ ldm set-mau 1 primary
```

2. **Assign virtual CPUs to the control domain.**

   For example, the following command would assign 4 virtual CPUs to the control domain, `primary`. This leaves the remainder of the virtual CPUs available to a guest domain.

```
primary$ ldm set-vcpu 4 primary
```

3. **Assign memory to the control domain.**

   For example, the following command would assign 4 gigabyte of memory to the control domain, `primary`. This leaves the remainder of the memory available to a guest domain.

```
primary$ ldm set-memory 4G primary
```

4. **Add a logical domain machine configuration to the system controller (SC).**

   For example, the following command would add a configuration called `initial`.

```
primary$ ldm add-config initial
```

5. **Verify that the configuration is ready to be used at the next reboot.**

```
primary$ ldm list-config
factory-default
initial [next poweron]
```

   This list subcommand shows the `initial` configuration set will be used once you power cycle.

Rebooting to Use Logical Domains

You must reboot the control/service domain for the configuration changes to take effect and the resources to be released for other logical domains to use.

▼ Reboot

- **Shut down and reboot the `primary` domain, which is also the service domain in our examples.**

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

Note – Either a reboot or power cycle instantiates the new configuration. Only a power cycle actually boots the configuration saved to the service processor (SP), which is then reflected in the `list-config` output.

Enabling Networking Between the Control/Service Domain and Other Domains

By default, networking between the control/service domain and other domains in the system is disabled. To enable this, the virtual switch device should be configured as a network device. The virtual switch can either replace the underlying physical device (`e1000g0` in this example) as the primary interface or be configured as an additional network interface in the domain.

Note – Perform the following configuration steps from the domain's console, as the procedure could temporarily disrupt network connectivity to the domain.

▼ Configure the Virtual Switch as the Primary Interface

1. **Print out the addressing information for all interfaces.**

```
primary# ifconfig -a
```

2. **Plumb the virtual switch. In this example,** `vsw0` **is the virtual switch being configured.**

```
primary# ifconfig vsw0 plumb
```

3. **(*Optional*) To obtain the list of all virtual switch instances in a domain, you can list them.**

```
primary# /usr/sbin/dladm show-link | grep vsw
vsw0            type: non-vlan  mtu: 1500       device: vsw0
```

4. **Unplumb the physical network device assigned to the virtual switch (`net-dev`), which is `e1000g0` in this example.**

```
primary# ifconfig e1000g0 down unplumb
```

5. **To migrate properties of the physical network device (`e1000g0`) to the virtual switch (`vsw0`) device, do one of the following:**
   - If networking is configured using a static IP address, reuse the IP address and netmask of e1000g0 for `vsw0`.

```
primary# ifconfig vsw0 IP_of_e1000g0 netmask netmask_of_e1000g0 broadcast + up
```

   - If networking is configured using DHCP, enable DHCP for `vsw0`.

```
primary# ifconfig vsw0 dhcp start
```

6. **Make the required configuration file modifications to make this change permanent.**

```
primary# mv /etc/hostname.e1000g0 /etc/hostname.vsw0
primary# mv /etc/dhcp.e1000g0 /etc/dhcp.vsw0
```

Note – If necessary, you can also configure the virtual switch as well as the physical network device. In this case, plumb the virtual switch as in Step 2, and do not unplumb the physical device (skip Step 4). You must then configure the virtual switch with either a static IP address or a dynamic IP address. You can obtain a dynamic IP address from a DHCP server. For additional information and an example of this case, see "Configuring Virtual Switch and Service Domain for NAT and Routing" on page 117.

Enabling the Virtual Network Terminal Server Daemon

You must enable the virtual network terminal server daemon (`vntsd`) to provide access to the virtual console of each logical domain. Refer to the Solaris 10 OS Reference Manual collection or the `vntsd`(1M) man page for information about how to use this daemon.

▼ Enable the Virtual Network Terminal Server Daemon

Note – Be sure you have created the default service `vconscon` on the control domain before you enable `vntsd`. See "Creating Default Services" on page 46 for more information.

---

1. **Use the `svcadm`(1M) command to enable the virtual network terminal server daemon, `vntsd`(1M).**

```
# svcadm enable vntsd
```

2. **Use the `svcs`(1) command to verify that the `vntsd` is enabled.**

```
# svcs -l vntsd
fmri         svc:/ldoms/vntsd:default
enabled      true
state        online
next_state   none
state_time   Sat Jan 27 03:14:17 2007
logfile      /var/svc/log/ldoms-vntsd:default.log
restarter    svc:/system/svc/restarter:default
contract_id  93
dependency   optional_all/error svc:/milestone/network (online)
dependency   optional_all/none svc:/system/system-log (online)
```

Creating and Starting a Guest Domain

The guest domain must run an operating system that understands both the `sun4v` platform and the virtual devices presented by the hypervisor. Currently, this is the Solaris 10 11/06 OS at a minimum. Refer to the *Logical Domains (LDoms) 1.1 Release Notes* for any specific patches that might be necessary. Once you have created default services and reallocated resources from the control domain, you can create and start a guest domain.

▼ Create and Start a Guest Domain

1. **Create a logical domain.**

   For example, the following command would create a guest domain named `ldg1`.

```
primary$ ldm add-domain ldg1
```

2. **Add CPUs to the guest domain.**

   For example, the following command would add four virtual CPUs to guest domain `ldg1`.

```
primary$ ldm add-vcpu 4 ldg1
```

3. **Add memory to the guest domain.**

   For example, the following command would add 2 gigabytes of memory to guest domain `ldg1`.

```
primary$ ldm add-memory 2G ldg1
```

4. **Add a virtual network device to the guest domain.**

   For example, the following command would add a virtual network device with these specifics to the guest domain `ldg1`.

```
primary$ ldm add-vnet vnet1 primary-vsw0 ldg1
```

   Where:

   - `vnet1` is a unique interface name to the logical domain, assigned to this virtual network device instance for reference on subsequent `set-vnet` or `remove-vnet` subcommands.

- `primary-vsw0` is the name of an existing network service (virtual switch) to which to connect.

Note – Steps 5 and 6 are simplified instructions for adding a virtual disk server device (`vdsdev`) to the primary domain and a virtual disk (`vdisk`) to the guest domain. To learn how ZFS volumes and file systems can be used as virtual disks, see "Export a ZFS Volume as a Single Slice Disk" on page 87 and "Using ZFS With Virtual Disks" on page 97.

---

5. **Specify the device to be exported by the virtual disk server as a virtual disk to the guest domain.**

   You can export a physical disk, disk slice, volumes, or file as a block device. The following examples show a physical disk and a file.

   - **Physical Disk Example.** The first example adds a physical disk with these specifics.

```
primary$ ldm add-vdsdev /dev/dsk/c0t0d0s2 vol1@primary-vds0
```

   Where:

   - `/dev/dsk/c0t0d0s2` is the path name of the actual physical device. When adding a device, the path name must be paired with the device name.
   - `vol1` is a unique name you must specify for the device being added to the virtual disk server. The volume name must be unique to this virtual disk server instance, because this name is exported by this virtual disk server to the clients for adding. When adding a device, the volume name must be paired with the path name of the actual device.
   - `primary-vds0` is the name of the virtual disk server to which to add this device.

   - **File Example.** This second example is exporting a file as a block device.

```
primary$ ldm add-vdsdev backend vol1@primary-vds0
```

   Where:

   - *backend* is the path name of the actual file exported as a block device. When adding a device, the backend must be paired with the device name.
   - `vol1` is a unique name you must specify for the device being added to the virtual disk server. The volume name must be unique to this virtual disk server instance, because this name is exported by this virtual disk server to the clients for adding. When adding a device, the volume name must be paired with the path name of the actual device.
   - `primary-vds0` is the name of the virtual disk server to which to add this device.

6. **Add a virtual disk to the guest domain.**

   The following example adds a virtual disk to the guest domain `ldg1`.

```
primary$ ldm add-vdisk vdisk1 vol1@primary-vds0 ldg1
```

   Where:

   - `vdisk1` is the name of the virtual disk.
   - `vol1` is the name of the existing volume to which to connect.
   - `primary-vds0` is the name of the existing virtual disk server to which to connect.

Note – The virtual disks are generic block devices that are backed by different types of physical devices, volumes, or files. A virtual disk is not synonymous with a SCSI disk and, therefore, excludes the target ID in the disk label. Virtual disks in a logical domain have the following format: c*N*d*N*s*N*, where c*N* is the virtual controller, d*N* is the virtual disk number, and s*N* is the slice.

---

7. **Set `auto-boot` and `boot-device` variables for the guest domain.**

   The first example command sets `auto-boot\?` to `true` for guest domain `ldg1`.

```
primary$ ldm set-var auto-boot\?=true ldg1
```

   The second example command sets `boot-device` to `vdisk` for the guest domain `ldg1`.

```
primary$ ldm set-var boot-device=vdisk ldg1
```

8. **Bind resources to the guest domain `ldg1` and then list the domain to verify that it is bound.**

```
primary$ ldm bind-domain ldg1
primary$ ldm list-domain ldg1
NAME          STATE    FLAGS  CONS    VCPU MEMORY   UTIL  UPTIME
ldg1          bound    -----  5001    4    2G
```

9. **To find the console port of the guest domain, you can look at the output of the preceding `list-domain` subcommand.**

   You can see under the heading `Cons` that logical domain guest 1 (`ldg1`) has its console output bound to port `5001`.

10. **Connect to the console of a guest domain from another terminal by logging into the control domain and connecting directly to the console port on the local host.**

```
$ ssh admin@controldom.domain
$ telnet localhost 5001
```

11. **Start the guest domain** `ldg1`**.**

```
primary$ ldm start-domain ldg1
```

Installing Solaris OS on a Guest Domain

This section provides instructions for several different ways you can install the Solaris OS on a guest domain.

▼ Install Solaris OS on a Guest Domain From a DVD

1. **Insert the Solaris 10 OS DVD into the DVD drive of the Sun SPARC Enterprise T5220 system, for example.**
2. **Stop the volume management daemon,** `vold`**(1M) on the** `primary` **domain.**

```
primary# svcadm disable volfs
```

3. **Confirm on the** `primary` **domain that the DVD disk is mounted successfully by** `vold`**(1M).**
4. **Stop and unbind the guest domain (**`ldg1`**). Then add the DVD with DVDROM media as a secondary volume (**`dvd_vol@primary-vds0`**) and virtual disk (**`vdisk_cd_media`**), for example.**

   `c0t0d0s2` is where the Solaris OS media resides

```
primary# ldm stop ldg1
primary# ldm unbind ldg1
primary# ldm add-vdsdev /dev/dsk/c0t0d0s2 dvd_vol@primary-vds0
primary# ldm add-vdisk vdisk_cd_media dvd_vol@primary-vds0 ldg1
```

5. **Check to see that the DVD is added as a secondary volume and virtual disk.**

   TOUT in the lists for virtual disks means the timeout set for the disk when added, if any. In this example, the virtual disk vdisk_cd_media would wait 60 seconds before timing out and sending an error message while trying to connect to the virtual disk server.

```
primary# ldm list-bindings
NAME             STATE    FLAGS   CONS    VCPU  MEMORY   UTIL  UPTIME
primary          active   -n-cv   SP      4     4G       0.2%  22h 45m
...
VDS
   NAME             VOLUME         OPTIONS          DEVICE
   primary-vds0     vol1                            /dev/dsk/c1t1d0s2
   iso_vol                                          /export/solarisdvd.iso
   dvd_vol                                          /dev/dsk/c0t0d0s2
   install_vol                                      /export/install_disk
....
------------------------------------------------------------------------------
NAME             STATE    FLAGS   CONS    VCPU  MEMORY   UTIL  UPTIME
ldg1             inactive -----           60    6G
...
DISK
   NAME             VOLUME                         TOUT DEVICE  SERVER
   vdisk1           vol1@primary-vds0
   vdisk_iso        iso_vol@primary-vds0
   vdisk_cd_media   dvd_vol@primary-vds0           60
   vdisk_install    install_vol@primary-vds0
....
```

6. **Create and add another disk (**install_disk**) on which to install the Solaris OS.**

   This is an example of using a disk space on an existing file system to install the OS. You can also use a physical disk to install the OS if it has already been defined for the guest domain.

```
primary# mkfile -n 20g /export/install_disk
primary# ldm add-vdsdev /export/install_disk install_vol@primary-vds0
primary# ldm add-vdisk vdisk_install install_vol@primary-vds0 ldg1
```

7. **Bind and start the guest domain (`ldg1`).**

```
primary# ldm bind ldg1
primary# ldm start ldg1
LDom ldg1 started
primary# telnet localhost 5000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

Connecting to console "ldg1" in group "ldg1" ....
Press ~? for control options ..
```

8. **Create the device aliases that you need. In this example, an alias for disk 2 is created.**

```
ok show-disks
a) /virtual-devices@100/channel-devices@200/disk@3
b) /virtual-devices@100/channel-devices@200/disk@2
c) /virtual-devices@100/channel-devices@200/disk@1
d) /virtual-devices@100/channel-devices@200/disk@0
q) NO SELECTION
Enter Selection, q to quit: b
```

9. **Show the device aliases in the client OpenBoot PROM.**

   In this example, see the device aliases for `vdisk_cd_media`, which is the Solaris DVD, and `vdisk_install`, which is the disk space.

```
ok devalias
vdisk_install   /virtual-devices@100/channel-devices@200/disk@3
vdisk_cd_media  /virtual-devices@100/channel-devices@200/disk@2
vdisk_iso       /virtual-devices@100/channel-devices@200/disk@1
vdisk1          /virtual-devices@100/channel-devices@200/disk@0
vnet1           /virtual-devices@100/channel-devices@200/network@0
net             /virtual-devices@100/channel-devices@200/network@0
disk            /virtual-devices@100/channel-devices@200/disk@0
virtual-console /virtual-devices/console@1
name            aliases
```

10. **On the guest domain's console, boot from `disk@2` on slice `f`.**

```
ok boot /virtual-devices@100/channel0-devices@200/disk@2:f -v
Boot device: /virtual-devices@100/channel-devices@200/disk@2:f  File and args:
-s
SunOS Release 5.10 Version Generic_137137-09 32-bit
Copyright 1983-2008 Sun Microsystems, Inc.  All rights reserved.
Use is subject to license terms.
```

11. **Continue with the Solaris OS installation menu.**

▼ Install Solaris OS on a Guest Domain From a Solaris ISO File

1. **Unbind the guest domain (`ldg1`).**

```
primary# ldm unbind ldg1
```

2. **Add the Solaris ISO file (`solarisdvd.iso`) as a secondary volume (`iso_vol@primary-vds0`) and virtual disk (`vdisk_iso`), for example.**

```
primary# ldm add-vdsdev /export/solarisdvd.iso  iso_vol@primary-vds0
primary# vdisk vdisk_iso iso_vol@primary-vds0 ldg1
```

3. **Check to see that the Solaris ISO file is added as a secondary volume and virtual disk.**

   `TOUT` in the lists for virtual disks means the timeout set for the disk when added, if any. There is no timeout period specified for the virtual disk `vdisk_iso`.

```
primary# ldm list-bindings
NAME             STATE    FLAGS   CONS    VCPU  MEMORY   UTIL  UPTIME
primary          active   -n-cv   SP      4     4G       0.2%  22h 45m
...
VDS
   NAME             VOLUME         OPTIONS          DEVICE
   primary-vds0     vol1                            /dev/dsk/c1t1d0s2
   iso_vol                                          /export/solarisdvd.iso
   dvd_vol                                          /dev/dsk/c0t0d0s2
   install_vol                                      /export/install_disk
....
------------------------------------------------------------------------------
NAME             STATE    FLAGS   CONS    VCPU  MEMORY   UTIL  UPTIME
ldg1             inactive -----           60    6G
...
DISK
   NAME             VOLUME                         TOUT DEVICE  SERVER
   vdisk1           vol1@primary-vds0
   vdisk_iso        iso_vol@primary-vds0
   vdisk_cd_media   dvd_vol@primary-vds0           60
   vdisk_install    install_vol@primary-vds0
....
```

4. **Bind and start the guest domain (`ldg1`).**

```
primary# ldm bind ldg1
primary# ldm start ldg1
LDom ldg1 started
primary# telnet localhost 5000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

Connecting to console "ldg1" in group "ldg1" ....
Press ~? for control options ..
```

5. **Show the device aliases in the client OpenBoot PROM.**

   In this example, see the device aliases for `vdisk_iso`, which is the Solaris ISO image, and `vdisk_install`, which is the disk space.

```
ok devalias
vdisk_install   /virtual-devices@100/channel-devices@200/disk@3
vdisk_cd_media  /virtual-devices@100/channel-devices@200/disk@2
vdisk_iso       /virtual-devices@100/channel-devices@200/disk@1
vdisk1          /virtual-devices@100/channel-devices@200/disk@0
vnet1           /virtual-devices@100/channel-devices@200/network@0
net             /virtual-devices@100/channel-devices@200/network@0
disk            /virtual-devices@100/channel-devices@200/disk@0
virtual-console /virtual-devices/console@1
name            aliases
```

6. **On the guest domain's console, boot from `disk@2` on slice `f`.**

```
ok boot /virtual-devices@100/channel0-devices@200/disk@1:f -v
Boot device: /virtual-devices@100/channel-devices@200/disk@1:f  File and args:
-s
SunOS Release 5.10 Version Generic_137137-09 32-bit
Copyright 1983-2008 Sun Microsystems, Inc.  All rights reserved.
Use is subject to license terms.
```

7. **Continue with the Solaris OS installation menu.**

▼ Jump-Start a Guest Domain

- **To jump-start a guest domain, use a normal JumpStart procedure with the following profile syntax changes from a regular Solaris OS JumpStart procedure to a JumpStart procedure specific to LDoms as shown in the following two examples.**

  Normal JumpStart Profile

```
filesys c1t1d0s0 free /
filesys c1t1d0s1 2048 swap
filesys c1t1d0s5 120 /spare1
filesys c1t1d0s6 120 /spare2
```

  Virtual disk device names in a logical domain differ from physical disk device names in that they do not contain a target ID (`t`*N*) in the device name. Instead of the normal `c`*N*`t`*N*`d`*N*`s`*N* format, virtual disk device names are of the format `c`*N*`d`*N*`s`*N*, where `c`*N* is the virtual controller, `d`*N* is the virtual disk number, and `s`*N* is the slice. Modify your JumpStart profile to reflect this change as in the following profile example.

  Actual Profile Used for a Logical Domain

```
filesys c0d0s0 free /
filesys c0d0s1 2048 swap
filesys c0d0s5 120 /spare1
filesys c0d0s6 120 /spare2
```

Note – You must use the MAC address of the virtual network (`vnet`) device as reported by the `ldm`(1M) command for your jumpstart configuration and not the one reported in the banner for the guest.

Saving Logical Domain Configurations for Future Rebuilding

The basic process is to save the constraints information for each domain into an XML file, which can then be re-issued to the Logical Domains Manager, for example, after a hardware failure to rebuild a desired configuration.

"Rebuild Guest Domain Configurations" on page 62 works for guest domains, not the control domain. You can save the control (primary) domain's constraints to an XML file, but you cannot feed it back into the ldm add-domain -i command. However, you can use the resource constraints from the XML file to create the CLI commands to reconfigure your primary domain. See "Rebuilding the Control Domain" on page 63 for instructions on how to translate typical XML output from an ldm list-constraints -x primary command into the CLI commands needed to reconfigure a primary domain.

The method that follows does not preserve actual bindings, only the constraints used to create those bindings. This means that, after this procedure, the domains will have the same virtual resources, but will not necessarily be bound to the same physical resources.

▼ Save All Logical Domain Configurations

- **For each logical domain, create an XML file containing the domain's constraints.**

```
# ldm ls-constraints -x ldom > ldom.xml
```

▼ Rebuild Guest Domain Configurations

- **Run the following commands for each guest domain's XML file you created.**

```
# ldm add-domain -i ldom.xml
# ldm bind-domain ldom
# ldm start-domain ldom
```

Rebuilding the Control Domain

This section provides instructions for how to translate typical XML output from an `ldm list-constraints -x primary` command into the CLI commands needed to reconfigure a primary domain. The resources and properties that you use to translate XML into CLI commands are shown in **bold** type in the sample XML output. Refer to the `ldm` man page or the *Logical Domains (LDoms) Manager 1.1 Man Page Guide* for complete information about the CLI commands.

A sample output from a `ldm list-constraints -x primary` command follows.

CODE EXAMPLE 4-1 Sample XML Output From `list-constraints` Subcommand

```
<?xml version="1.0"?>
<LDM_interface version="1.1" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="./schemas/combined-v3.xsd"
        xmlns:ovf="./schemas/envelope"
        xmlns:rasd="./schemas/CIM_ResourceAllocationSettingData"
        xmlns:vssd="./schemas/CIM_VirtualSystemSettingData"
        xmlns:gprop="./schemas/GenericProperty" xmlns:bind=
"./schemas/Binding">
  <data version="3.0">
    <Envelope>
      <References/>
      <Content xsi:type="ovf:VirtualSystem_Type" ovf:id="primary">
        <Section xsi:type="ovf:ResourceAllocationSection_Type">
          <Item>
            <rasd:OtherResourceType>ldom_info</rasd:OtherResourceType>
            <rasd:Address>00:03:ba:d8:ba:f6</rasd:Address>
            <gprop:GenericProperty key=
"hostid">0x83d8baf6</gprop:GenericProperty>
          </Item>
        </Section>
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>cpu</rasd:OtherResourceType>
            <rasd:AllocationUnits>4</rasd:AllocationUnits>
          </Item>
        </Section>
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>mau</rasd:OtherResourceType>
            <rasd:AllocationUnits>1</rasd:AllocationUnits>
          </Item>
        </Section>
```

CODE EXAMPLE 4-1 Sample XML Output From `list-constraints` Subcommand *(Continued)*

```
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>memory</rasd:OtherResourceType>
            <rasd:AllocationUnits>4G</rasd:AllocationUnits>
          </Item>
        </Section>
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>physio_device</rasd:OtherResourceType>
            <gprop:GenericProperty key="name">pci@7c0</gprop:GenericProperty>
          </Item>
        </Section>
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>vsw</rasd:OtherResourceType>
            <rasd:Address>auto-allocated</rasd:Address>
            <gprop:GenericProperty key=
"service_name">primary-vsw0</gprop:GenericProperty>
            <gprop:GenericProperty key=
"dev_path">e1000g0</gprop:GenericProperty>
            <gprop:GenericProperty key=
"default-vlan-id">1</gprop:GenericProperty>
            <gprop:GenericProperty key="pvid">1</gprop:GenericProperty>
          </Item>
        </Section>
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>vcc</rasd:OtherResourceType>
            <gprop:GenericProperty key=
"service_name">primary-vcc0</gprop:GenericProperty>
            <gprop:GenericProperty key="min_port">5000</gprop:GenericProperty>
            <gprop:GenericProperty key="max_port">6000</gprop:GenericProperty>
          </Item>
        </Section>
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>vds</rasd:OtherResourceType>
            <gprop:GenericProperty key=
"service_name">primary-vds0</gprop:GenericProperty>
          </Item>
        </Section>
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>vds_volume</rasd:OtherResourceType>
            <gprop:GenericProperty key=
"vol_name">primary-vds0-vol0</gprop:GenericProperty>
```

CODE EXAMPLE 4-1 Sample XML Output From `list-constraints` Subcommand *(Continued)*

```
            <gprop:GenericProperty key"block_dev">
/opt/SUNWldm/domain_disks/testdisk.nv.53.1</gprop:GenericProperty>
            <gprop:GenericProperty key=
"service_name">primary-vds0</gprop:GenericProperty>
          </Item>
        </Section>
      </Content>
    </Envelope>
  </data>
</LDM_interface>
```

The <Content> tag and the <Section>s inside the <Content> tag describe the primary domain and all the resources contained in the primary domain. The <rasd:...> and <gprop:GenericProperty...> tags within <Item>s describe the properties needed for each resource. You can go through each resource in each <Section> and construct CLI commands based on the resource's constraints. The following sections identify some of the more common resources in the domain XML description and the equivalent CLI command for that resource.

Logical Domain Information (ldom_info) Section

This section describes the primary domain's MAC address and host ID information. Because this is the primary domain, you cannot set this information; it is automatically set.

CODE EXAMPLE 4-2 LDoms Information (ldom_info) Section

```
        <Section> xsi:type="ovf:ResourceAllocationSection_Type">
          <Item>
            <rasd:OtherResourceType>ldom_info</rasd:OtherResourceType>
            <rasd:Address>00:03:ba:d8:ba:f6</rasd:Address>
            <gprop:GenericProperty key=
"hostid">0x83d8baf6</gprop:GenericProperty>
          </Item>
        </Section>
```

In this example, the logical domain information (ldom_info) is as follows:

- (MAC) Address - 00:03:ba:d8:ba:f6
- hostid - 0x83d8baf6

Cryptographic (mau) Section

This section describes the number of cryptographic units (maus) allocated to the primary domain.

Note – Even though the mau section comes after the cpu section in the XML listing, you must run the set-mau subcommand before the set-cpu subcommand, because you cannot remove CPUs from a domain without also removing their corresponding cryptographic units.

CODE EXAMPLE 4-3 Cryptographic (mau) Section

```
        <Section> xsi:type="ovf:VirtualHardwareSection_Type"
          <Item>
            <rasd:OtherResourceType>mau</rasd:OtherResourceType>
            <rasd:AllocationUnits>1</rasd:AllocationUnits>
          </Item>
        </Section>
```

This section is equivalent to the following CLI command:

```
# ldm set-mau 1 primary
```

CPU (cpu) Section

This section describes the number of virtual cpus allocated to the primary domain.

CODE EXAMPLE 4-4 CPU (cpu) Section

```
        <Section> xsi:type="ovf:VirtualHardwareSection_Type"
          <Item>
            <rasd:OtherResourceType>cpu</rasd:OtherResourceType>
            <rasd:AllocationUnits>4</rasd:AllocationUnits>
          </Item>
        </Section>
```

This section is equivalent to the following CLI command:

```
# ldm set-vcpu 4 primary
```

Memory (memory) Section

This section describes the amount of memory allocated to the primary domain.

CODE EXAMPLE 4-5 Memory (memory) Section

```
<Section> xsi:type="ovf:VirtualHardwareSection_Type"
  <Item>
    <rasd:OtherResourceType>memory</rasd:OtherResourceType>
    <rasd:AllocationUnits>4G</rasd:AllocationUnits>
  </Item>
</Section>
```

This section is equivalent to the following CLI command:

```
# ldm set-memory 4G primary
```

Physical Input/Output (physio_device) Section

This section describes the physical I/O buses that you want to remain in the primary domain.

CODE EXAMPLE 4-6 Physical I/O (physio_device) Section

```
<Section> xsi:type="ovf:VirtualHardwareSection_Type"
  <Item>
    <rasd:OtherResourceType>physio_device</rasd:OtherResourceType>
    <gprop:GenericProperty key="name">pci@7c0</gprop:GenericProperty>
  </Item>
</Section>
```

To set your primary domain with the same I/O devices as previously configured, you first need to list the I/O devices that are configured on startup.

```
# ldm list -l primary
....
IO
    DEVICE          PSEUDONYM        OPTIONS
    pci@7c0         bus_b
    pci@780         bus_a
....
```

In CODE EXAMPLE 4-6, the bus previously configured to remain in the primary domain was pci@7c0. If there are no other physio-device sections in the XML, the pci@780 bus must be removed.

This section is equivalent to the following CLI command:

```
# ldm remove-io pci@780 primary
```

Virtual Switch (vsw) Section

This section describes any virtual switches (vsws) allocated to the primary domain.

```
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>vsw</rasd:OtherResourceType>
            <rasd:Address>auto-allocated</rasd:Address>
            <gprop:GenericProperty key=
"service_name">primary-vsw0</gprop:GenericProperty>
            <gprop:GenericProperty key=
"dev_path">e1000g0</gprop:GenericProperty>
            <gprop:GenericProperty key="mode">sc</gprop:GenericProperty>
            <gprop:GenericProperty key=
"default-vlan-id">1</gprop:GenericProperty>
            <gprop:GenericProperty key="pvid">1</gprop:GenericProperty>
          </Item>
        </Section>
```

Where:

- The <rasd:Address> tag describes the MAC address to be used for the virtual switch. If the value of this tag is auto-allocated, you do not need to supply a MAC address.
- The XML key property service_name is the name of the virtual switch; in this case, primary-vsw0.
- The XML key property dev_path is the path name for the actual network device; in this case net-dev=e1000g.
- The XML key property mode indicates sc for SunCluster heartbeat support.

Some of the values in this section are default values, such as the default-vlan-id (1) and pvid (1), so the section is equivalent to the following CLI command:

```
# ldm add-vswitch net-dev=e1000g primary-vsw0 primary
```

Virtual Console Concentrator (vcc) Section

This section describes any virtual console concentrator (vcc) allocated to the primary domain.

```
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>vcc</rasd:OtherResourceType>
            <gprop:GenericProperty key=
"service_name">primary-vcc0</gprop:GenericProperty>
            <gprop:GenericProperty key="min_port">5000</gprop:GenericProperty>
            <gprop:GenericProperty key="max_port">6000</gprop:GenericProperty>
          </Item>
        </Section>
```

Where:

- The XML key property service_name is the name of the vcc service; in this case, primary-vcc0.

This section is the equivalent of the following CLI command:

```
# ldm add-vcc port-range=5000-6000 primary-vcc0 primary
```

Virtual Disk Server (vds) Section

This section describes any virtual disk server (vds) allocated to the primary domain.

```
         <Section xsi:type="ovf:VirtualHardwareSection_Type">
           <Item>
             <rasd:OtherResourceType>vds</rasd:OtherResourceType>
             <gprop:GenericProperty key=
"service_name">primary-vds0</gprop:GenericProperty>
           </Item>
         </Section>
```

Where:

- The XML key property service_name is the service name for this instance of the virtual disk server; in this case, primary-vds0. The service_name must be unique among all virtual disk server instances on the server.

This section is the equivalent of the following CLI command:

```
# ldm add-vds primary-vds0 primary
```

Virtual Disk Server Device (vdsdev) Section

This section describes any device (vdsdev) exported by the virtual disk server that is allocated to the primary domain.

```
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>vds_volume</rasd:OtherResourceType>
            <gprop:GenericProperty key="vol_name">vdsdev0<
/gprop:GenericProperty>
        <gprop:GenericProperty key="service_name">primary-vds0<
        /gprop:GenericProperty>
            <gprop:GenericProperty key="block_dev">
/opt/SUNWldm/domain_disks/testdisk1</gprop:GenericProperty>
        <gprop:GenericProperty key="vol_opts">ro<
/gprop:GenericProperty>
        <gprop:GenericProperty key="mpgroup">mpgroup-name<
/gprop:GenericProperty>
          </Item>
        </Section>
```

Where:

- The XML key properties volume name (vol_name) and service name (service_name) are paired in the CLI command; in this case, vdsdev0@primary-vds0.
- The XML key property block_dev is the *backend* argument in the equivalent CLI command, which is the location where data of a virtual disk are stored; in this case, /opt/SUNWldm/domain_disks/testdisk1.
- The optional XML key property vol_opts is one or more of the following, comma-separated, within one string: {ro,slice,excl}.
- The optional XML key property mpgroup is the name of the multipath (failover) group.

This section is equivalent to the following CLI command:

```
# ldm add-vdsdev options=ro mpgroup=mpgroup-name
/opt/SUNWldm/domain_disks/testdisk1 vdsdev0@primary-vds0
```

CHAPTER 5

Using PCI Busses With Logical Domains Software

This chapter describes how to configure PCI express busses across multiple logical domains and how to enable the I/O MMU bypass mode on a PCI bus.

Configuring PCI Express Busses Across Multiple Logical Domains

Note – For Sun UltraSPARC T-2 based servers, such as the Sun SPARC Enterprise T5120 and T5220 servers, you would assign a Network Interface Unit (NIU) to the logical domain rather than use this procedure.

The PCI Express (PCIe) bus on a Sun UltraSPARC T1-based server consists of two ports with various leaf devices attached to them. These are identified on a server with the names `pci@780 (bus_a)` and `pci@7c0 (bus_b)`. In a multidomain environment, the PCIe bus can be programmed to assign each leaf to a separate domain using the Logical Domains Manager. Thus, you can enable more than one domain with direct access to physical devices instead of using I/O virtualization.

When the Logical Domains system is powered on, the control (`primary`) domain uses all the physical device resources, so the primary domain owns both the PCIe bus leaves.

Caution – All internal disks on the supported servers are connected to a single leaf. If a control domain is booted from an internal disk, do not remove that leaf from the domain. Also, ensure that you are not removing the leaf with the primary network port. If you remove the wrong leaf from the control or service domain, that domain would not be able to access required devices and would become unusable. If the primary network port is on a different bus than the system disk, then move the network cable to an onboard network port and use the Logical Domains Manager to reconfigure the virtual switch (`vsw`) to reflect this change.

▼ Create a Split PCI Configuration

The example shown here is for a Sun Fire T2000 server. This procedure also can be used on other Sun UltraSPARC T1-based servers, such a Sun Fire T1000 server and a Netra T2000 server. The instructions for different servers might vary slightly from these, but you can obtain the basic principles from the example. Mainly, you need to retain the leaf that has the boot disk and remove the other leaf from the primary domain and assign it to another domain.

1. Verify that the `primary` domain owns both leaves of the PCI Express bus.

```
primary# ldm list-bindings primary
...
IO
    DEVICE          PSEUDONYM        OPTIONS
    pci@780         bus_a
    pci@7c0         bus_b
...
```

2. Determine the device path of the boot disk, which needs to be retained.

```
primary# df /
/              (/dev/dsk/c1t0d0s0 ): 1309384 blocks   457028 files
```

3. Determine the physical device to which the block device `c1t0d0s0` is linked.

```
primary# ls -l /dev/dsk/c1t0d0s0
lrwxrwxrwx   1 root     root          65 Feb  2 17:19 /dev/dsk/c1t0d0s0 -> ../
../devices/pci@7c0/pci@0/pci@1/pci@0,2/LSILogic,sas@2/sd@0,0:a
```

In this example, the physical device for the boot disk for domain `primary` is under the leaf `pci@7c0`, which corresponds to our earlier listing of `bus_b`. This means that we can assign `bus_a` (`pci@780`) of the PCIe bus to another domain.

4. **Check `/etc/path_to_inst` to find the physical path of the onboard network ports.**

```
primary# grep e1000g /etc/path_to_inst
```

5. **Remove the leaf that does not contain the boot disk (`pci@780` in this example) from the `primary` domain.**

```
primary# ldm remove-io pci@780 primary
```

6. **Add this split PCI configuration (`split-cfg` in this example) to the system controller.**

```
primary# ldm add-config split-cfg
```

This configuration (`split-cfg`) is also set as the next configuration to be used after the reboot.

Note – Currently, there is a limit of 8 configurations that can be saved on the SC, not including the `factory-default` configuration.

---

7. **Reboot the `primary` domain so that the change takes effect.**

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

8. **Add the leaf (`pci@780` in this example) to the domain (`ldg1` in this example) that needs direct access.**

```
primary# ldm add-io pci@780 ldg1
Notice: the LDom Manager is running in configuration mode. Any
configuration changes made will only take effect after the machine
configuration is downloaded to the system controller and the
host is reset.
```

If you have an Infiniband card, you might need to enable the bypass mode on the `pci@780` bus. See "Enabling the I/O MMU Bypass Mode on a PCI Bus" on page 75 for information on whether you need to enable the bypass mode.

9. **Reboot domain `ldg1` so that the change takes effect.**

All domains must be inactive for this reboot. If you are configuring this domain for the first time, the domain will be inactive.

```
ldg1# shutdown -i6 -g0 -y
```

10. **Confirm that the correct leaf is still assigned to the `primary` domain and the correct leaf is assigned to domain `ldg1`.**

```
primary# ldm list-bindings primary
NAME             STATE    FLAGS  CONS    VCPU  MEMORY   UTIL  UPTIME
primary          active   -n-cv  SP      4     4G       0.4%  18h 25m
...
IO
    DEVICE           PSEUDONYM        OPTIONS
    pci@7c0          bus_b
...
------------------------------------------------------------------
NAME             STATE    FLAGS  CONS    VCPU  MEMORY   UTIL  UPTIME
ldg1             active   -n---  5000    4     2G       10%   35m
...
IO
    DEVICE           PSEUDONYM        OPTIONS
    pci@780          bus_a
...
```

This output confirms that the PCIe leaf `bus_b` and the devices below it are assigned to domain `primary`, and `bus_a` and its devices are assigned to `ldg1`.

Enabling the I/O MMU Bypass Mode on a PCI Bus

If you have an Infiniband Host Channel Adapter (HCA) card, you might need to turn the I/O memory management unit (MMU) bypass mode on. By default, Logical Domains software controls PCIe transactions so that a given I/O device or PCIe option can only access the physical memory assigned within the I/O domain. Any attempt to access memory of another guest domain is prevented by the I/O MMU. This provides a higher level of security between the I/O domain and all other domains. However, in the rare case where a PCIe or PCI-X option card does not load or operate with the I/O MMU bypass mode off, this option allows you to turn the I/O MMU bypass mode on. However, if you turn the bypass mode on, there no longer is a hardware-enforced protection of memory accesses from the I/O domain.

The `bypass=on` option turns on the I/O MMU bypass mode. This bypass mode should be enabled only if the respective I/O domain and I/O devices within that I/O domain are trusted by all guest domains. This example turns on the bypass mode.

```
primary# ldm add-io bypass=on pci@780 ldg1
```

The output shows `bypass=on` under `OPTIONS`.

CHAPTER 6

Using Virtual Disks With Logical Domains

This chapter describes how to use virtual disks with Logical Domains software.

Introduction to Virtual Disks

A virtual disk contains two components: the virtual disk itself as it appears in a guest domain, and the virtual disk backend, which is where data is stored and where virtual I/O ends up. The virtual disk backend is exported from a service domain by the virtual disk server (`vds`) driver. The `vds` driver communicates with the virtual disk client (`vdc`) driver in the guest domain through the hypervisor using a logical domain channel (LDC). Finally, a virtual disk appears as `/dev/[r]dsk/cXdYsZ` devices in the guest domain.

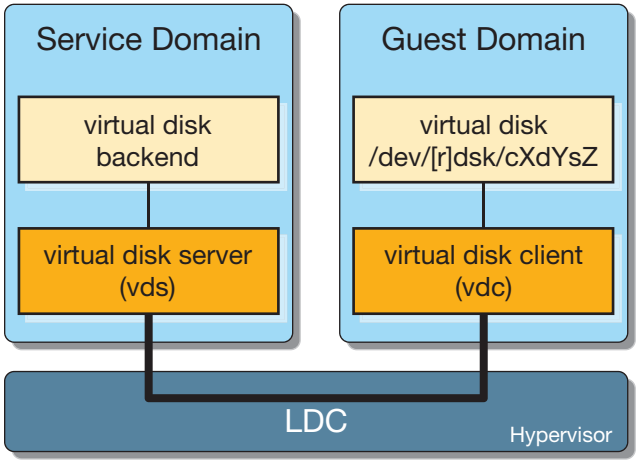
The virtual disk backend can be physical or logical. Physical devices can include the following:

- Physical disk or disk logical unit number (LUN)
- Physical disk slice

Logical devices can be any of the following:

- File on a file system, such as ZFS or UFS
- Logical volume from a volume manager, such as ZFS, VxVM, or Solaris™ Volume Manager (SVM)
- Any disk pseudo device accessible from the service domain

FIGURE 6-1 Virtual Disks With Logical Domains



Managing Virtual Disks

This section describes adding a virtual disk to a guest domain, changing virtual disk and timeout options, and removing a virtual disk from a guest domain. See "Virtual Disk Backend Options" on page 81 for a description of virtual disk options. See "Virtual Disk Timeout" on page 95 for a description of the virtual disk timeout.

▼ Add a Virtual Disk

1. Export the virtual disk backend from a service domain.

```
# ldm add-vdsdev [options={ro,slice,excl}] [mpgroup=mpgroup] backend
volume_name@service_name
```

2. Assign the backend to a guest domain.

```
# ldm add-vdisk [timeout=seconds] disk_name volume_name@service_name ldom
```

Note – A backend is actually exported from the service domain and assigned to the guest domain when the guest domain (*ldom*) is bound.

▼ Export a Virtual Disk Backend Multiple Times

A virtual disk backend can be exported multiple times either through the same or different virtual disk servers. Each exported instance of the virtual disk backend can then be assigned to either the same or different guest domains.

When a virtual disk backend is exported multiple times, it should not be exported with the exclusive (`excl`) option. Specifying the `excl` option will only allow exporting the backend once. The backend can be safely exported multiple times as a read-only device with the `ro` option.

Caution – When a virtual disk backend is exported multiple times, applications running on guest domains and using that virtual disk are responsible for coordinating and synchronizing concurrent write access to ensure data coherency.

The following example describes how to add the same virtual disk to two different guest domains through the same virtual disk service.

1. **Export the virtual disk backend two times from a service domain by using the following commands.**

```
# ldm add-vdsdev [options={ro,slice}] backend volume1@service_name
# ldm add-vdsdev [options={ro,slice}] backend volume2@service_name
```

   The `add-vdsdev` subcommand displays the following warning to indicate that the backend is being exported more than once.

```
Warning: "backend" is already in use by one or more servers in guest
"ldom"
```

2. **Assign the exported backend to each guest domain by using the following commands.**

   The *disk_name* can be different for `ldom1` and `ldom2`.

```
# ldm add-vdisk [timeout=seconds] disk_name volume1@service_name ldom1
# ldm add-vdisk [timeout=seconds] disk_name volume2@service_name ldom2
```

▼ Change Virtual Disk Options

- **After a backend is exported from the service domain, you can change the virtual disk options by using the following command.**

```
# ldm set-vdsdev options=[{ro,slice,excl}] volume_name@service_name
```

▼ Change the Timeout Option

- **After a virtual disk is assigned to a guest domain, you can change the timeout of the virtual disk by using the following command.**

```
# ldm set-vdisk timeout=seconds disk_name ldom
```

▼ Remove a Virtual Disk

1. **Remove a virtual disk from a guest domain by using the following command.**

```
# ldm rm-vdisk disk_name ldom
```

2. **Stop exporting the corresponding backend from the service domain by using the following command.**

```
# ldm rm-vdsdev volume_name@service_name
```

Virtual Disk Appearance

When a backend is exported as a virtual disk, it can appear in the guest domain either as a full disk or as a single slice disk. The way it appears depends on the type of the backend and on the options used to export it.

Full Disk

When a backend is exported to a domain as a full disk, it appears in that domain as a regular disk with 8 slices (`s0` to `s7`). Such a disk is visible with the `format`(1M) command. The disk's partition table can be changed using either the `fmthard`(1M) or `format`(1M) command.

A full disk is also visible to the OS installation software and can be selected as a disk onto which the OS can be installed.

Any backend can be exported as a full disk except physical disk slices that can only be exported as single slice disks.

Single Slice Disk

When a backend is exported to a domain as a single slice disk, it appears in that domain as a regular disk with 8 slices (`s0` to `s7`). However, only the first slice (`s0`) is usable. Such a disk is visible with the `format`(1M) command, but the disk's partition table cannot be changed.

A single slice disk is also visible from the OS installation software and can be selected as a disk onto which you can install the OS. In that case, if you install the OS using the UNIX File System (UFS), then only the root partition (`/`) must be defined, and this partition must use all the disk space.

Any backend can be exported as a single slice disk except physical disks that can only be exported as full disks.

Note – Before the Solaris 10 10/08 OS release, a single slice disk appeared as a disk with a single partition (`s0`). Such a disk was not visible with the `format`(1M) command. The disk also was not visible from the OS installation software and could not be selected as a disk device onto which the OS could be installed.

Virtual Disk Backend Options

Different options can be specified when exporting a virtual disk backend. These options are indicated in the `options=` argument of the `ldm add-vdsdev` command as a comma separated list. The valid options are: `ro`, `slice`, and `excl`.

Read-only (`ro`) Option

The read-only (`ro`) option specifies that the backend is to be exported as a read-only device. In that case, the virtual disk assigned to the guest domain can only be accessed for read operations, and any write operation to the virtual disk will fail.

Exclusive (`excl`) Option

The exclusive (`excl`) option specifies that the backend in the service domain has to be opened exclusively by the virtual disk server when it is exported as a virtual disk to another domain. When a backend is opened exclusively, it is not accessible by

other applications in the service domain. This prevents the applications running in the service domain from inadvertently using a backend that is also being used by a guest domain.

Note – Some drivers do not honor the `excl` option and will disallow some virtual disk backends from being opened exclusively. The `excl` option is known to work with physical disks and slices, but the option does not work with files. It may or may not work with pseudo devices, such as disk volumes. If the driver of the backend does not honor the exclusive open, the backend `excl` option is ignored, and the backend is not opened exclusively.

Because the `excl` option prevents applications running in the service domain from accessing a backend exported to a guest domain, do not set the `excl` option in the following situations:

- When guest domains are running, if you want to be able to use commands such as `format`(1M) or `luxadm`(1M) to manage physical disks, then do not export these disks with the `excl` option.
- When you export an SVM volume, such as a RAID or a mirrored volume, do not set the `excl` option. Otherwise, this can prevent SVM from starting some recovery operation in case a component of the RAID or mirrored volume fails. See "Using Virtual Disks on Top of SVM" on page 103 for more information.
- If the Veritas Volume Manager (VxVM) is installed in the service domain and Veritas Dynamic Multipathing (VxDMP) is enabled for physical disks, then physical disks have to be exported without the (non-default) `excl` option. Otherwise, the export fails, because the virtual disk server (`vds`) is unable to open the physical disk device. See "Using Virtual Disks When VxVM Is Installed" on page 103 for more information.
- If you are exporting the same virtual disk backend multiple times from the same virtual disk service, see "Export a Virtual Disk Backend Multiple Times" on page 79 for more information.

By default, the backend is opened non-exclusively. That way the backend still can be used by applications running in the service domain while it is exported to another domain. Note that this is a new behavior starting with the Solaris 10 5/08 OS release. Before the Solaris 10 5/08 OS release, disk backends were always opened exclusively, and it was not possible to have a backend opened non-exclusively.

Slice (`slice`) Option

A backend is normally exported either as a full disk or as a single slice disk depending on its type. If the `slice` option is specified, then the backend is forcibly exported as a single slice disk.

This option is useful when you want to export the raw content of a backend. For example, if you have a ZFS or SVM volume where you have already stored data and you want your guest domain to access this data, then you should export the ZFS or SVM volume using the `slice` option.

For more information about this option, see "Virtual Disk Backend" on page 83.

Virtual Disk Backend

The virtual disk backend is the location where data of a virtual disk are stored. The backend can be a disk, a disk slice, a file, or a volume, such as ZFS, SVM, or VxVM. A backend appears in a guest domain either as a full disk or as single slice disk, depending on whether the `slice` option is set when the backend is exported from the service domain. By default, a virtual disk backend is exported non-exclusively as a readable-writable full disk.

Physical Disk or Disk LUN

A physical disk or disk LUN is always exported as a full disk. In that case, virtual disk drivers (`vds` and `vdc`) forward I/O from the virtual disk and act as a pass-through to the physical disk or disk LUN.

A physical disk or disk LUN is exported from a service domain by exporting the device corresponding to the slice 2 (`s2`) of that disk without setting the `slice` option. If you export the slice 2 of a disk with the `slice` option, only this slice is exported and not the entire disk.

▼ Export a Physical Disk as a Virtual Disk

1. **For example, to export the physical disk `c1t48d0` as a virtual disk, you must export slice 2 of that disk (`c1t48d0s2`) from the service domain as follows.**

```
service# ldm add-vdsdev /dev/dsk/c1t48d0s2 c1t48d0@primary-vds0
```

2. **From the service domain, assign the disk (`pdisk`) to guest domain `ldg1`, for example.**

```
service# ldm add-vdisk pdisk c1t48d0@primary-vds0 ldg1
```

3. **After the guest domain is started and running the Solaris OS, you can list the disk (`c0d1`, for example) and see that the disk is accessible and is a full disk; that is, a regular disk with 8 slices.**

```
ldg1# ls -1 /dev/dsk/c0d1s*
/dev/dsk/c0d1s0
/dev/dsk/c0d1s1
/dev/dsk/c0d1s2
/dev/dsk/c0d1s3
/dev/dsk/c0d1s4
/dev/dsk/c0d1s5
/dev/dsk/c0d1s6
/dev/dsk/c0d1s7
```

Physical Disk Slice

A physical disk slice is always exported as a single slice disk. In that case, virtual disk drivers (`vds` and `vdc`) forward I/O from the virtual disk and act as a pass-through to the physical disk slice.

A physical disk slice is exported from a service domain by exporting the corresponding slice device. If the device is different from slice 2 then it is automatically exported as a single slice disk whether or not you specify the `slice` option. If the device is the slice 2 of the disk, you must set the `slice` option to export only slice 2 as a single slice disk; otherwise, the entire disk is exported as full disk.

▼ Export a Physical Disk Slice as a Virtual Disk

1. **For example, to export slice 0 of the physical disk `c1t57d0` as a virtual disk, you must export the device corresponding to that slice (`c1t57d0s0`) from the service domain as follows.**

```
service# ldm add-vdsdev /dev/dsk/c1t57d0s0 c1t57d0s0@primary-vds0
```

   You do not need to specify the `slice` option, because a slice is always exported as a single slice disk.

2. **From the service domain, assign the disk (`pslice`) to guest domain `ldg1`, for example.**

```
service# ldm add-vdisk pslice c1t57d0s0@primary-vds0 ldg1
```

3. **After the guest domain is started and running the Solaris OS, you can list the disk (`c0d13`, for example) and see that the disk is accessible.**

```
ldg1# ls -1 /dev/dsk/c0d13s*
/dev/dsk/c0d13s0
/dev/dsk/c0d13s1
/dev/dsk/c0d13s2
/dev/dsk/c0d13s3
/dev/dsk/c0d13s4
/dev/dsk/c0d13s5
/dev/dsk/c0d13s6
/dev/dsk/c0d13s7
```

Although there are 8 devices, because the disk is a single slice disk, only the first slice (`s0`) is usable.

▼ Export Slice 2

- **To export slice 2 (disk `c1t57d0s2`, for example) you must specify the `slice` option; otherwise, the entire disk is exported.**

```
# ldm add-vdsdev options=slice /dev/dsk/c1t57d0s2 c1t57d0s2@primary-vds0
```

File and Volume

A file or volume (for example from ZFS or SVM) is exported either as a full disk or as single slice disk depending on whether or not the `slice` option is set.

File or Volume Exported as a Full Disk

If you do not set the `slice` option, a file or volume is exported as a full disk. In that case, virtual disk drivers (`vds` and `vdc`) forward I/O from the virtual disk and manage the partitioning of the virtual disk. The file or volume eventually becomes a disk image containing data from all slices of the virtual disk and the metadata used to manage the partitioning and disk structure.

When a blank file or volume is exported as full disk, it appears in the guest domain as an unformatted disk; that is, a disk with no partition. Then you need to run the `format`(1M) command in the guest domain to define usable partitions and to write a valid disk label. Any I/O to the virtual disk fails while the disk is unformatted.

Note – Before the Solaris 10 5/08 OS release, when a blank file was exported as a virtual disk, the system wrote a default disk label and created default partitioning. This is no longer the case with the Solaris 10 5/08 OS release, and you must run `format`(1M) in the guest domain to create partitions.

▼ Export a File as a Full Disk

1. **From the service domain, create a file (`fdisk0` for example) to use as the virtual disk.**

```
service# mkfile 100m /ldoms/domain/test/fdisk0
```

   The size of the file defines the size of the virtual disk. This example creates a 100-megabyte blank file to get a 100-megabyte virtual disk.

2. **From the service domain, export the file as a virtual disk.**

```
service# ldm add-vdsdev /ldoms/domain/test/fdisk0
fdisk0@primary-vds0
```

   In this example, the `slice` option is not set, so the file is exported as a full disk.

3. **From the service domain, assign the disk (`fdisk`) to guest domain `ldg1`, for example.**

```
service# ldm add-vdisk fdisk fdisk0@primary-vds0 ldg1
```

4. **After the guest domain is started and running the Solaris OS, you can list the disk (`c0d5`, for example) and see that the disk is accessible and is a full disk; that is, a regular disk with 8 slices.**

```
ldg1# ls -1 /dev/dsk/c0d5s*
/dev/dsk/c0d5s0
/dev/dsk/c0d5s1
/dev/dsk/c0d5s2
/dev/dsk/c0d5s3
/dev/dsk/c0d5s4
/dev/dsk/c0d5s5
/dev/dsk/c0d5s6
/dev/dsk/c0d5s7
```

File or Volume Exported as a Single Slice Disk

If the `slice` option is set, then the file or volume is exported as a single slice disk. In that case, the virtual disk has only one partition (`s0`), which is directly mapped to the file or volume backend. The file or volume only contains data written to the virtual disk with no extra data like partitioning information or disk structure.

When a file or volume is exported as a single slice disk, the system simulates a fake disk partitioning which makes that file or volume appear as a disk slice. Because the disk partitioning is simulated, you do not create partitioning for that disk.

▼ Export a ZFS Volume as a Single Slice Disk

1. **From the service domain, create a ZFS volume (`zdisk0` for example) to use as a single slice disk.**

```
service# zfs create -V 100m ldoms/domain/test/zdisk0
```

   The size of the volume defines the size of the virtual disk. This example creates a 100-megabyte volume to get a 100-megabyte virtual disk.

2. **From the service domain, export the device corresponding to that ZFS volume, and set the `slice` option so that the volume is exported as a single slice disk.**

```
service# ldm add-vdsdev options=slice
/dev/zvol/dsk/ldoms/domain/test/zdisk0 zdisk0@primary-vds0
```

3. **From the service domain, assign the volume (`zdisk0`) to guest domain `ldg1`, for example.**

```
service# ldm add-vdisk zdisk0 zdisk0@primary-vds0 ldg1
```

4. **After the guest domain is started and running the Solaris OS, you can list the disk (`c0d9`, for example) and see that the disk is accessible and is a single slice disk (`s0`).**

```
ldg1# ls -1 /dev/dsk/c0d9s*
/dev/dsk/c0d9s0
/dev/dsk/c0d9s1
/dev/dsk/c0d9s2
/dev/dsk/c0d9s3
/dev/dsk/c0d9s4
/dev/dsk/c0d9s5
/dev/dsk/c0d9s6
/dev/dsk/c0d9s7
```

Exporting Volumes and Backward Compatibility

Before the Solaris 10 5/08 OS release, the `slice` option did not exist, and volumes were exported as single slice disks. If you have a configuration exporting volumes as virtual disks and if you upgrade the system to the Solaris 10 5/08 OS, volumes are now exported as full disks instead of single slice disks. To preserve the old behavior and to have your volumes exported as single slice disks, you need to do either of the following:

- Use the `ldm set-vdsdev` command in LDoms 1.1 software, and set the `slice` option for all volumes you want to export as single slice disks. Refer to the `ldm` man page or the *Logical Domains (LDoms) Manager 1.1 Man Page Guide* for more information about this command.
- Add the following line to the `/etc/system` file on the service domain.

```
set vds:vd_volume_force_slice = 1
```

Note – Setting this tunable forces the export of all volumes as single slice disks, and you cannot export any volume as a full disk.

Summary of How Different Types of Backends Are Exported

Backend	No Slice Option	Slice Option Set
Disk (disk slice 2)	Full disk*	Single slice diskd
Disk slice (not slice 2)	Single slice disk\	Single slice disk
File	Full disk	Single slice disk
Volume, including ZFS, SVM, or VxVM	Full disk	Single slice disk

* Export the entire disk.

\ A slice is always exported as a single slice disk.

d Export only slice 2

Guidelines

Using the Loopback File (`lofi`) Driver

It is possible to use the loopback file (`lofi`) driver to export a file as a virtual disk. However, doing this adds an extra driver layer and impacts performance of the virtual disk. Instead, you can directly export a file as a full disk or as a single slice disk. See "File and Volume" on page 85.

Directly or Indirectly Exporting a Disk Slice

To export a slice as a virtual disk either directly or indirectly (for example through a SVM volume), ensure that the slice does not start on the first block (block 0) of the physical disk by using the `prtvtoc`(1M) command.

If you directly or indirectly export a disk slice which starts on the first block of a physical disk, you might overwrite the partition table of the physical disk and make all partitions of that disk inaccessible.

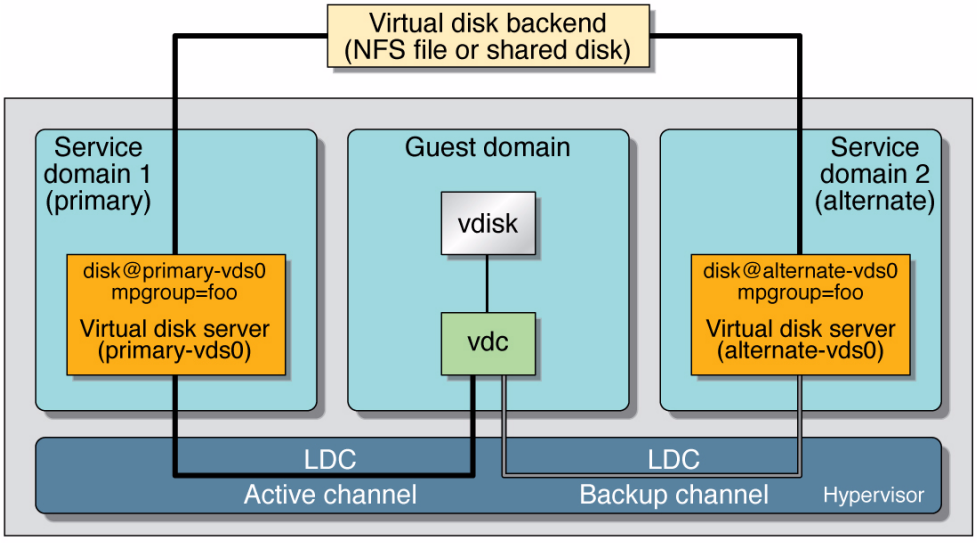
Configuring Virtual Disk Multipathing

If a virtual disk backend is accessible through different service domains, then you can configure virtual disk multipathing so that the virtual disk in a guest domain remains accessible if a service domain goes down. An example of a virtual disk backend accessible through different service domains is a file on a network file system (NFS) server or a shared physical disk connected to several service domains.

To enable virtual disk multipathing, you must export a virtual disk backend from the different service domains and add it to the same multipathing group (`mpgroup`). The `mpgroup` is identified by a name and is configured when the virtual disk backend is exported.

Figure 5-2 illustrates how to configure virtual disk multipathing. In this example, a multipathing group named **`foo`** is used to create a virtual disk, whose backend is accessible from two service domains: primary and alternate.

FIGURE 6-2 Configuring Virtual Disk Multipathing



▼ Configure Virtual Disk Multipathing

1. **Export the virtual backend from the primary service domain.**

```
# ldm add-vdsdev mpgroup=foo backend_path1 volume@primary-vds0
```

   Where *backend_path1* is the path to the virtual disk backend from the primary domain.

2. **Export the same virtual backend from the alternate service domain.**

```
# ldm add-vdsdev mpgroup=foo backend_path2 volume@alternate-vds0
```

   Where *backend_path2* is the path to the virtual disk backend from the alternate domain.

Note – *backend_path1* and *backend_path2* are paths to the same virtual disk backend, but from two different domains (primary and alternate). These paths might be the same or might be different depending on the configuration of the primary and alternate domains. The *volume* name is a user choice. It might be the same or different for both commands.

---

3. **Export the virtual disk to the guest domain.**

```
# ldm add-vdisk disk_name volume@primary-vds0 ldom
```

Note – Although the virtual disk backend is exported several times through different service domains, you assign only one virtual disk to the guest domain and associate it with the virtual disk backend through any of the service domains.

Result of Virtual Disk Multipathing

Once you configure the virtual disk with multipathing and start the guest domain, the virtual disk accesses its backend through the service domain it has been associated with (the primary domain in this example). If this service domain becomes unavailable, then the virtual disk tries to access its backend through a difference service domain that is part of the same multipathing group.

Caution – When defining a multipathing group (`mpgroup`), ensure that the virtual disk backends that are part of the same `mpgroup` are effectively the same virtual disk backend. If you add different virtual disks' backends into the same `mpgroup`, you might see some unexpected behavior, and you can potentially lose or corrupt data stored on the backends.

CD, DVD and ISO Images

You can export a compact disc (CD) or digital versatile disc (DVD) the same way you export any regular disk. To export a CD or DVD to a guest domain, export slice 2 of the CD or DVD device as a full disk; that is, without the `slice` option.

Note – You cannot export the CD or DVD drive itself; you only can export the CD or DVD that is inside the CD or DVD drive. Therefore, a CD or DVD must be present inside the drive before you can export it. Also, to be able to export a CD or DVD, that CD or DVD cannot be in use in the service domain. In particular, the Volume Management file system, `volfs`(7FS) service must not use the CD or DVD. See "Export a CD or DVD From the Service Domain to the Guest Domain" on page 93 for instructions on how to remove the device from use by `volfs`.

If you have an International Organization for Standardization (ISO) image of a CD or DVD stored in file or on a volume, and export that file or volume as a full disk then it appears as a CD or DVD in the guest domain.

When you export a CD, DVD, or an ISO image, it automatically appears as a read-only device in the guest domain. However, you cannot perform any CD control operations from the guest domain; that is, you cannot start, stop, or eject the CD from the guest domain. If the exported CD, DVD, or ISO image is bootable, the guest domain can be booted on the corresponding virtual disk.

For example, if you export a Solaris OS installation DVD, you can boot the guest domain on the virtual disk corresponding to that DVD and install the guest domain from that DVD. To do so, when the guest domain reaches the `ok` prompt, use the following command.

```
ok boot /virtual-devices@100/channel-devices@200/disk@n:f
```

Where *n* is the index of the virtual disk representing the exported DVD.

Note – If you export a Solaris OS installation DVD and boot a guest domain on the virtual disk corresponding to that DVD to install the guest domain, then you cannot change the DVD during the installation. So you might need to skip any step of the installation requesting a different CD/DVD, or you will need to provide an alternate path to access this requested media.

▼ Export a CD or DVD From the Service Domain to the Guest Domain

1. **Insert the CD or DVD in the CD or DVD drive.**
2. **From the service domain, check whether the volume management daemon, `vold`(1M), is running and online.**

```
service# svcs volfs
STATE          STIME    FMRI
online         12:28:12 svc:/system/filesystem/volfs:default
```

3. **Do one of the following.**
   - If the volume management daemon is not running or online, go to step 5.
   - If the volume management daemon is running and online, as in the example in Step 2, do the following:

   a. **Edit the `/etc/vold.conf` file and comment out the line starting with the following words.**

```
use cdrom drive....
```

   Refer to the `vold.conf`(1M) man page for more information.

   b. **From the service domain, restart the volume management file system service.**

```
service# svcadm refresh volfs
service# svcadm restart volfs
```

4. **From the service domain, find the disk path for the CD-ROM device.**

```
service# cdrw -l
Looking for CD devices...
   Node                  Connected Device                 Device type
----------------------+--------------------------------+-----------------
/dev/rdsk/c1t0d0s2    | MATSHITA CD-RW  CW-8124   DZ13 | CD Reader/Writer
```

5. **From the service domain, export the CD or DVD disk device as a full disk.**

```
service# ldm add-vdsdev /dev/dsk/c1t0d0s2 cdrom@primary-vds0
```

6. **From the service domain, assign the exported CD or DVD to the guest domain (`ldg1` in this example).**

```
service# ldm add-vdisk cdrom cdrom@primary-vds0 ldg1
```

Exporting a CD or DVD Multiple Times

A CD or DVD can be exported multiple times and assigned to different guest domains. See "Export a Virtual Disk Backend Multiple Times" on page 79 for more information.

Virtual Disk Timeout

By default, if the service domain providing access to a virtual disk backend is down, all I/O from the guest domain to the corresponding virtual disk is blocked. The I/O automatically is resumed when the service domain is operational and is servicing I/O requests to the virtual disk backend.

However, there are some cases when file systems or applications might not want the I/O operation to block, but for it to fail and report an error if the service domain is down for too long. It is now possible to set a connection timeout period for each virtual disk, which can then be used to establish a connection between the virtual disk client on a guest domain and the virtual disk server on the service domain. When that timeout period is reached, any pending I/O and any new I/O will fail as long as the service domain is down and the connection between the virtual disk client and server is not reestablished.

This timeout can be set by doing one of the following:

- Using the `ldm add-vdisk` command.

```
ldm add-vdisk timeout=seconds disk_name volume_name@service_name ldom
```

- Using the `ldm set-vdisk` command.

```
ldm set-vdisk timeout=seconds disk_name ldom
```

Specify the timeout in seconds. If the timeout is set to `0`, the timeout is disabled and I/O is blocked while the service domain is down (this is the default setting and behavior).

Alternatively, the timeout can be set by adding the following line to the `/etc/system` file on the guest domain.

```
set vdc:vdc_timeout = seconds
```

Note – If this tunable is set, it overwrites any timeout setting done using the `ldm` CLI. Also, the tunable sets the timeout for all virtual disks in the guest domain.

Virtual Disk and SCSI

If a physical SCSI disk or LUN is exported as a full disk, the corresponding virtual disk supports the user SCSI command interface, `uscsi`(7D) and multihost disk control operations `mhd`(7I). Other virtual disks, such as virtual disks having a file or a volume as a backend, do not support these interfaces.

As a consequence, applications or product features using SCSI commands (such as SVM `metaset`, or Solaris Cluster `shared devices`) can be used in guest domains only with virtual disks having a physical SCSI disk as a backend.

Note – SCSI operations are effectively executed by the service domain, which manages the physical SCSI disk or LUN used as a virtual disk backend. In particular, SCSI reservations are done by the service domain. Therefore, applications running in the service domain and in guest domains should not issue SCSI commands to the same physical SCSI disks; otherwise, this can lead to an unexpected disk state.

Virtual Disk and the `format`(1M) Command

The `format`(1M) command works in a guest domain with virtual disks exported as full disk. Single slice disks are not seen by the `format`(1M) command, and it is not possible to change the partitioning of such disks.

Virtual disks whose backends are SCSI disks support all `format`(1M) subcommands. Virtual disks whose backends are not SCSI disks do not support some `format`(1M) subcommands, such as `repair` and `defect`. In that case, the behavior of `format`(1M) is similar to the behavior of Integrated Drive Electronics (IDE) disks.

Using ZFS With Virtual Disks

This section describes using the Zettabyte File System (ZFS) to store virtual disk backends exported to guest domains. ZFS provides a convenient and powerful solution to create and manage virtual disk backends. ZFS enables:

- Storing disk images in ZFS volumes or ZFS files
- Using snapshots to backup disk images
- Using clones to duplicate disk images and provision additional domains

Refer to the *Solaris ZFS Administration Guide* in the Solaris 10 System Administrator Collection for more information about using the ZFS.

In the following descriptions and examples, the primary domain is also the service domain where disk images are stored.

Configuring a ZFS Pool in a Service Domain

To store the disk images, first create a ZFS storage pool in the service domain. For example, this command creates the ZFS storage pool `ldmpool` containing the disk `c1t50d0` in the `primary` domain.

```
primary# zpool create ldmpool c1t50d0
```

Storing Disk Images With ZFS

This example is going to create a disk image for guest domain `ldg1`. To do so, a ZFS for this guest domain is created, and all disk images of this guest domain will be stored on that file system.

```
Primary# zfs create ldmpool/ldg1
```

Disk images can be stored on ZFS volumes or ZFS files. Creating a ZFS volume, whatever its size, is quick using the `zfs create -V` command. On the other hand, ZFS files have to be created using the `mkfile` command. The command can take some time to complete, especially if the file to create is quite large, which is often the case when creating a disk image.

Both ZFS volumes and ZFS files can take advantage of ZFS features such as snapshot and clone, but a ZFS volume is a pseudo device while a ZFS file is a regular file.

If the disk image is to be used as a virtual disk onto which the Solaris OS is to be installed, then it should be large enough to contain:

- Installed software – about 6 gigabytes
- Swap partition – about 1 gigabyte
- Extra space to store system data – at least 1 gigabyte

Therefore, the size of a disk image to install the entire Solaris OS should be at least 8 gigabytes.

Examples of Storing Disk Images With ZFS

The following examples:

1. Create a 10-gigabyte image on a ZFS volume or file.
2. Export the ZFS volume or file as a virtual disk. The syntax to export a ZFS volume or file is the same, but the path to the backend is different.
3. Assign the exported ZFS volume or file to a guest domain.

When the guest domain is started, the ZFS volume or file appears as a virtual disk on which the Solaris OS can be installed.

▼ Create a Disk Image Using a ZFS Volume

- **For example, create a 10-gigabyte disk image on a ZFS volume.**

```
primary# zfs create -V 10gb ldmpool/ldg1/disk0
```

▼ Create a Disk Image Using a ZFS File

- **For example, create a 10-gigabyte disk image on a ZFS volume.**

```
primary# zfs create ldmpool/ldg1/disk0
primary# mkfile 10g /ldmpool/ldg1/disk0/file
```

▼ Export the ZFS Volume

- **Export the ZFS volume as a virtual disk.**

```
primary# ldm add-vdsdev /dev/zvol/dsk/ldmpool/ldg1/disk0
ldg1_disk0@primary-vds0
```

▼ Export the ZFS File

- **Export the ZFS file as a virtual disk.**

```
primary# ldm add-vdsdev /ldmpool/ldg1/disk0/file
ldg1_disk0@primary-vds0
```

▼ Assign the ZFS Volume or File to a Guest Domain

- **Assign the ZFS volume or file to a guest domain; in this example,** `ldg1`**.**

```
primary# ldm add-vdisk disk0 ldg1_disk0@primary-vds0 ldg1
```

Creating a Snapshot of a Disk Image

When your disk image is stored on a ZFS volume or on a ZFS file, you can create snapshots of this disk image by using the ZFS snapshot command.

Before you create a snapshot of the disk image, ensure that the disk is not currently in use in the guest domain to ensure that data currently stored on the disk image are coherent. There are several ways to ensure that a disk is not in use in a guest domain. You can either:

- Stop and unbind the guest domain. This is the safest solution, and this is the only solution available if you want to create a snapshot of a disk image used as the boot disk of a guest domain.
- Alternatively, you can unmount any slices of the disk you want to snapshot used in the guest domain, and ensure that no slice is in use the guest domain.

In this example, because of the ZFS layout, the command to create a snapshot of the disk image is the same whether the disk image is stored on a ZFS volume or on a ZFS file.

▼ Create a Snapshot of a Disk Image

- **Create a snapshot of the disk image that was created for the** `ldg1` **domain, for example.**

```
primary# zfs snapshot ldmpool/ldg1/disk0@version_1
```

Using Clone to Provision a New Domain

Once you have created a snapshot of a disk image, you can duplicate this disk image by using the ZFS clone command. Then the cloned image can be assigned to another domain. Cloning a boot disk image quickly creates a boot disk for a new guest domain without having to perform the entire Solaris OS installation process.

For example, if the `disk0` created was the boot disk of domain `ldg1`, do the following to clone that disk to create a boot disk for domain `ldg2`.

```
primary# zfs create ldmpool/ldg2
primary# zfs clone ldmpool/ldg1/disk0@version_1 ldmpool/ldg2/disk0
```

Then `ldompool/ldg2/disk0` can be exported as a virtual disk and assigned to the new `ldg2` domain. The domain `ldg2` can directly boot from that virtual disk without having to go through the OS installation process.

Cloning a Boot Disk Image

When a boot disk image is cloned, the new image is exactly the same as the original boot disk, and it contains any information that has been stored on the boot disk before the image was cloned, such as the host name, the IP address, the mounted file system table, or any system configuration or tuning.

Because the mounted file system table is the same on the original boot disk image and on the cloned disk image, the cloned disk image has to be assigned to the new domain in the same order as it was on the original domain. For example, if the boot disk image was assigned as the first disk of the original domain, then the cloned disk image has to be assigned as the first disk of the new domain. Otherwise, the new domain is unable to boot.

If the original domain was configured with a static IP address, then a new domain using the cloned image starts with the same IP address. In that case, you can change the network configuration of the new domain by using the `sys-unconfig`(1M) command. To avoid this problem you can also create a snapshot of a disk image of an unconfigured system.

▼ Create a Snapshot of a Disk Image of an Unconfigured System

1. **Bind and start the original domain.**
2. **Execute the `sys-unconfig`(1M) command.**
3. **After the `sys-unconfig`(1M) command completes, the domain halts.**

4. **Stop and unbind the domain; do *not* reboot it.**

5. **Take a snapshot of the domain boot disk image, for example.**

```
primary# zfs snapshot ldmpool/ldg1/disk0@unconfigured
```

6. **At this point you have the snapshot of the boot disk image of an unconfigured system. You can clone this image to create a new domain which, when first booted, asks for the configuration of the system.**

   If the original domain was configured with the Dynamic Host Configuration Protocol (DHCP), then a new domain using the cloned image also uses DHCP. In that case, you do not need to change the network configuration of the new domain because it automatically receives an IP address and its network configuration as it boots.

Note – The host ID of a domain is not stored on the boot disk, but it is assigned by the Logical Domains Manager when you create a domain. Therefore, when you clone a disk image, the new domain does not keep the host ID of the original domain.

Using Volume Managers in a Logical Domains Environment

This section describes using volume managers in a Logical Domains environment.

Using Virtual Disks on Top of Volume Managers

Any Zettabyte File System (ZFS), Solaris Volume Manager (SVM), or Veritas Volume Manager (VxVM) volume can be exported from a service domain to a guest domain as a virtual disk. A volume can be exported either as a single slice disk (if the `slice` option is specified with the `ldm add-vdsdev` command) or as a full disk.

Note – The remainder of this section uses an SVM volume as an example. However, the discussion also applies to ZFS and VxVM volumes.

The following example shows how to export a volume as a single slice disk. For example, if a service domain exports the SVM volume `/dev/md/dsk/d0` to `domain1` as a single slice disk, and `domain1` sees that virtual disk as `/dev/dsk/c0d2*`, then `domain1` only has an `s0` device; that is, `/dev/dsk/c0d2s0`.

The virtual disk in the guest domain (for example, `/dev/dsk/c0d2s0`) is directly mapped to the associated volume (for example, `/dev/md/dsk/d0`), and data stored onto the virtual disk from the guest domain are directly stored onto the associated volume with no extra metadata. So data stored on the virtual disk from the guest domain can also be directly accessed from the service domain through the associated volume.

Examples

- If the SVM volume `d0` is exported from the `primary` domain to `domain1`, then the configuration of `domain1` requires some extra steps.

```
primary# metainit d0 3 1 c2t70d0s6 1 c2t80d0s6 1 c2t90d0s6
primary# ldm add-vdsdev options=slice /dev/md/dsk/d0
vol3@primary-vds0
primary# ldm add-vdisk vdisk3 vol3@primary-vds0 domain1
```

- After `domain1` has been bound and started, the exported volume appears as `/dev/dsk/c0d2s0`, for example, and you can use it.

```
domain1# newfs /dev/rdsk/c0d2s0
domain1# mount /dev/dsk/c0d2s0 /mnt
domain1# echo test-domain1 > /mnt/file
```

- After `domain1` has been stopped and unbound, data stored on the virtual disk from `domain1` can be directly accessed from the primary domain through SVM volume `d0`.

```
primary# mount /dev/md/dsk/d0 /mnt
primary# cat /mnt/file
test-domain1
```

Note – A single slice disk cannot be seen by the `format`(1M) command, cannot be partitioned, and cannot be used as an installation disk for the Solaris OS. See "Virtual Disk Appearance" on page 80 for more information about this topic.

Using Virtual Disks on Top of SVM

When a RAID or mirror SVM volume is used as a virtual disk by another domain, then it has to be exported without setting the exclusive (`excl`) option. Otherwise, if there is a failure on one of the components of the SVM volume, then the recovery of the SVM volume using the `metareplace` command or using a hot spare does not start. The `metastat` command sees the volume as resynchronizing, but the resynchronization does not progress.

For example, `/dev/md/dsk/d0` is a RAID SVM volume exported as a virtual disk with the `excl` option to another domain, and `d0` is configured with some hot-spare devices. If a component of `d0` fails, SVM replaces the failing component with a hot spare and resynchronizes the SVM volume. However, the resynchronization does not start. The volume is reported as resynchronizing, but the resynchronization does not progress.

```
# metastat d0
d0: RAID
    State: Resyncing
    Hot spare pool: hsp000
    Interlace: 32 blocks
    Size: 20097600 blocks (9.6 GB)
Original device:
    Size: 20100992 blocks (9.6 GB)
Device                                          Start Block  Dbase    State Reloc
c2t2d0s1                                            330  No        Okay  Yes
c4t12d0s1                                           330  No        Okay  Yes
/dev/dsk/c10t600C0FF00000000000015153295A4B100d0s1  330  No  Resyncing  Yes
```

In such a situation, the domain using the SVM volume as a virtual disk has to be stopped and unbound to complete the resynchronization. Then the SVM volume can be resynchronized using the `metasync` command.

```
# metasync d0
```

Using Virtual Disks When VxVM Is Installed

When the Veritas Volume Manager (VxVM) is installed on your system, and if Veritas Dynamic Multipathing (DMP) is enabled on a physical disk or partition you want to export as virtual disk, then you have to export that disk or partition without setting the (non-default) `excl` option. Otherwise, you receive an error in `/var/adm/messages` while binding a domain that uses such a disk.

```
vd_setup_vd():  ldi_open_by_name(/dev/dsk/c4t12d0s2) = errno 16
vds_add_vd():  Failed to add vdisk ID 0
```

You can check if Veritas DMP is enabled by checking multipathing information in the output of the command `vxdisk list`; for example:

```
# vxdisk list Disk_3
Device:    Disk_3
devicetag: Disk_3
type:      auto
info:      format=none
flags:     online ready private autoconfig invalid
pubpaths:  block=/dev/vx/dmp/Disk_3s2 char=/dev/vx/rdmp/Disk_3s2
guid:      -
udid:      SEAGATE%5FST336753LSUN36G%5FDISKS%5F30323339483031443 04E0000
site:      -
Multipathing information:
numpaths:  1
c4t12d0s2  state=enabled
```

Alternatively, if Veritas DMP is enabled on a disk or a slice that you want to export as a virtual disk with the `excl` option set, then you can disable DMP using the `vxdmpadm` command. For example:

```
# vxdmpadm -f disable path=/dev/dsk/c4t12d0s2
```

Using Volume Managers on Top of Virtual Disks

This section describes using volume managers on top of virtual disks.

Using ZFS on Top of Virtual Disks

Any virtual disk can be used with ZFS. A ZFS storage pool (`zpool`) can be imported in any domain that sees all the storage devices that are part of this `zpool`, regardless of whether the domain sees all these devices as virtual devices or real devices.

Using SVM on Top of Virtual Disks

Any virtual disk can be used in the SVM local disk set. For example, a virtual disk can be used for storing the SVM metadevice state database, `metadb`(1M), of the local disk set or for creating SVM volumes in the local disk set.

Any virtual disk whose backend is a SCSI disk can be used in a SVM shared disk set, `metaset`(1M). Virtual disks whose backends are not SCSI disks cannot be added into a SVM share disk set. Trying to add a virtual disk whose backend is not a SCSI disk into a SVM shared disk set fails with an error similar to the following.

```
# metaset -s test -a c2d2
metaset: domain1: test: failed to reserve any drives
```

Using VxVM on Top of Virtual Disks

For VxVM support in guest domains, refer to the VxVM documentation from Symantec.

CHAPTER 7

Using a Virtual Network With Logical Domains

This chapter describes how to use a virtual network with Logical Domains software.

Introduction to a Virtual Network

A virtual network allows domains to communicate with each other without using any external physical networks. A virtual network also can allow domains to use the same physical network interface to access a physical network and communicate with remote systems. A virtual network is created by having a virtual switch to which you can connect virtual network devices.

Virtual Switch

A virtual switch (`vsw`) is a component running in a service domain and managed by the virtual switch driver. A virtual switch can be connected to some guest domains to enable network communications between those domains. In addition, if the virtual switch is associated also with a physical network interface, then this allows network communications between guest domains and the physical network over the physical network interface. A virtual switch also has a network interface, `vsw`*n*, where *n* is a number corresponding to the instance of the virtual switch, for example `vsw0` for the first virtual switch in a service domain. This interface allows the service domain to communicate with the other domains connected to that virtual switch. It can be used like any regular network interface and configured with the `ifconfig`(1M) command.

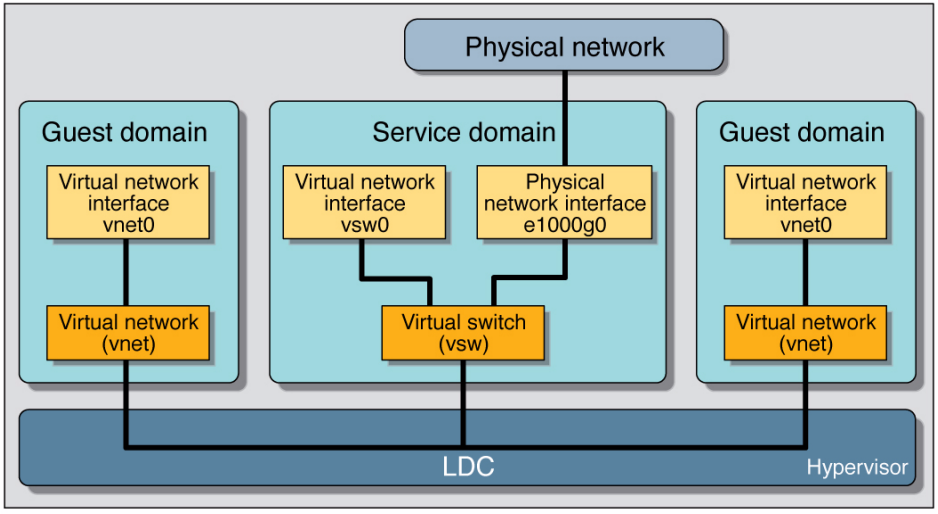
Note – When a virtual switch is added to a service domain, its network interface is not plumbed. So, by default, the service domain is unable to communicate with the guest domains connected to its virtual switch. To enable network communications between guest domains and the service domain, the network interface of the associated virtual switch must be plumbed and configured in the service domain. See "Enabling Networking Between the Control/Service Domain and Other Domains" on page 50 for instructions.

Virtual Network Device

A virtual network (`vnet`) device is a virtual device that is defined in a domain connected to a virtual switch. A virtual network device is managed by the virtual network driver, and it is connected to a virtual network through the hypervisor using logical domain channels (LDCs).

A virtual network device can be used as a network interface with the name `vnet`*n*, where *n* is a number corresponding to the instance of the virtual network device, and it can be used like any regular network interface and configured with the `ifconfig`(1M) command.

FIGURE 7-1 Setting Up a Virtual Network



Following is a explanation for the example in Figure 6-1.

- The virtual switch in the service domain is connected to the guest domains. This allows guest domains to communicate with each other.
- The virtual switch also is connected to the physical network interface `e1000g0`. This allows guest domains to communicate with the physical network.
- The virtual switch network interface `vsw0` is plumbed in the service domain, so this allows the two guest domains to communicate with the service domain.
- The virtual switch network interface `vsw0` in the service domain can be configured using the `ifconfig`(1M) command.
- The virtual network interfaces `vnet0` in the guest domains can be configured using the `ifconfig`(1M) command.

Basically the virtual switch behaves like a regular physical network switch and switches network packets between the different systems, such as guest domains, service domain, and physical network, to which it is connected.

Managing a Virtual Switch

This section describes adding a virtual switch to a domain, setting options for a virtual switch, and removing a virtual switch.

▼ Add a Virtual Switch

- **Use the following command syntax to add a virtual switch.**

```
# ldm add-vsw [default-vlan-id=vlan-id] [pvid=port-vlan-id] [vid=vlan-id1,vlan-id2,...]
[mac-addr=num] [net-dev=device] [mode=sc] vswitch_name ldom
```

Where:

- `default-vlan-id=`*vlan-id* specifies the default virtual local area network (VLAN) to which a virtual switch and its associated virtual network devices belong to implicitly, in untagged mode. It serves as the default port VLAN id (pvid) of the virtual switch and virtual network devices. Without this option, the default value of this property is `1`. Normally, you would not need to use this option. It is provided only as a way to change the default value of `1`. See "Using VLAN Tagging With Logical Domains Software" on page 122 for more information.

- `pvid=`*port-vlan-id* specifies the VLAN to which the virtual switch needs to be a member, in untagged mode. See "Using VLAN Tagging With Logical Domains Software" on page 122 for more information.
- `vid=`*vlan-id* specifies one or more VLANs to which a virtual switch needs to be a member, in tagged mode. See "Using VLAN Tagging With Logical Domains Software" on page 122 for more information.
- `mac-addr=`*num* is the MAC address to be used by this switch. The number must be in standard octet notation; for example, 80:00:33:55:22:66. If you do not specify a MAC address, the switch is automatically assigned an address from the range of public MAC addresses allocated to the Logical Domains Manager. See "Assigning MAC Addresses Automatically or Manually" on page 114 for more information.
- `net-dev=`*device* is the path to the network device over which this switch operates.
- `mode=sc` enables virtual networking support for prioritized processing of Solaris Cluster heartbeat packets in a Logical Domains environment. Applications like Solaris Cluster need to ensure that high priority heartbeat packets are not dropped by congested virtual network and switch devices. This option prioritizes Solaris Cluster heartbeat frames and ensures that they are transferred in a reliable manner.

  You must set this option when running Solaris Cluster in a Logical Domains environment and using guest domains as Solaris Cluster nodes. Do *not* set this option when you are not running Solaris Cluster software in guest domains, because you could impact virtual network performance.
- *vswitch_name* is the unique name of the switch that is to be exported as a service. Clients (network) can attach to this service.
- *ldom* specifies the logical domain in which to add a virtual switch.

▼ Set Options for an Existing Virtual Switch

- **Use the following command syntax to set options for a virtual switch that already exists.**

```
# ldm set-vsw [pvid=port-vlan-id] [vid=vlan-id1,vlan-id2,...] [mac-addr=num]
[net-dev=device] [mode=[sc]] vswitch_name
```

Where:

- `mode=` (left blank) stops special processing of Solaris Cluster heartbeat packets.
- Otherwise, the command arguments are the same as described in "Add a Virtual Switch" on page 109.

▼ Remove a Virtual Switch

- **Use the following command syntax to remove a virtual switch.**

```
# ldm rm-vsw [-f] vswitch_name
```

Where:

- `-f` attempts to force the removal of a virtual switch. The removal might fail.
- *vswitch_name* is the name of the switch that is to be removed as a service.

Managing a Virtual Network Device

This section describes adding a virtual network device to a domain, setting options for an existing virtual network device, and removing a virtual network device.

▼ Add a Virtual Network Device

- **Use the following command syntax to add a virtual network device.**

```
# ldm add-vnet [mac-addr=num] [mode=hybrid] [pvid=port-vlan-id]
[vid=vlan-id1,vlan-id2,...] if_name vswitch_name ldom
```

Where:

- `mac-addr=`*num* is the MAC address for this network device. The number must be in standard octet notation; for example, `80:00:33:55:22:66`. See "Assigning MAC Addresses Automatically or Manually" on page 114 for more information.
- `mode=hybrid` to request the system to use NIU Hybrid I/O on this `vnet` if possible. If it is not possible, the system reverts to virtual I/O. This hybrid mode is considered a delayed reconfiguration if set on an active `vnet`. See "Using NIU Hybrid I/O" on page 125 for more information.
- `pvid=`*port-vlan-id* specifies the VLAN to which the virtual network device needs to be a member, in untagged mode. See "Using VLAN Tagging With Logical Domains Software" on page 122 for more information.
- `vid=`*vlan-id* specifies one or more VLANs to which a virtual network device needs to be a member, in tagged mode. See "Using VLAN Tagging With Logical Domains Software" on page 122 for more information.

- *if_name*, interface name, is a unique name to the logical domain, assigned to this virtual network device instance for reference on subsequent `set-vnet` or `rm-vnet` subcommands.
- *vswitch_name* is the name of an existing network service (virtual switch) to which to connect.
- *ldom* specifies the logical domain to which to add the virtual network device.

▼ Set Options for an Existing Virtual Network Device

- **Use the following command syntax to set options for a virtual network device that already exists.**

```
# ldm set-vnet [mac-addr=num] [vswitch=vswitch_name] [mode=[hybrid]]
[pvid=port-vlan-id] [vid=vlan-id1,vlan-id2,...] if_name ldom
```

Where:

- `mode=` (left blank) disables NIU Hybrid I/O.
- *if_name*, interface name, is the unique name assigned to the virtual network device you want to set.
- *ldom* specifies the logical domain from which to remove the virtual network device.
- Otherwise, the command arguments are the same as described in "Add a Virtual Network Device" on page 111.

▼ Remove a Virtual Network Device

- **Use the following command syntax to remove a virtual network device.**

```
# ldm rm-vnet [-f] if_name ldom
```

Where:

- `-f` attempts to force the removal of a virtual network device from a logical domain. The removal might fail.
- *if_name*, interface name, is the unique name assigned to the virtual network device you want to remove.
- *ldom* specifies the logical domain from which to remove the virtual network device.

Determining the Solaris Network Interface Name Corresponding to a Virtual Network Device

There is no way to determine the Solaris OS network interface name on a guest, corresponding to a given virtual device, directly from the output provided by the `ldm list-*` commands. However, you can do this by using a combination of the output from `ldm list -l` command and the entries under `/devices` on the Solaris OS guest.

▼ Find Solaris OS Network Interface Name

In this example, guest domain `ldg1` contains two virtual network devices, `net-a` and `net-c`. To find the Solaris OS network interface name in `ldg1` that corresponds to `net-c`, do the following.

1. **Use the `ldm` command to find the virtual network device instance for `net-c`.**

```
# ldm list -l ldg1
...
NETWORK
NAME          SERVICE                     DEVICE       MAC
net-a         primary-vsw0@primary        network@0    00:14:4f:f8:91:4f
net-c         primary-vsw0@primary        network@2    00:14:4f:f8:dd:68
...
#
```

The virtual network device instance for `net-c` is `network@2`.

2. **To find the corresponding network interface on `ldg1`, log into `ldg1` and find the entry for this instance under `/devices`.**

```
# uname -n
ldg1
# find /devices/virtual-devices@100 -type c -name network@2\*
/devices/virtual-devices@100/channel-devices@200/network@2:vnet1
#
```

The network interface name is the part of the entry after the colon; that is, `vnet1`.

3. **Plumb `vnet1` to see that it has the MAC address `00:14:4f:f8:dd:68` as shown in the `ldm list -l` output for `net-c` in Step 1.**

```
# ifconfig vnet1
vnet1: flags=1000842<BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 3
          inet 0.0.0.0 netmask 0
          ether 0:14:4f:f8:dd:68
#
```

Assigning MAC Addresses Automatically or Manually

You must have enough media access control (MAC) addresses to assign to the number of logical domains, virtual switches, and virtual networks you are going to use. You can have the Logical Domains Manager automatically assign MAC addresses to a logical domain, a virtual network (`vnet`), and a virtual switch (`vsw`), or you can manually assign MAC addresses from your own pool of assigned MAC addresses. The `ldm` subcommands that set MAC addresses are `add-domain`, `add-vsw`, `set-vsw`, `add-vnet`, and `set-vnet`. If you do not specify a MAC address in these subcommands, the Logical Domains Manager assigns one automatically.

The advantage to having the Logical Domains Manager assign the MAC addresses is that it utilizes the block of MAC addresses dedicated for use with logical domains. Also, the Logical Domains Manager detects and prevents MAC address collisions with other Logical Domains Manager instances on the same subnet. This frees you from having to manually manage your pool of MAC addresses.

MAC address assignment happens as soon as a logical domain is created or a network device is configured into a domain. In addition, the assignment is persistent until the device, or the logical domain itself, is removed.

Range of MAC Addresses Assigned to Logical Domains Software

Logical domains have been assigned the following block of 512K MAC addresses:

`00:14:4F:F8:00:00 ~ 00:14:4F:FF:FF:FF`

The lower 256K addresses are used by the Logical Domains Manager for **automatic MAC address allocation**, and you *cannot* manually request an address in this range:

```
00:14:4F:F8:00:00 - 00:14:4F:FB:FF:FF
```

You can use the upper half of this range for **manual MAC address allocation**:

```
00:14:4F:FC:00:00 - 00:14:4F:FF:FF:FF
```

Automatic Assignment Algorithm

When you do not specify a MAC address in creating logical domain or a network device, the Logical Domains Manager automatically allocates and assigns a MAC address to that logical domain or network device. To obtain this MAC address, the Logical Domains Manager iteratively attempts to select an address and then checks for potential collisions.

Before selecting a potential address, the Logical Domains Manager first looks to see if it has a recently freed, automatically assigned address saved in a database for this purpose (see "Freed MAC Addresses" on page 116). If so, the Logical Domains Manager selects its candidate address from the database.

If no recently freed addresses are available, the MAC address is randomly selected from the 256K range of addresses set aside for this purpose. The MAC address is selected randomly to lessen the chance of a duplicate MAC address being selected as a candidate.

The address selected is then checked against other Logical Domains Managers on other systems to prevent duplicate MAC addresses from actually being assigned. The algorithm employed is described in "Duplicate MAC Address Detection" on page 115. If the address is already assigned, the Logical Domains Manager iterates, choosing another address, and again checking for collisions. This continues until a MAC address is found that is not already allocated, or a time limit of 30 seconds has elapsed. If the time limit is reached, then the creation of the device fails, and an error message similar to the following is shown.

```
Automatic MAC allocation failed.  Please set the vnet MAC address
manually.
```

Duplicate MAC Address Detection

To prevent the same MAC address from being allocated to different devices, one Logical Domains Manager checks with other Logical Domains Managers on other systems by sending a multicast message over the control domain's default network interface, including the address that the Logical Domain Manager wants to assign to

the device. The Logical Domains Manger attempting to assign the MAC address waits for one second for a response back. If a different device on another LDoms-enabled system has already been assigned that MAC address, the Logical Domains Manager on that system sends back a response containing the MAC address in question. If the requesting Logical Domains Manager receives a response, it knows the chosen MAC address has already been allocated, chooses another, and iterates.

By default, these multicast messages are sent only to other managers on the same subnet; the default time-to-live (TTL) is `1`. The TTL can be configured using the Service Management Facilities (SMF) property `ldmd/hops`.

Each Logical Domains Manager is responsible for:

- Listening for multicast messages
- Keeping track of MAC addresses assigned to its domains
- Looking for duplicates
- Responding so that duplicates do not occur

If the Logical Domains Manager on a system is shut down for any reason, duplicate MAC addresses could occur while the Logical Domains Manager is down.

Automatic MAC allocation occurs at the time the logical domain or network device is created and persists until the device or the logical domain is removed.

Freed MAC Addresses

When a logical domain or a device associated with an automatic MAC address is removed, that MAC address is saved in a database of recently freed MAC addresses for possible later use on that system. These MAC addresses are saved to prevent the exhaustion of Internet Protocol (IP) addresses from a Dynamic Host Configuration Protocol (DHCP) server. When DHCP servers allocate IP addresses, they do so for a period of time (the lease time). The lease duration is often configured to be quite long, generally hours or days. If network devices are created and removed at a high rate without the Logical Domains Manager reusing automatically allocated MAC addresses, the number of MAC addresses allocated could soon overwhelm a typically configured DHCP server.

When a Logical Domains Manager is requested to automatically obtain a MAC address for a logical domain or network device, it first looks to the freed MAC address database to see if there is a previously assigned MAC address it can reuse. If there is a MAC address available from this database, the duplicate MAC address detection algorithm is run. If the MAC address had not been assigned to someone else since it was previously freed, it will be reused and removed from the database.

If a collision is detected, the address is simply removed from the database. The Logical Domains Manager then either tries the next address in the database or if none is available, randomly picks a new MAC address.

Using Network Adapters With LDoms

In a logical domains environment, the virtual switch service running in a service domain can directly interact with GLDv3-compliant network adapters. Though non-GLDv3 compliant network adapters can be used in these systems, the virtual switch cannot interface with them directly. See "Configuring Virtual Switch and Service Domain for NAT and Routing" on page 117 for information about how to use non-GLDv3 compliant network adapters.

▼ Determine If a Network Adapter Is GLDv3-Compliant

1. **Use the Solaris OS `dladm`(1M) command, where, for example, `bge0` is the network device name.**

```
# dladm show-link bge0
bge0            type: non-vlan   mtu: 1500      device: bge0
```

2. **Look at `type:` in the output:**
   - GLDv3-compliant drivers will have a type of `non-vlan` or `vlan`.
   - Non-GLDv3-compliant drivers will have a type of `legacy`.

Configuring Virtual Switch and Service Domain for NAT and Routing

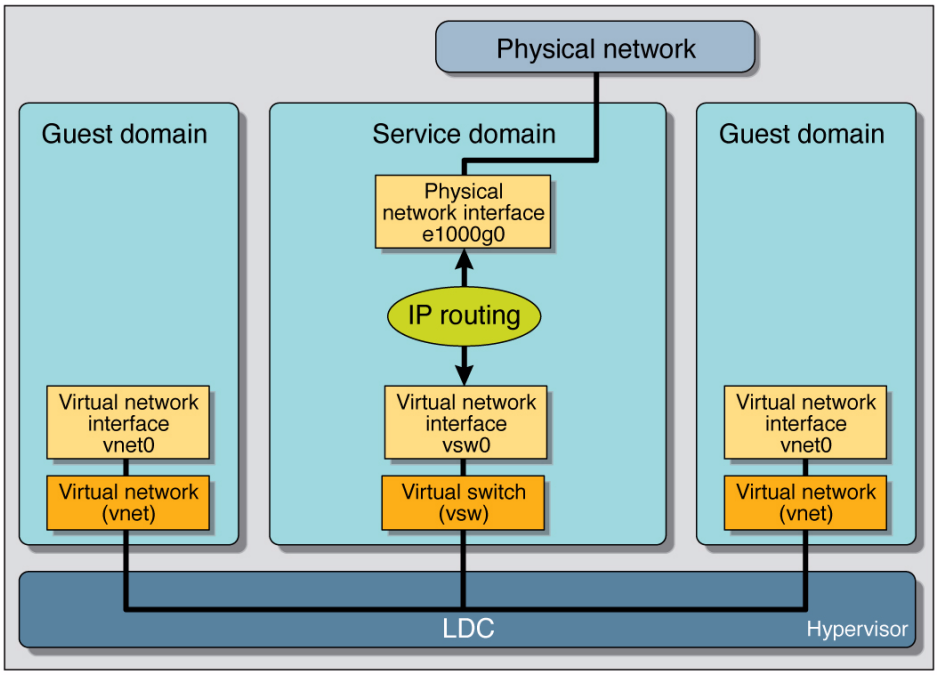
The virtual switch (`vsw`) is a layer-2 switch, that also can be used as a network device in the service domain. The virtual switch can be configured to act only as a switch between the virtual network (`vnet`) devices in the various logical domains but with no connectivity to a network outside the box through a physical device. In this mode, plumbing the `vsw` as a network device and enabling IP routing in the service domain enables virtual networks to communicate outside the box using the

service domain as a router. This mode of operation is very essential to provide external connectivity to the domains when the physical network adapter is not GLDv3-compliant.

The advantages of this configuration are:

- The virtual switch does not need to use a physical device directly and can provide external connectivity even when the underlying device is not GLDv3-compliant.
- The configuration can take advantage of the IP routing and filtering capabilities of the Solaris OS.

FIGURE 7-2 Virtual Network Routing



▼ Set Up the Virtual Switch to Provide External Connectivity to Domains

1. **Create a virtual switch with no associated physical device.**

   If assigning an address, ensure that the virtual switch has an unique MAC address.

```
primary# ldm add-vsw [mac-addr=xx:xx:xx:xx:xx:xx] primary-vsw0 primary
```

2. **Plumb the virtual switch as a network device in addition to the physical network device being used by the domain.**

   See "Configure the Virtual Switch as the Primary Interface" on page 50 for more information about plumbing the virtual switch.

3. **Configure the virtual switch device for DHCP, if needed.**

   See "Configure the Virtual Switch as the Primary Interface" on page 50 for more information about configuring the virtual switch device for DHCP.

4. **Create the `/etc/dhcp.vsw` file, if needed.**

5. **Configure IP routing in the service domain, and set up required routing tables in all the domains.**

   For information about how to do this, refer to the section on "Packet Forwarding and Routing on IPv4 Networks" in Chapter 5 "Configuring TCP/IP Network Services and IPv4 Administration" in the *System Administration Guide: IP Services* in the Solaris Express System Administrator Collection.

Configuring IPMP in a Logical Domains Environment

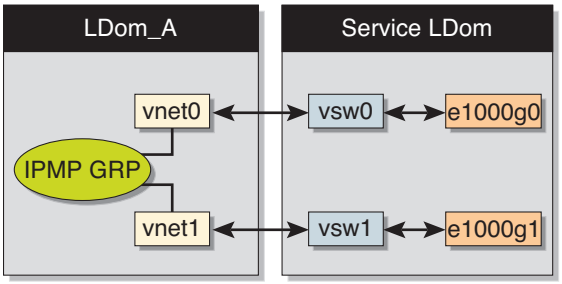
Internet Protocol Network Multipathing (IPMP) provides fault-tolerance and load balancing across multiple network interface cards. By using IPMP, you can configure one or more interfaces into an IP multipathing group. After configuring IPMP, the system automatically monitors the interfaces in the IPMP group for failure. If an interface in the group fails or is removed for maintenance, IPMP automatically migrates, or fails over, the failed interface's IP addresses. In a Logical Domains environment, either the physical or virtual network interfaces can be configured for failover using IPMP.

Configuring Virtual Network Devices into an IPMP Group in a Logical Domain

A logical domain can be configured for fault-tolerance by configuring its virtual network devices to an IPMP group. When setting up an IPMP group with virtual network devices, in a active-standby configuration, set up the group to use probe-based detection. Link-based detection and failover currently are not supported for virtual network devices in Logical Domains 1.1 software.

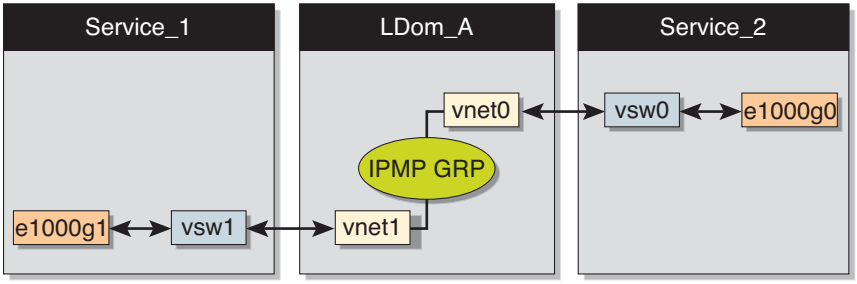
The following diagram shows two virtual networks (vnet1 and vnet2) connected to separate virtual switch instances (vsw0 and vsw1) in the service domain, which, in turn, use two different physical interfaces (e1000g0 and e1000g1). In the event of a physical interface failure, the IP layer in LDom_A detects failure and loss of connectivity on the corresponding vnet through probe-based detection, and automatically fails over to the secondary vnet device.

FIGURE 7-3 Two Virtual Networks Connected to Separate Virtual Switch Instances



Further reliability can be achieved in the logical domain by connecting each virtual network device (vnet0 and vnet1) to virtual switch instances in different service domains (as shown in the following diagram). Two service domains (Service_1 and Service_2) with virtual switch instances (vsw1 and vsw2) can be set up using a split-PCI configuration. In this case, in addition to network hardware failure, LDom_A can detect virtual network failure and trigger a failover following a service domain crash or shutdown.

FIGURE 7-4 Each Virtual Network Device Connected to Different Service Domains



Refer to the Solaris 10 *System Administration Guide: IP Services* for more information about how to configure and use IPMP groups.

▼ Configure a Host Route

If no explicit route is configured for a router in the network corresponding to the IPMP interfaces, then one or more explicit host routes to target systems need to be configured for the IPMP probe-based detection to work as expected. Otherwise, probe detection can fail to detect the network failures.

- **Configure a host route.**

```
# route add -host destination-IP gateway-IP -static
```

For example:

```
# route add -host 192.168.102.1 192.168.102.1 -static
```

Refer to the Solaris 10 *System Administration Guide: IP Services*, IPMP, Chapter 31 Administering IPMP (Tasks), "Configuring Target Systems" for more information.

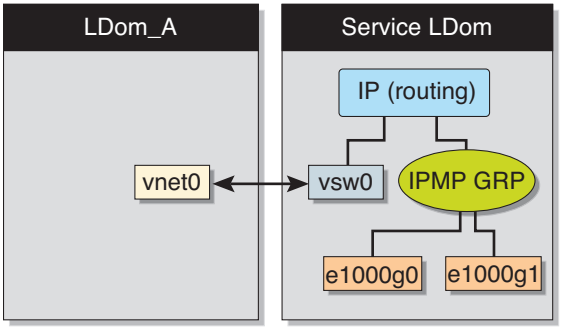
Configuring and Using IPMP in the Service Domain

Network failure detection and recovery can also be set up in a Logical Domains environment by configuring the physical interfaces in the service domain into a IPMP group. To do this, configure the virtual switch in the service domain as a network device, and configure the service domain itself to act as an IP router. (Refer to the Solaris 10 *System Administration Guide: IP Services* for information on setting up IP routing).

Once configured, the virtual switch sends all packets originating from virtual networks (and destined for an external machine), to its IP layer, instead of sending the packets directly via the physical device. In the event of a physical interface failure, the IP layer detects failure and automatically re-routes packets through the secondary interface.

Since the physical interfaces are directly being configured into a IPMP group, the group can be set up for either link-based or probe-based detection. The following diagram shows two network interfaces (`e1000g0` and `e1000g1`) configured as part of an IPMP group. The virtual switch instance (`vsw0`) has been plumbed as a network device to send packets to its IP layer.

FIGURE 7-5 Two Network Interfaces Configured as Part of IPMP Group



Using VLAN Tagging With Logical Domains Software

As of the release of Solaris 10 10/08 OS and LDoms 1.1 software, 802.1Q VLAN-Tagging support is available in the Logical Domains network infrastructure.

Note – Tagged VLANs are not supported in any of the previous releases for LDoms networking components.

The virtual switch (`vsw`) and virtual network (`vnet`) devices support switching of Ethernet packets based on the virtual local area network (VLAN) identifier (ID) and handle the necessary tagging or untagging of Ethernet frames.

You can create multiple VLAN interfaces over a `vnet` device in a guest domain. You can use the Solaris OS `ifconfig`(1M) command to create a VLAN interface over a virtual network device, the same way it is used to configure a VLAN interface over any other physical network device. The additional requirement in the LDoms environment is that you must assign the `vnet` to the corresponding VLANs using the Logical Domains Manager CLI commands. Refer to the `ldm`(1M) man page or the *Logical Domains (LDoms) Manager 1.1 Man Page Guide* for complete information about the Logical Domains Manager CLI commands.

Similarly, you can configure VLAN interfaces over a virtual switch device in the service domain. VLAN IDs 2 through 4094 are valid; VLAN ID 1 is reserved as the `default-vlan-id`.

When you create a vnet device on a guest domain, you must assign it to the required VLANs by specifying a port VLAN ID and zero or more VLAN IDs for this vnet, using the pvid= and vid= arguments to the ldm add-vnet command. This configures the virtual switch to support multiple VLANs in the LDoms network and switch packets using both MAC address and VLAN IDs in the network.

Similarly, any VLANs to which the vsw device itself should belong, when plumbed as a network interface, must be configured in the vsw device using the pvid= and vid= arguments to the ldm add-vsw command.

You can change the VLANs to which a device belongs using ldm set-vnet or ldm set-vsw command.

Port VLAN ID (PVID)

The PVID indicates a VLAN to which the virtual network device needs to be a member, in untagged mode. In this case, the vsw device provides the necessary tagging or untagging of frames for the vnet device over the VLAN specified by its PVID. Any outbound frames from the virtual network that are untagged are tagged with its PVID by the virtual switch. Inbound frames tagged with this PVID are untagged by the virtual switch, before sending it to the vnet device. Thus, assigning a PVID to a vnet implicitly means that the corresponding virtual network port on the virtual switch is marked untagged for the VLAN specified by the PVID. You can have only one PVID for a vnet device.

The corresponding virtual network interface, when configured using the ifconfig(1M) command without a VLAN ID and using only its device instance, results in the interface being implicitly assigned to the VLAN specified by the virtual network's PVID.

For example, if you were to plumb vnet instance 0, using the following command, and if the pvid= argument for the vnet has been specified as 10, the vnet0 interface would be implicitly assigned to belong to the VLAN 10.

```
# ifconfig vnet0 plumb
```

VLAN ID (VID)

The VID indicates the VLAN to which a virtual network device or virtual switch needs to be a member, in tagged mode. The virtual network device sends and receives tagged frames over the VLANs specified by its VIDs. The virtual switch passes any frames that are tagged with the specified VID between the virtual network device and the external network.

▼ Assign VLANs to a Virtual Switch and Virtual Network Device

1. **Assign the virtual switch (`vsw`) to two VLANs, for example. Configure VLAN 21 as untagged and VLAN 20 as tagged. Assign the virtual network (`vnet`) to three VLANs, for example. Configure VLAN 20 as untagged and VLAN 21 and 22 as tagged.**

```
# ldm add-vsw net-dev=e1000g0 pvid=21 vid=20 primary-vsw0 primary
# ldm add-vnet vnet01 primary-vsw0 pvid=20 vid=21,22 ldom1
```

2. **Plumb the VLAN interfaces.**

   This example assumes that the instance number of these devices is `0` in the domains and the VLANs are mapped to these subnets:

VLAN	Subnet
20	192.168.1.0 (netmask: 255.255.255.0)
21	192.168.2.0 (netmask: 255.255.255.0)
22	192.168.3.0 (netmask: 255.255.255.0)

   a. **Plumb the VLAN interface in the service (`primary`) domain.**

```
primary# ifconfig vsw0 plumb
primary# ifconfig vsw0 192.168.2.100 netmask 0xffffff00 broadcast + up
primary# ifconfig vsw20000 plumb
primary# ifconfig vsw20000 192.168.1.100 netmask 0xffffff00 broadcast + up
```

   b. **Plumb the VLAN interface in the guest (`ldom1`) domain.**

```
ldom1# ifconfig vnet0 plumb
ldom1# ifconfig vnet0 192.168.1.101 netmask 0xffffff00 broadcast + up
ldom1# ifconfig vnet21000 plumb
ldom1# ifconfig vnet21000 192.168.2.101 netmask 0xffffff00 broadcast + up
ldom1# ifconfig vnet22000 plumb
ldom1# ifconfig vnet22000 192.168.3.101 netmask 0xffffff00 broadcast + up
```

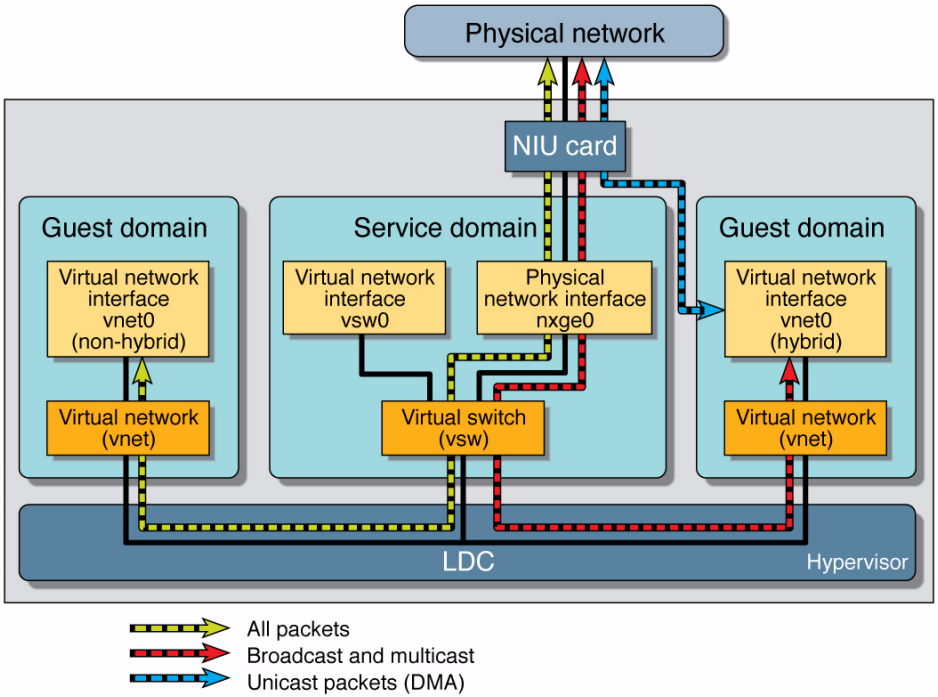
   For more information about how to configure VLAN interfaces in the Solaris OS, refer to "Administering Virtual Local Area Networks" in the *Solaris System Administration Guide: IP Services*.

Using NIU Hybrid I/O

LDoms 1.1 virtual I/O framework implements a *hybrid* I/O model for improved functionality and performance. The hybrid I/O model combines direct and virtualized I/O to allow flexible deployment of I/O resources to virtual machines. It is particularly useful when direct I/O does not provide full capability for the virtual machine, or direct I/O is not persistently or consistently available to the virtual machine. This could be because of resource availability or virtual machine migration. The hybrid I/O architecture is well-suited to the Network Interface Unit (NIU), a network I/O interface integrated on chip, on Sun UltraSPARC T2–based platforms. This allows the dynamic assignment of Direct Memory Access (DMA) resources to virtual networking devices and, thereby, provides consistent performance to applications in the domain.

As of the release of Solaris 10 10/08 OS and LDoms 1.1 software, NIU hybrid I/O is available for Sun UltraSPARC T2–based platforms. This feature is enabled by an optional hybrid mode that provides for a virtual network (`vnet`) device where the DMA hardware resources are loaned to a `vnet` device in a guest domain for improved performance. In the hybrid mode, a vnet device in a guest domain can send and receive unicast traffic from an external network directly into the guest domain using the DMA hardware resources. The broadcast traffic and unicast traffic to the other guest domains in the same system continue to be sent using the virtual I/O communication mechanism.

FIGURE 7-6 Hybrid Virtual Networking



The hybrid mode applies only for the `vnet` devices that are associated with a virtual switch (`vsw`) configured to use an NIU network device. As the shareable DMA hardware resources are limited, up to only three `vnet` devices per `vsw` can have DMA hardware resources assigned at a given time. If more than three `vnet` devices have the hybrid mode enabled, the assignment is done on a first-come, first-served basis. As there are two NIU network devices in a system, there can be a total of six `vnet` devices on two different virtual switches with DMA hardware resources assigned.

Following are points you need to be aware of when using this feature:

- Hybrid mode option for a `vnet` device is treated as a suggestion only. That means the DMA resources are assigned only when they are available and the device is capable of using them.
- Logical Domains Manager CLI commands do not validate the hybrid mode option; that is, it is possible to set the hybrid mode on any `vnet` or any number of `vnet` devices.
- Guest domains and the service domain need to run Solaris 10 10/08 OS at a minimum.

- Up to a maximum of only three `vnet` devices per `vsw` can have DMA hardware resources loaned at a given time. As there are two NIU network devices, there can be a total of six `vnet` devices with DMA hardware resources loaned.

Note – Set the hybrid mode only for three `vnet` devices per `vsw` so that they are guaranteed to have DMA hardware resources assigned.

- Hybrid mode is disabled by default for a `vnet` device. It needs to be explicitly enabled with Logical Domains Manager CLI commands. See "Enable Hybrid Mode" on page 128.

  (Refer to the *Logical Domains (LDoms) Manager 1.1 Man Page Guide* or the `ldm` man page for more details.)
- The hybrid mode option cannot be changed dynamically while the guest domain is active.
- The DMA hardware resources are assigned only when a `vnet` device is active that is plumbed in the guest domain.
- The Sun x8 Express 1/10G Ethernet Adapter (`nxge`) driver is used for the NIU card, but the same driver is also used for other 10-gigabit network cards. However, the NUI hybrid I/O feature is available for NIU network devices only.

▼ Configure a Virtual Switch With an NIU Network Device

- **For example, do the following to configure a virtual switch with a NIU network device.**

  a. For example, determine an NIU network device.

```
# grep nxge /etc/path_to_inst
"/niu@80/network@0" 0 "nxge"
"/niu@80/network@1" 1 "nxge"
```

  b. For example, configure a virtual switch.

```
# ldm add-vsw net-dev=nxge0 primary-vsw0 primary
```

▼ Enable Hybrid Mode

- **For example, enable a hybrid mode for a `vnet` device while it is being created.**

```
# ldm add-vnet mode=hybrid vnet01 primary-vsw0 ldom01
```

▼ Disable Hybrid Mode

- **For example, disable hybrid mode for a `vnet` device.**

```
# ldm set-vnet mode= vnet01 ldom01
```

CHAPTER 8

Migrating Logical Domains

This chapter describes how to migrate logical domains from one host machine to another as of this release of LDoms 1.1 software.

Introduction to Logical Domain Migration

Logical Domain Migration provides the ability to migrate a logical domain from one host machine to another. The host where the migration is initiated is referred to as the source machine, and the host where the domain is migrated to is referred to as the target machine. Similarly, once a migration is started, the domain to be migrated is referred to as the source domain and the shell of a domain created on the target machine is referred to as the target domain while the migration is in progress.

Overview of a Migration Operation

The Logical Domains Manager on the source machine accepts the request to migrate a domain and establishes a secure network connection with the Logical Domains Manager running on the target machine. Once this connection has been established, the migration occurs. The migration itself can be broken down into different phases.

Phase 1: After connecting with the Logical Domains Manager running in the target host, information about the source machine and domain are transferred to the target host. This information is used to perform a series of checks to determine whether a migration is possible. The checks differ depending on the state of the source domain. For example, if the source domain is active, a different set of checks are performed than if the domain is bound or inactive.

Phase 2: When all checks in Phase 1 have passed, the source and target machines prepare for the migration. In the case where the source domain is active, this includes shrinking the number of CPUs to one and suspending the domain. On the target machine, a domain is created to receive the source domain.

Phase 3: For an active domain, the next phase is to transfer all the runtime state information for the domain to the target. This information is retrieved from the hypervisor. On the target, the state information is installed in the hypervisor.

Phase 4: Handoff. After all state information is transferred, the handoff occurs when the target domain resumes execution (if the source was active) and the source domain is destroyed. From this point on, the target domain is the sole version of the domain running.

Software Compatibility

For a migration to occur, both the source and target machines must be running compatible software:

- The hypervisor on the source and target machines both must support the most recent version of the LDoms 1.1 firmware.

  If you see the following error, you do not have the correct version of system firmware on either the source or target machine.

```
System Firmware version on <downrev machine> does not support Domain Migration
Domain Migration of LDom <source domain> failed
```

- A compatible version of the Logical Domains Manager must be running on both machines.

Note – Since this is the first release of the migration feature, both machines must be running LDoms 1.1 software and up-to-date firmware. Refer to the *Logical Domains (LDoms) 1.1 Release Notes* for the latest firmware for your platform.

Authentication

Since the migration operation executes on two machines, a user must be authenticated on both the source and target host. In particular, the user must have the `solaris.ldoms.write` authorization on both machines.

The `ldm` command line interface for migration allows the user to specify an optional alternate user name for authentication on the target host. If this is not specified, the user name of the user executing the migration command is used. In both cases, the user is prompted for a password for the target machine.

Migrating an Active Domain

For the migration of an active domain to occur with LDoms 1.1 software, there is a certain set of requirements and restrictions imposed on the source logical domain, the source machine, and the target machine. The sections following describe these requirements and restrictions for each of the resource types.

CPUs

Following are the requirements and restrictions on CPUs when performing a migration.

- The source and target machines must have the same processor type running at the same frequency.
- The target machine must have sufficient free strands to accommodate the number of strands in use by the domain. In addition, full cores must be allocated for the migrated domain. If the number of strands in the source are less than a full core, the extra strands are unavailable to any domain until after the migrated domain is rebooted.
- After a migration, CPU dynamic reconfiguration (DR) is disabled for the target domain until it has been rebooted. Once a reboot has occurred, CPU DR becomes available for that domain.
- Either the source domain must have only a single strand, or the guest OS must support CPU DR, so that the domain can be shrunk to a single strand before migration. Conditions in the guest domain that would cause a CPU DR removal

to fail would also cause the migration attempt to fail. For example, processes bound to CPUs within the guest domain, or processor sets configured in the source logical domain, can cause a migration operation to fail.

Memory

There must be sufficient free memory on the target machine to accommodate the migration of the source domain. In addition, following are a few properties that must be maintained across the migration:

- It must be possible to create the same number of identically-sized memory blocks.
- The physical addresses of the memory blocks do not need to match, but the same real addresses must be maintained across the migration.

Physical Input/Output

The logical domain to be migrated must not contain any physical I/O devices. If a domain has any physical I/O devices, the migration fails.

Virtual Input/Output

All virtual I/O (VIO) services used by the source domain must be available on the target machine. In other words, the following conditions must exist:

- Each logical volume used in the source logical domain must also be available on the target host and must refer to the same storage.

Caution – If the logical volume used by the source as a boot device exists on the target but does not refer to the same storage, the migration appears to succeed, but the machine is not usable as it is unable to access its boot device. The domain has to be stopped, the configuration issue corrected, and then the domain restarted. Otherwise, the domain could be left in an inconsistent state.

- For each virtual network device in the source domain, a virtual network switch must exist on the target host, with the same name as the virtual network switch the device is attached to on the source host.

  For example, if `vnet0` in the source domain is attached to a virtual switch service name `switch-y`, then there must be a logical domain on the target host providing a virtual switch service named `switch-y`.

Note – The switches do not have to be connected to the same network for the migration to occur, though the migrated domain can experience networking problems if the switches are not connected to the same network.

MAC addresses used by the source domain that are in the automatically allocated range must be available for use on the target host.

- A virtual console concentrator (`vcc`) service must exist on the target host and have at least one free port. Explicit console constraints are ignored during the migration. The console for the target domain is created using the target domain name as the console group and using any available port on the first `vcc` device in the control domain. If there is a conflict with the default group name, the migration fails.

NIU Hybrid Input/Output

A domain using NIU Hybrid I/O resources can be migrated. A constraint specifying NIU Hybrid I/O resources is not a hard requirement of a logical domain. If such a domain is migrated to a machine that does not have available NIU resources, the constraint is preserved, but not fulfilled.

Cryptographic Units

You cannot migrate a logical domain that has bound cryptographic units. Attempts to migrate such a domain fail.

Delayed Reconfiguration

Any active delayed reconfiguration operations on the source or target hosts prevent a migration from starting. Delayed reconfiguration operations are blocked while a migration is in progress.

Operations on Other Domains

While a migration is in progress on a machine, any operation which could result in the modification of the Machine Description (MD) of the domain being migrated is blocked. This includes all operations on the domain itself as well as operations such as bind, stop, and start on other domains on the machine.

Migrating Bound or Inactive Domains

Because a bound or inactive domain is not executing at the time of the migration, there are fewer restrictions than when you migrate an active domain.

CPUs

You can migrate a bound or inactive domain between machines running different processor types and machines that are running at different frequencies.

The Solaris OS image in the guest must support the processor type on the target machine.

Virtual Input/Output

For an inactive domain, there are no checks performed against the virtual input/output (VIO) constraints. So, the VIO servers do not need to exist for the migration to succeed. As with any inactive domain, the VIO servers need to exist and be available at the time the domain is bound.

Performing a Dry Run

When you provide the `-n` option to the `migrate-domain` subcommand, migration checks are performed, but the source domain is not migrated. Any requirement that is not satisfied is reported as an error. This allows you to correct any configuration errors before attempting a real migration.

Note – Because of the dynamic nature of logical domains, it is possible for a dry run to succeed and a migration to fail and vice-versa.

Monitoring a Migration in Progress

When a migration is in progress, the source and target domains are displayed differently in the status output. In particular, the short version of the status output shows a new flag indicating the state of the migrating domain. The source domain shows a `s` to indicate that it is the source of the migration. The target domain shows a `t` to indicate that it is the target of a migration. If an error occurs that requires user intervention, an `e` is displayed.

In the long form of the status output, additional information is displayed about the migration. On the source, the percentage of the operation complete is displayed along with the target host and domain name. Similarly, on the target, the percentage of the operation complete is displayed along with the source host and domain name.

CODE EXAMPLE 8-1 Monitoring a Migration in Progress

```
# ldm ls -o status ldg-src
NAME
ldg-src

STATUS
    OPERATION    PROGRESS    TARGET
    migration    17%         t5440-sys-2
```

Canceling a Migration in Progress

Once a migration starts, if the `ldm` command is interrupted with a KILL signal, the migration is terminated. The target domain is destroyed, and the source domain is resumed if it was active. If the controlling shell of the `ldm` command is lost, the migration continues in the background.

A migration operation can also be canceled externally from the `ldm` command using the `cancel-operation` subcommand. This terminates the migration in progress, and the source domain resumes as the master domain.

Note – Once a migration has been initiated, suspending the `ldm`(1M) process does not pause the operation, because it is the Logical Domains Manager daemon (`ldmd`) on the source and target machines that are effecting the migration. The `ldm` process waits for a signal from the `ldmd` that the migration has been completed before returning.

Recovering From a Failed Migration

If the network connection is lost after the source has completed sending all the runtime state information to the target, but before the target can acknowledge that the domain has been resumed, the migration operation terminates, and the source is placed in an error state. This indicates that user interaction is required to determine whether or not the migration was completed successfully. In such a situation, take the following steps.

- Determine whether the target domain has resumed successfully. The target domain will be in one of two states:
  - If the migration completed successfully, the target domain is in the normal state.
  - If the migration failed, the target cleans up and destroys the target domain.
- If the target is resumed, it is safe to destroy the source domain in the error state. If the target is not present, the source domain is still the master version of the domain, and it must be recovered. To do this, execute the cancel command on the source machine. This clears the error state and restores the source domain back to its original condition.

Examples

CODE EXAMPLE 8-2 shows how a domain, called `ldg1`, can be migrated to a machine called `t5440-sys-2`.

CODE EXAMPLE 8-2 Migrating a Guest Domain

```
# ldm migrate-domain ldg1 t5440-sys-2
Target Password:
#
```

CODE EXAMPLE 8-3 shows that a domain can be renamed as part of the migration. In this example, `ldg-src` is the source domain, and it is renamed to `ldg-tgt` on the target machine (`t5440-sys-2`) as part of the migration. In addition, the user name (`root`) on the target machine is explicitly specified.

CODE EXAMPLE 8-3 Migrating and Renaming a Guest Domain

```
# ldm migrate ldg-src root@t5440-sys-2:ldg-tgt
Target Password:
#
```

CODE EXAMPLE 8-4 shows a sample failure message if the target domain does not have migration support; that is, if you are running an LDoms version prior to version 1.1.

CODE EXAMPLE 8-4 Migration Failure Message

```
# ldm migrate ldg1 t5440-sys-2
Target Password:
Failed to establish connection with ldmd(1m) on target: t5440-sys-2
Check that the 'ldmd' service is enabled on the target machine and
that the version supports Domain Migration. Check that the 'xmpp_enabled'
and 'incoming_migration_enabled' properties of the 'ldmd' service on
the target machine are set to 'true' using svccfg(1M).
```

CODE EXAMPLE 8-5 shows how to obtain status on a target domain while the migration is in progress. In this example, the source machine is `t5440-sys-1`.

CODE EXAMPLE 8-5 Obtaining Target Domain Status

```
# ldm ls -o status ldg-tgt
NAME
ldg-tgt

STATUS
    OPERATION    PROGRESS    SOURCE
    migration    55%         t5440-sys-1
```

CODE EXAMPLE 8-6 shows how to obtain parseable status on the source domain while the migration is in progress. In this example, the target machine is `t5440-sys-2`.

CODE EXAMPLE 8-6 Obtaining Source Domain Parseable Status

```
# ldm ls -o status -p ldg-src
VERSION 1.3
DOMAIN|name=ldg-src|
STATUS
|op=migration|progress=42|error=no|target=t5440-sys-2
```

CHAPTER 9

Other Information and Tasks

This chapter contains information and tasks about using the Logical Domains software that are not described in the preceding chapters.

Using CPU Power Management With LDoms 1.1 Software

To use CPU Power Management (PM) with LDoms 1.1 software, you first need to set the power management policy in ILOM 3.0 firmware. This section summarizes the information that you need to use power management with LDoms software. Refer to "Monitoring Power Consumption" in the *Sun Integration Lights Out Management (ILOM) 3.0 CLI Procedures Guide* for more details.

The power policy is the setting that governs system power usage at any point in time. The Logical Domains Manager, version 1.1, supports two power policies, assuming that the underlying platform has implemented Power Management features:

- Performance — The system is allowed to use all the power that is available.
- Elastic — The system power usage is adapted to the current utilization level. For example, power up or down just enough system components to keep utilization within thresholds at all times, even if the workload fluctuates.

For instructions on configuring the power policy using the ILOM 3.0 firmware CLI, refer to "Monitoring Power Consumption" in the *Sun Integration Lights Out Management (ILOM) 3.0 CLI Procedures Guide*.

Showing CPU Power-Managed Strands in LDoms 1.1 Software

This section shows how to list power-managed strands and virtual CPUs using LDoms 1.1 software.

▼ List CPU Power-Managed Strands

- **List power-managed strands by doing one of the following.**

  a. Use the `list -l` subcommand.

  A dash (`---`) in the `UTIL` column of the CPU means the strand is power-managed.

```
# ldm list -l primary
NAME          STATE   FLAGS  CONS  VCPU MEMORY  UTIL  UPTIME
primary       active  -n-cv  SP    8    4G      4.3%  7d 19h 43m

SOFTSTATE
Solaris running

VCPU
    VID    PID    UTIL STRAND
    0      0      0.0%   100%
    1      1      ---    100%
    2      2      ---    100%
    3      3      ---    100%
    4      4      ---    100%
    5      5      ---    100%
    6      6      ---    100%
    7      7      ---    100%
....
```

b. Use the parseable option (`-p`) to the list `-l` subcommand.

A blank after `util=` means the strand is power-managed.

```
# ldm ls -l -p

VCPU
|vid=0|pid=0|util=0.7%|strand=100
|vid=1|pid=1|util=|strand=100
|vid=2|pid=2|util=|strand=100
|vid=3|pid=3|util=|strand=100
|vid=4|pid=4|util=0.7%|strand=100
|vid=5|pid=5|util=|strand=100
|vid=6|pid=6|util=|strand=100
|vid=7|pid=7|util=|strand=100
```

▼ List Power-Managed CPUs

- **List power-managed CPUs by doing one of the following.**

a. Use the `list-devices -a cpu` subcommand.

In the power management (`PM`) column, a `yes` means the CPU is power-managed and a `no` means the CPU is powered on. It is assumed that 100 percent free CPUs are power-managed by default, hence the dash (`---`) under PM.

```
# ldm list-devices -a cpu
VCPU
    PID     %FREE      PM
    0       0          no
    1       0          yes
    2       0          yes
    3       0          yes
    4       100        ---
    5       100        ---
    6       100        ---
    7       100        ---
```

b. Use the parseable option (`-p`) to the `list-devices -a cpu` subcommand.

In the power management (`pm=`) field, a `yes` means the CPU is power-managed and a `no` means the CPU is powered on. It is assumed that 100 percent free CPUs are power-managed by default, hence the blank in that field.

```
# ldm list-devices -a -p cpu
VERSION 1.4
VCPU
|pid=0|free=0|pm=no
|pid=1|free=0|pm=yes
|pid=2|free=0|pm=yes
|pid=3|free=0|pm=yes
|pid=4|free=0|pm=no
|pid=5|free=0|pm=yes
|pid=6|free=0|pm=yes
|pid=7|free=0|pm=yes
|pid=8|free=100|pm=
|pid=9|free=100|pm=
|pid=10|free=100|pm=
```

Entering Names in the CLI

The following sections describe the restrictions on entering names in the Logical Domains Manager CLI.

File Names (*file*) and Variable Names (*var_name*)

- First character must be a letter, a number, or a forward slash (/).
- Subsequent letters must be letters, numbers, or punctuation.

Virtual Disk Server *backend* and Virtual Switch Device Names

- Must contain letters, numbers, or punctuation.

Configuration Name (*config_name*)

The logical domain configuration name (*config_name*) that you assign to a configuration stored on the system controller must have no more than 64 characters.

All Other Names

The remainder of the names, such as the logical domain name (*ldom*), service names (*vswitch_name*, *service_name*, *vdpcs_service_name*, and *vcc_name*), virtual network name (*if_name*), and virtual disk name (*disk_name*), must be in the following format:

- First character must be a letter or number.
- Subsequent characters must be letters, numbers, or any of the following characters: '-_+#.:;~()'

Listing Logical Domains Resources

This section shows the syntax usage for the `ldm` subcommands, defines some output terms, such as flags and utilization statistics, and provides examples that are similar to what you actually see as output.

Machine-Readable Output

If you are creating scripts that use `ldm list` command output, *always* use the `-p` option to produce the machine-readable form of the output. See "Generate a Parseable, Machine-Readable List (-p)" on page 153 for more information.

▼ Show Syntax Usage for `ldm` Subcommands

- **Look at syntax usage for all `ldm` subcommands.**

CODE EXAMPLE 9-1 Syntax Usage for All `ldm` Subcommands

```
primary# ldm --help

Usage:
     ldm [--help] command [options] [properties] operands
     ldm -V
```

CODE EXAMPLE 9-1 Syntax Usage for All `ldm` Subcommands *(Continued)*

```
Options:
     -V         Display version information

Command(s) for each resource (aliases in parens):

     bindings
         list-bindings [-e] [-p] [<ldom>...]

     services
         list-services [-e] [-p] [<ldom>...]

     constraints
         list-constraints ([-x] | [-e] [-p]) [<ldom>...]

     devices
         list-devices [-a] [-p] [cpu] [crypto|mau] [memory] [io]

     domain      ( dom )
         add-domain (-i <file> | mac-addr=<num] [hostid=<num>] <ldom> |
             <ldom>...)
         remove-domain (-a | <ldom>...)
         list-domain [-e] [-l] [-o <format>] [-p] [<ldom>...]
            'format' is one or more of:
            console,cpu,crypto,disk,domain,memory,network,physio,serial,status
         start-domain (-a | -i <file> | <ldom>...)
         stop-domain [-f] (-a | <ldom>...)
         bind-domain (-i <file> | <ldom>)
         unbind-domain <ldom>
         panic-domain <ldom>
         migrate-domain [-n|--dry-run] <source_ldom>
             [<user>@]<target_host>[:<target_ldom>]

     io
         add-io [bypass=on] <bus> <ldom>
         remove-io <bus> <ldom>

     crypto       ( mau )
         add-crypto <number> <ldom>
         set-crypto <number> <ldom>
         remove-crypto <number> <ldom>

     memory       ( mem )
         add-memory <number>[GMK] <ldom>
         set-memory <number>[GMK] <ldom>
         remove-memory <number>[GMK] <ldom>

     operation
```

CODE EXAMPLE 9-1 Syntax Usage for All `ldm` Subcommands *(Continued)*

```
        cancel-operation (migration | reconf) <ldom>

    reconf
        cancel-reconf <ldom>

    spconfig      ( config )
        add-spconfig <config_name>
        set-spconfig <config_name>
        remove-spconfig <config_name>
        list-spconfig

    variable    ( var )
        add-variable <var_name>=<value>... <ldom>
        set-variable <var_name>=<value>... <ldom>
        remove-variable <var_name>... <ldom>
        list-variable [<var_name>...] <ldom>

    vconscon    ( vcc )
        add-vconscon port-range=<x>-<y> <vcc_name> <ldom>
        set-vconscon port-range=<x>-<y> <vcc_name>
        remove-vconscon [-f] <vcc_name>

    vconsole    ( vcons )
        set-vcons [port=[<port-num>]] [group=<group>] [service=<vcc_server>]
            <ldom>

    vcpu
        add-vcpu <number> <ldom>
        set-vcpu <number> <ldom>
        remove-vcpu <number> <ldom>

    vdisk
        add-vdisk [timeout=<seconds>] <disk_name>
             <volume_name>@<service_name> <ldom>
        set-vdisk [timeout=<seconds>] [volume=<volume_name>@<service_name>]
            <disk_name> <ldom>
        remove-vdisk [-f] <disk_name> <ldom>

    vdiskserver ( vds )
        add-vdiskserver <service_name> <ldom>
        remove-vdiskserver [-f] <service_name>

    vdpcc       ( ndpsldcc )
        add-vdpcc <vdpcc_name> <service_name> <ldom>
        remove-vdpcc [-f] <vdpcc_name> <ldom>

    vdpcs       ( ndpsldcs )
```

CODE EXAMPLE 9-1 Syntax Usage for All `ldm` Subcommands *(Continued)*

```
          add-vdpcs <vdpcs_name> <ldom>
          remove-vdpcs [-f] <vdpcs_name>

     vdiskserverdevice   ( vdsdev )
          add-vdiskserverdevice [options={ro,slice,excl}] [mpgroup=<mpgroup>]
              <backend>  <volume_name>@<service_name>
          set-vdiskserverdevice options=[{ro,slice,excl}] [mpgroup=<mpgroup>]
              <volume_name>@<service_name>
          remove-vdiskserverdevice [-f] <volume_name>@<service_name>

     vnet
          add-vnet [mac-addr=<num>] [mode=hybrid] [pvid=<pvid>]
              [vid=<vid1,vid2,...>] <if_name> <vswitch_name> <ldom>
          set-vnet [mac-addr=<num>] [mode=[hybrid]] [pvid=[<pvid>]]
              [vid=[<vid1,vid2,...>]] [vswitch=<vswitch_name>] <if_name>  <ldom>
          remove-vnet [-f] <if_name> <ldom>

     vswitch     ( vsw )
          add-vswitch [default-vlan-id=<vid>] [pvid=<pvid>]
               [vid=<vid1,vid2,...>] [mac-addr=<num>] [net-dev=<device>]
               [mode=<mode>] <vswitch_name> <ldom>
          set-vswitch [pvid=[<pvid>]] [vid=[<vid1,vid2,...>]] [mac-addr=<num>]
               [net-dev=<device>] [mode=<mode>] <vswitch_name>
          remove-vswitch [-f] <vswitch_name>

Verb aliases:
          Alias          Verb
          -----          -------
          rm             remove
          ls             list

Command aliases:
          Alias          Command
          -----          -------
          cancel-op      cancel-operation
          create         add-domain
          destroy        remove-domain
          remove-reconf  cancel-reconf
          start          start-domain
          stop           stop-domain
          bind           bind-domain
          unbind         unbind-domain
          panic          panic-domain
          migrate        migrate-domain
```

Flag Definitions

The following flags can be shown in the output for a domain (`ldm list`). If you use the long, parseable options (`-l -p`) for the command, the flags are spelled out; for example, `flags=normal,control,vio-service`. If not, you see the letter abbreviation; for example `-n-cv-`. The list flag values are position dependent. Following are the values that can appear in each of the six columns from left to right.

Column 1

- `s` starting or stopping
- `-` placeholder

Column 2

- `n` normal
- `t` transition

Column 3

- `d` delayed reconfiguration
- `-` placeholder

Column 4

- `c` control domain
- `-` placeholder

Column 5

- `v` virtual I/O service domain
- `-` placeholder

Column 6

- `s` source domain in a migration
- `t` target domain in a migration
- `e` error occurred during a migration
- `-` placeholder

Utilization Statistic Definition

The per virtual CPU utilization statistic (`UTIL`) is shown on the long (`-l`) option of the `ldm list` command. The statistic is the percentage of time that the virtual CPU spent executing on behalf of the guest operating system. A virtual CPU is considered to be executing on behalf of the guest operating system except when it has been yielded to the hypervisor. If the guest operating system does not yield virtual CPUs to the hypervisor, the utilization of CPUs in the guest operating system will always show as `100%`.

The utilization statistic reported for a logical domain is the average of the virtual CPU utilizations for the virtual CPUs in the domain. A dash (`---`) in the `UTIL` column means that the strand is power-managed.

Examples of Various Lists

Note – The actual output might vary slightly from what is shown here.

▼ Show Software Versions (`-V`)

- **View the current software versions installed, and you receive a listing similar to the following.**

CODE EXAMPLE 9-2 Software Versions Installed

```
primary$ ldm -V

Logical Domain Manager (v 1.1)
   Hypervisor control protocol v 1.3
   Using Hypervisor MD v 0.1

System PROM:
   Hypervisor  v. 1.7.0.    @(#)Hypervisor 1.7.0. 2008/11/19 10:20
   OpenBoot    v. 4.30.0.   @(#)OBP 4.30.0. 2008/11/18 13:44
```

▼ Generate a Short List

- **Generate a short list for all domains.**

CODE EXAMPLE 9-3 Short List for All Domains

```
primary$ ldm list
NAME             STATE    FLAGS   CONS    VCPU  MEMORY   UTIL  UPTIME
primary          active   -t-cv           4     1G       0.5%  3d 21h 7m
ldg1             active   -t---   5000    8     1G        23%  2m
```

▼ Generate a Long List (`-l`)

- **Generate a long list for all domains.**

CODE EXAMPLE 9-4 Long List for All Domains

```
primary$ ldm list -l
NAME             STATE    FLAGS   CONS    VCPU  MEMORY   UTIL  UPTIME
primary          active   -t-cv           1     768M     0.0%  0s

VCPU
    VID    PID    UTIL STRAND
    0      0      0.0%   100%

MEMORY
    RA               PA               SIZE
    0x4000000        0x4000000        768M

IO
    DEVICE           PSEUDONYM        OPTIONS
    pci@780          bus_a
    pci@7c0          bus_b            bypass=on

VCC
    NAME             PORT-RANGE
    vcc0             5000-5100

VSW
    NAME             MAC               NET-DEV   DEVICE    MODE
    vsw0             08:00:20:aa:bb:e0 e1000g0   switch@0  prog,promisc
    vsw1             08:00:20:aa:bb:e1                     routed

VDS
    NAME             VOLUME         OPTIONS          DEVICE
    vds0             myvol-a        slice            /disk/a
                     myvol-b                         /disk/b
```

CODE EXAMPLE 9-4 Long List for All Domains *(Continued)*

```
                    myvol-c         ro,slice,excl    /disk/c
    vds1            myvol-d                          /disk/d

VDPCS
    NAME
    vdpcs0
    vdpcs1

------------------------------------------------------------------------------
NAME             STATE    FLAGS   CONS    VCPU  MEMORY   UTIL  UPTIME
ldg1             bound    -----   5000    1     512M

VCPU
    VID    PID    UTIL STRAND
    0      1              100%

MEMORY
    RA               PA               SIZE
    0x4000000        0x34000000       512M

NETWORK
    NAME         SERVICE                     DEVICE       MAC
    mynet-b      vsw0@primary                network@0    08:00:20:ab:9a:12
    mynet-a      vsw0@primary                network@1    08:00:20:ab:9a:11

DISK
    NAME             VOLUME                      DEVICE     SERVER
    mydisk-a         myvol-a@vds0                disk@0     primary
    mydisk-b         myvol-b@vds0                disk@1     primary

VDPCC
    NAME             SERVICE
    myvdpcc-a        vdpcs0@primary
    myvdpcc-b        vdpcs0@primary

VCONS
    NAME             SERVICE                     PORT
    mygroup          vcc0@primary                5000
```

▼ Generate an Extended List (-e)

- **Generate an extended list of all domains.**

CODE EXAMPLE 9-5 Extended List for all Domains

```
primary$ ldm list -e
NAME             STATE    FLAGS   CONS    VCPU  MEMORY   UTIL  UPTIME
primary          active   -t-cv           1     768M     0.0%  0s

VCPU
    VID    PID    UTIL STRAND
    0      0      0.0%   100%

MEMORY
    RA               PA               SIZE
    0x4000000        0x4000000        768M

IO
    DEVICE           PSEUDONYM        OPTIONS
    pci@780          bus_a
    pci@7c0          bus_b            bypass=on

VLDC
    NAME
    primary

VCC
    NAME             PORT-RANGE
    vcc0             5000-5100

VSW
    NAME             MAC               NET-DEV   DEVICE    MODE
    vsw0             08:00:20:aa:bb:e0 e1000g0   switch@0  prog,promisc
    vsw1             08:00:20:aa:bb:e1                     routed

VDS
    NAME             VOLUME         OPTIONS          DEVICE
    vds0             myvol-a        slice            /disk/a
                     myvol-b                         /disk/b
                     myvol-c        ro,slice,excl    /disk/c
    vds1             myvol-d                         /disk/d

VDPCS
    NAME
    vdpcs0
    vdpcs1

VLDCC
```

CODE EXAMPLE 9-5 Extended List for all Domains *(Continued)*

```
    NAME             SERVICE                     DESC
    hvctl            primary@primary             hvctl
    vldcc0           primary@primary             ds

------------------------------------------------------------------------------
NAME             STATE    FLAGS   CONS    VCPU  MEMORY   UTIL  UPTIME
ldg1             bound    -----   5000    1     512M

VCPU
    VID    PID    UTIL STRAND
    0      1              100%

MEMORY
    RA               PA               SIZE
    0x4000000        0x34000000       512M

VLDCC
 NAME             SERVICE                     DESC
 vldcc0           primary@primary             ds

NETWORK
    NAME         SERVICE                     DEVICE       MAC
    mynet-b      vsw0@primary                network@0    08:00:20:ab:9a:12
    mynet-a      vsw0@primary                network@1    08:00:20:ab:9a:11

DISK
    NAME             VOLUME                      DEVICE     SERVER
    mydisk-a         myvol-a@vds0                disk@0     primary
    mydisk-b         myvol-b@vds0                disk@1     primary

VDPCC
    NAME             SERVICE
    myvdpcc-a        vdpcs0@primary
    myvdpcc-b        vdpcs0@primary

VCONS
    NAME             SERVICE                     PORT
    mygroup          vcc0@primary                5000
```

▼ Generate a Parseable, Machine-Readable List (`-p`)

- **Generate a parseable, machine-readable list of all domains.**

CODE EXAMPLE 9-6 Machine-Readable List

```
primary$ ldm list -p
VERSION 1.0
DOMAIN|name=primary|state=active|flags=-t-cv|cons=|ncpu=1|mem=805306368|util=
0.0|uptime=0
DOMAIN|name=ldg1|state=bound|flags=-----|cons=5000|ncpu=1|mem=536870912|util=
|uptime=
```

▼ Generate a Subset of a Long List (`-o` *format*)

- **Generate output as a subset of resources by entering one or more of the following *format* options. If you specify more than one format, delimit the items by a comma with no spaces.**
  - `console` - output contains virtual console (`vcons`) and virtual console concentrator (`vcc`) service
  - `cpu` - output contains virtual CPU (`vcpu`) and physical CPU (`pcpu`)
  - `crypto` - cryptographic unit output contains Modular Arithmetic Unit (`mau`) and any other LDoms-supported cryptographic unit, such as the Control Word Queue (CWQ)
  - `disk` - output contains virtual disk (`vdisk`) and virtual disk server (`vds`)
  - `domain` - output contains variables (`var`), host ID (`hostid`), domain state, flags, and software state
  - `memory` - output contains `memory`
  - `network` - output contains media access control (`mac`) address , virtual network switch (`vsw`), and virtual network (`vnet`) device
  - `physio` - physical input/output contains peripheral component interconnect (`pci`) and network interface unit (`niu`)
  - `serial` - output contains virtual logical domain channel (`vldc`) service, virtual logical domain channel client (`vldcc`), virtual data plane channel client (`vdpcc`), virtual data plane channel service (`vdpcs`)
  - `status` - output contains status about a domain migration in progress.

The following examples show various subsets of output that you can specify.

CODE EXAMPLE 9-7 List CPU Information for the Control Domain

```
# ldm ls -o cpu primary
NAME
primary

VCPU
   VID    PID    UTIL STRAND
   0      0      1.0%   100%
   1      1      0.6%   100%
   2      2      0.2%   100%
   3      3      0.5%   100%
```

CODE EXAMPLE 9-8 List Domain Information for a Guest Domain

```
# ldm ls -o domain ldm2
NAME             STATE    FLAGS
ldm2             active   -t---

SOFTSTATE
Openboot initializing

VARIABLES
   auto-boot?=false
   boot-device=/virtual-devices@100/channel-devices@200/disk@0
```

CODE EXAMPLE 9-9 Lists Memory and Network Information for a Guest Domain

```
# ldm ls -o network,memory ldm1
NAME
ldm1

MAC
   00:14:4f:f9:dd:ae

MEMORY
   RA               PA               SIZE
   0x6800000        0x46800000       1500M

NETWORK
NAME           SERVICE               DEVICE     MAC               MODE PVID VID
ldm1-network0  primary-vsw0@primary network@0  00:14:4f:fb:21:0f    1
```

▼ List a Variable

- **List a variable (for example,** `boot-device`**) for a domain (for example,** `ldg1`**).**

CODE EXAMPLE 9-10 Variable List for a Domain

```
primary$ ldm list-variable boot-device ldg1
boot-device=/virtual-devices@100/channel-devices@200/disk@0:a
```

▼ List Bindings

- **List resources that are bound for a domain (for example,** `ldg1`**).**

CODE EXAMPLE 9-11 Bindings List for a Domain

```
primary$ ldm list-bindings ldg1
NAME             STATE    FLAGS   CONS    VCPU  MEMORY   UTIL  UPTIME
ldg1             bound    -----   5000    1     512M

VCPU
    VID    PID    UTIL STRAND
    0      1              100%

MEMORY
    RA               PA               SIZE
    0x4000000        0x34000000       512M

NETWORK
    NAME             SERVICE                     DEVICE     MAC
    mynet-b          vsw0@primary                network@0  08:00:20:ab:9a:12
        PEER                         MAC
        vsw0@primary                 08:00:20:aa:bb:e0
        mynet-a@ldg1                 08:00:20:ab:9a:11
        mynet-c@ldg2                 08:00:20:ab:9a:22
    NAME             SERVICE                     DEVICE     MAC
    mynet-a          vsw0@primary                network@1  08:00:20:ab:9a:11
        PEER                         MAC
        vsw0@primary                 08:00:20:aa:bb:e0
        mynet-b@ldg1                 08:00:20:ab:9a:12
        mynet-c@ldg2                 08:00:20:ab:9a:22

DISK
    NAME             VOLUME                        DEVICE     SERVER
    mydisk-a         myvol-a@vds0                  disk@0     primary
    mydisk-b         myvol-b@vds0                  disk@1     primary

VDPCC
```

CODE EXAMPLE 9-11 Bindings List for a Domain *(Continued)*

```
    NAME             SERVICE
    myvdpcc-a        vdpcs0@primary
    myvdpcc-b        vdpcs0@primary

VCONS
    NAME             SERVICE                     PORT
    mygroup          vcc0@primary                5000
```

▼ List Configurations

- **List logical domain configurations that have been stored on the SC.**

CODE EXAMPLE 9-12 Configurations List

```
primary$ ldm list-config
factory-default
3guests
foo [next poweron]
primary
reconfig-primary
```

Meaning of Labels

The labels to the right of the configuration name mean the following:

- [`current`] - last booted configuration, only as long as it matches the currently running configuration; that is, until you initiate a reconfiguration. After the reconfiguration, the annotation changes to [`next poweron`].
- [`next poweron`] - configuration to be used at the next power cycle.

▼ List Devices

- **List all server resources, bound and unbound.**

CODE EXAMPLE 9-13 List of All Server Resources

```
primary$ ldm list-devices -a
VCPU
    PID  %FREE     PM
    0       0      NO
    1       0      YES
    2       0      YES
    3       0      YES
    4       100    ---
    5       100    ---
    6       100    ---
    7       100    ---
    8       100    ---
    9       100    ---
    10      100    ---
    11      100    ---
    12      100    ---
    13      100    ---
    14      100    ---
    15      100    ---
    16      100    ---
    17      100    ---
    18      100    ---
    19      100    ---
    20      100    ---
    21      100    ---
    22      100    ---
    23      100    ---
    24      100    ---
    25      100    ---
    26      100    ---
    27      100    ---
    28      100    ---
    29      100    ---
    30      100    ---
    31      100    ---

MAU
    CPUSET                                          BOUND
    (0, 1, 2, 3)                                    ldg2
    (4, 5, 6, 7)
    (8, 9, 10, 11)
    (12, 13, 14, 15)
    (16, 17, 18, 19)
```

CODE EXAMPLE 9-13 List of All Server Resources *(Continued)*

```
    (20, 21, 22, 23)
    (24, 25, 26, 27)
    (28, 29, 30, 31)

MEMORY
    PA                  SIZE            BOUND
    0x0                 512K            _sys_
    0x80000             1536K           _sys_
    0x200000            62M             _sys_
    0x4000000           768M            primary
    0x34000000          512M            ldg1
    0x54000000          8M              _sys_
    0x54800000          2G              ldg2
    0xd4800000          29368M

IO
    DEVICE          PSEUDONYM       BOUND   OPTIONS
    pci@780         bus_a           yes
    pci@7c0         bus_b           yes     bypass=on
```

▼ List Available Memory

- **List the amount of memory available to be allocated.**

```
primary$ ldm list-devices mem
MEMORY
    PA                  SIZE
    0x14e000000         2848M
```

▼ List Services

- **List the services that are available.**

CODE EXAMPLE 9-14 Services List

```
primary$ ldm list-services
VDS
    NAME             VOLUME         OPTIONS          DEVICE
    primary-vds0
VCC
    NAME             PORT-RANGE
    primary-vcc0     5000-5100
VSW
   NAME          MAC               NET-DEV  DEVICE   MODE
   primary-vsw0 00:14:4f:f9:68:d0 e1000g0  switch@0 prog,promisc
```

Listing Constraints

To the Logical Domains Manager, constraints are one or more resources you want to have assigned to a particular domain. You either receive all the resources you ask to be added to a domain or you get none of them, depending upon the available resources. The `list-constraints` subcommand lists those resources you requested assigned to the domain.

▼ List Constraints for One Domain

- **List constraints for one domain (for example, `ldg1`).**

CODE EXAMPLE 9-15 Constraints List for One Domain

```
primary$ ldm list-constraints ldg1
DOMAIN
ldg1

VCPU
    COUNT
    1

MEMORY
    SIZE
    512M

NETWORK
    NAME         SERVICE                     DEVICE       MAC
    mynet-b      vsw0                        network@0    08:00:20:ab:9a:12
    mynet-b      vsw0                        network@0    08:00:20:ab:9a:12

DISK
    NAME             VOLUME
    mydisk-a         myvol-a@vds0
    mydisk-b         myvol-b@vds0

VDPCC
    NAME             SERVICE
    myvdpcc-a        vdpcs0@primary
    myvdpcc-b        vdpcs0@primary

VCONS
    NAME             SERVICE
    mygroup          vcc0
```

▼ List Constraints in XML Format

- **List constraints in XML format for a particular domain (for example, `ldg1`).**

CODE EXAMPLE 9-16 Constraints for a Domain in XML Format

```
primary$ ldm list-constraints -x ldg1
<?xml version="1.0"?>
<LDM_interface version="1.0">
  <data version="2.0">
    <ldom>
      <ldom_info>
        <ldom_name>ldg1</ldom_name>
      </ldom_info>
      <cpu>
        <number>8</number>
      </cpu>
      <memory>
        <size>1G</size>
      </memory>
      <network>
        <vnet_name>vnet0</vnet_name>
        <service_name>primary-vsw0</service_name>
        <mac_address>01:14:4f:fa:0f:55</mac_address>
      </network>
      <disk>
        <vdisk_name>vdisk0</vdisk_name>
        <service_name>primary-vds0</service_name>
        <vol_name>vol0</vol_name>
      </disk>
      <var>
        <name>boot-device</name>
        <value>/virtual-devices@100/channel-devices@200/disk@0:a</value>
      </var>
      <var>
        <name>nvramrc</name>
        <value>devalias vnet0 /virtual-devices@100/channel-devices@200/
network@0</value>
      </var>
      <var>
        <name>use-nvramrc?</name>
        <value>true</value>
      </var>
    </ldom>
  </data>
</LDM_interface>
```

▼ List Constraints in a Machine-Readable Format

- **List constraints for all domains in a parseable format.**

CODE EXAMPLE 9-17 Constraints for All Domains in a Machine-Readable Format

```
primary$ ldm list-constraints -p
VERSION 1.0
DOMAIN|name=primary
MAC|mac-addr=00:03:ba:d8:b1:46
VCPU|count=4
MEMORY|size=805306368
IO
|dev=pci@780|alias=
|dev=pci@7c0|alias=
VDS|name=primary-vds0
|vol=disk-ldg2|opts=|dev=/ldoms/nv72-ldg2/disk
|vol=vol0|opts=|dev=/ldoms/nv72-ldg1/disk
VCC|name=primary-vcc0|port-range=5000-5100
VSW|name=primary-vsw0|mac-addr=|net-dev=e1000g0|dev=switch@0
DOMAIN|name=ldg1
VCPU|count=8
MEMORY|size=1073741824
VARIABLES
|boot-device=/virtual-devices@100/channel-devices@200/disk@0:a
|nvramrc=devalias vnet0 /virtual-devices@100/channel-devices@200/network@0
|use-nvramrc?=true
VNET|name=vnet0|dev=network@0|service=primary-vsw0|mac-addr=01:14:4f:fa:0f:55
VDISK|name=vdisk0|vol=vol0@primary-vds0
```

Connecting to a Guest Console Over a Network

You can connect to a guest console over a network if the `listen_addr` property is set to the IP address of the control domain in the `vntsd`(1M) SMF manifest. For example:

```
$ telnet host-name 5001
```

Note – Enabling network access to a console has security implications. Any user can connect to a console and for this reason it is disabled by default.

A Service Management Facility manifest is an XML file that describes a service. For more information about creating an SMF manifest, refer to the Solaris 10 System Administrator Collection.

Note – To access a non-English OS in a guest domain through the console, the terminal for the console must be in the locale required by the OS.

Stopping a Heavily-Loaded Domain Can Time Out

An `ldm stop-domain` command can time out before the domain completes shutting down. When this happens, an error similar to the following is returned by the Logical Domains Manager.

```
LDom ldg8 stop notification failed
```

However, the domain could still be processing the shutdown request. Use the `ldm list-domain` command to verify the status of the domain. For example:

```
# ldm list-domain ldg8
NAME          STATE    FLAGS   CONS    VCPU MEMORY   UTIL UPTIME
ldg8          active   s----   5000    22   3328M    0.3% 1d 14h 31m
```

The preceding list shows the domain as active, but the `s` flag indicates that the domain is in the process of stopping. This should be a transitory state.

The following example shows the domain has now stopped.

```
# ldm list-domain ldg8
NAME          STATE    FLAGS   CONS    VCPU MEMORY   UTIL UPTIME
ldg8          bound    -----   5000    22   3328M
```

Determining Where Errors Occur by Mapping CPU and Memory Addresses

The Solaris Fault Management Architecture (FMA) reports CPU errors in terms of physical CPU numbers and memory errors in terms of physical memory addresses.

If you want to determine within which logical domain an error occurred and the corresponding virtual CPU number or real memory address within the domain, then you must perform a mapping.

CPU Mapping

The domain and the virtual CPU number within the domain, which correspond to a given physical CPU number, can be determined with the following procedures.

▼ Determine the CPU Number

1. **Generate a long parseable list for all domains.**

```
primary$ ldm ls -l -p
```

2. **Look for the entry in the list's `VCPU` sections that has a `pid` field equal to the physical CPU number.**
   - **If you find such an entry, the CPU is in the domain the entry is listed under, and the virtual CPU number within the domain is given by the entry's `vid` field.**
   - **If you do not find such an entry, the CPU is not in any domain.**

Memory Mapping

The domain and the real memory address within the domain, which correspond to a given physical memory address (PA), can be determined as follows.

▼ Determine the Real Memory Address

1. **Generate a long parseable list for all domains.**

```
primary$ ldm ls -l -p
```

2. **Look for the line in the list's `MEMORY` sections where the PA falls within the inclusive range *pa* to (*pa* + *size* - 1); that is, *pa* <= PA < (*pa* + *size* - 1).**

   Here *pa* and *size* refer to the values in the corresponding fields of the line.

   - **If you find such an entry, the PA is in the domain the entry is listed under and the corresponding real address within the domain is given by *ra* + (PA - *pa*).**
   - **If you do not find such an entry, the PA is not in any domain.**

Examples of CPU and Memory Mapping

Suppose you have a logical domain configuration as shown in CODE EXAMPLE 9-18, and you want to determine the domain and the virtual CPU corresponding to physical CPU number `5`, and the domain and the real address corresponding to physical address `0x7e816000`.

Looking through the `VCPU` entries in the list for the one with the `pid` field equal to `5`, you can find the following entry under logical domain `ldg1`.

```
|vid=1|pid=5|util=29|strand=100
```

Hence, the physical CPU number `5` is in domain `ldg1` and within the domain it has virtual CPU number `1`.

Looking through the `MEMORY` entries in the list, you can find the following entry under domain `ldg2`.

```
ra=0x8000000|pa=0x78000000|size=1073741824
```

Where 0x78000000 <= 0x7e816000 <= (0x78000000 + 1073741824 - 1); that is, *pa* <= PA <= (*pa* + *size* - 1).

Hence, the PA is in domain `ldg2` and the corresponding real address is 0x8000000 + (0x7e816000 - 0x78000000) = 0xe816000.

CODE EXAMPLE 9-18 Long Parseable List of Logical Domains Configurations

```
primary$ ldm ls -l -p
VERSION 1.0
DOMAIN|name=primary|state=active|flags=normal,control,vio-service|cons=
SP|ncpu=4|mem=1073741824|util=0.6|uptime=64801|softstate=Solaris running
VCPU
|vid=0|pid=0|util=0.9|strand=100
|vid=1|pid=1|util=0.5|strand=100
|vid=2|pid=2|util=0.6|strand=100
|vid=3|pid=3|util=0.6|strand=100
MEMORY
|ra=0x8000000|pa=0x8000000|size=1073741824
IO
|dev=pci@780|alias=bus_a
|dev=pci@7c0|alias=bus_b
VDS|name=primary-vds0|nclients=2
|vol=disk-ldg1|opts=|dev=/opt/ldoms/testdisk.1
|vol=disk-ldg2|opts=|dev=/opt/ldoms/testdisk.2
VCC|name=primary-vcc0|nclients=2|port-range=5000-5100
VSW|name=primary-vsw0|nclients=2|mac-addr=00:14:4f:fb:42:5c|net-dev=
e1000g0|dev=switch@0|mode=prog,promisc
VCONS|type=SP
DOMAIN|name=ldg1|state=active|flags=normal|cons=5000|ncpu=2|mem=
805306368|util=29|uptime=903|softstate=Solaris running
VCPU
|vid=0|pid=4|util=29|strand=100
|vid=1|pid=5|util=29|strand=100
MEMORY
|ra=0x8000000|pa=0x48000000|size=805306368
VARIABLES
|auto-boot?=true
|boot-device=/virtual-devices@100/channel-devices@200/disk@0
VNET|name=net|dev=network@0|service=primary-vsw0@primary|mac-addr=
00:14:4f:f9:8f:e6
VDISK|name=vdisk-1|vol=disk-ldg1@primary-vds0|dev=disk@0|server=primary
VCONS|group=group1|service=primary-vcc0@primary|port=5000
DOMAIN|name=ldg2|state=active|flags=normal|cons=5001|ncpu=3|mem=
1073741824|util=35|uptime=775|softstate=Solaris running
VCPU
|vid=0|pid=6|util=35|strand=100
|vid=1|pid=7|util=34|strand=100
|vid=2|pid=8|util=35|strand=100
MEMORY
|ra=0x8000000|pa=0x78000000|size=1073741824
VARIABLES
```

CODE EXAMPLE 9-18 Long Parseable List of Logical Domains Configurations *(Continued)*

```
|auto-boot?=true
|boot-device=/virtual-devices@100/channel-devices@200/disk@0
VNET|name=net|dev=network@0|service=primary-vsw0@primary|mac-addr=
00:14:4f:f9:8f:e7
VDISK|name=vdisk-2|vol=disk-ldg2@primary-vds0|dev=disk@0|server=primary
VCONS|group=group2|service=primary-vcc0@primary|port=5000
```

Using Console Groups

The virtual network terminal server daemon, `vntsd`(1M), enables you to provide access for multiple domain consoles using a single TCP port. At the time of domain creation, the Logical Domains Manager assigns a unique TCP port to each console by creating a new default group for that domain's console. The TCP port is then assigned to the console group as opposed to the console itself. The console can be bound to an existing group using the `set-vcons` subcommand.

▼ Combine Multiple Consoles Into One Group

1. Bind the consoles for the domains into one group.

The following example shows binding the console for three different domains (`ldg1`, `ldg2`, and `ldg3`) to the same console group (`group1`).

```
primary# ldm set-vcons group=group1 service=primary-vcc0 ldg1
primary# ldm set-vcons group=group1 service=primary-vcc0 ldg2
primary# ldm set-vcons group=group1 service=primary-vcc0 ldg3
```

2. Connect to the associated TCP port (`localhost` at port `5000` in this example).

```
# telnet localhost 5000
primary-vnts-group1: h, l, c{id}, n{name}, q:
```

You are prompted to select one of the domain consoles.

3. List the domains within the group by selecting `l` (list).

```
primary-vnts-group1: h, l, c{id}, n{name}, q: l
DOMAIN ID           DOMAIN NAME                   DOMAIN STATE
0                   ldg1                          online
1                   ldg2                          online
2                   ldg3                          online
```

Note – To re-assign the console to a different group or `vcc` instance, the domain must be unbound; that is, it has to be in the inactive state. Refer to the Solaris 10 OS `vntsd`(1M) man page for more information on configuring and using SMF to manage `vntsd` and using console groups.

Operating the Solaris OS With Logical Domains

This section describes the changes in behavior in using the Solaris OS that occur once a configuration created by the Logical Domains Manager is instantiated; that is, domaining is enabled.

Note – Any discussion about whether domaining is enabled pertains only to Sun UltraSPARC T1–based platforms. Otherwise, domaining is always enabled.

OpenBoot Firmware Not Available After Solaris OS Has Started If Domaining Is Enabled

Domaining is enabled once a logical domains configuration created by the Logical Domains Manager is instantiated. If domaining is enabled, the OpenBoot™ firmware is not available after the Solaris OS has started, because it is removed from memory.

To reach the `ok` prompt from the Solaris OS, you must halt the domain. You can use the Solaris OS `halt` command to halt the domain.

Power-Cycling a Server

Whenever performing any maintenance on a system running LDoms software that requires power-cycling the server, you must save your current logical domain configurations to the SC first.

▼ Save Your Current Logical Domain Configurations to the SC

- **Use the following command.**

```
# ldm add-config config_name
```

Do Not Use the psradm(1M) Command on Active CPUs in a Power-Managed Domain

Do not attempt to change an active CPU's operational status in a power-managed domain by using the psradm(1M) command. This only applies if your platform supports power management.

Result of Solaris OS Breaks

If domaining is not enabled, the Solaris OS normally goes to the OpenBoot prompt after a break is issued. The behavior described in this section is seen in two situations:

1. You press the L1-A key sequence when the input device is set to keyboard.
2. You enter the send break command when the virtual console is at the telnet prompt.

If domaining is enabled, you receive the following prompt after these types of breaks.

```
c)ontinue, s)ync, r)eboot, h)alt?
```

Type the letter that represents what you want the system to do after these types of breaks.

Results From Halting or Rebooting the Control Domain

The following table shows the expected behavior of halting or rebooting the control (primary) domain.

Note – The question in TABLE 9-1 regarding whether domaining is enabled pertains only to the Sun UltraSPARC T1 processors. Otherwise, domaining is always enabled.

TABLE 9-1 Expected Behavior of Halting or Rebooting the Control (`primary`) Domain

Command	Domaining Enabled?	Other Domain Configured?	Behavior
`halt`	Disabled	N/A	**For Sun UltraSPARC T1 Processors:** Drops to the `ok` prompt.
	Enabled	No	**For Sun UltraSPARC T1 Processors:** System either resets and goes to the OpenBoot `ok` prompt or goes to the following prompt: `r)eboot, o)k prompt, or h)alt?` **For Sun UltraSPARC T2 Processors:** Host powered off and stays off until powered on at the SC.
	Enabled	Yes	Soft resets and boots up if the variable `auto-boot?=true`. Soft resets and halts at ok prompt if the variable `auto-boot?=false`.
`reboot`	Disabled	N/A	**For Sun UltraSPARC T1 Processors:** Powers off and powers on the host.
	Enabled	No	**For Sun UltraSPARC T1 Processors:** Powers off and powers on the host. **For Sun UltraSPARC T2 Processors:** Reboots the host, no power off.
	Enabled	Yes	**For Sun UltraSPARC T1 Processors:** Powers off and powers on the host. **For Sun UltraSPARC T2 Processors:** Reboots the host, no power off.
`shutdown -i 5`	Disabled	N/A	**For Sun UltraSPARC T1 Processors:** Powers off the host.
	Enabled	No	Host powered off, stays off until powered on at the SC.
	Enabled	Yes	Soft resets and reboots.

Using LDoms With ALOM CMT

The section describes information to be aware of in using Advanced Lights Out Manager (ALOM) chip multithreading (CMT) with the Logical Domains Manager. For more information about using the ALOM CMT software, refer to the *Advanced Lights Out Management (ALOM) CMT v1.3 Guide*.

Caution – The ALOM CMT documentation refers to only one domain, so you must be aware that the Logical Domains Manager is introducing multiple domains. If a logical domain is restarted, I/O services for guest domains might be unavailable until the control domain has restarted. This is because the control domain functions as a service domain in the Logical Domains Manager 1.1 software. Guest domains appear to freeze during the reboot process. Once the control domain has fully restarted, the guest domains resume normal operations. It is only necessary to shut down guest domains when power is going to be removed from the entire server.

An additional option is available to the existing ALOM CMT command.

```
bootmode [normal|reset_nvram|bootscript=strong|config="config-name"]
```

The `config="`*config-name*`"` option enables you to set the configuration on the next power on to another configuration, including the `factory-default` shipping configuration.

You can invoke the command whether the host is powered on or off. It takes effect on the next host reset or power on.

▼ Reset the Logical Domain Configuration to the Default or Another Configuration

- **Reset the logical domain configuration on the next power on to the default shipping configuration by executing this command in ALOM CMT software.**

```
sc> bootmode config="factory-default"
```

You also can select other configurations that have been created with the Logical Domains Manager using the `ldm add-config` command and stored on the system controller (SC). The name you specify in the Logical Domains Manager

`ldm add-config` command can be used to select that configuration with the ALOM CMT `bootmode` command. For example, assume you stored the configuration with the name `ldm-config1`.

```
sc> bootmode config="ldm-config1"
```

Refer to the `ldm`(1M) man page or the *Logical Domains (LDoms) Manager 1.1 Man Page Guide* for more information about the `ldm add-config` command.

Enabling and Using BSM Auditing

The Logical Domains Manager uses the Solaris OS Basic Security module (BSM) auditing capability. BSM auditing provides the means to examine the history of actions and events on your control domain to determine what happened. The history is kept in a log of what was done, when it was done, by whom, and what was affected.

If you want to use this auditing capability, this section describes how to enable, verify, disable, print output, and rotate audit logs. You can find further information about BSM auditing in the Solaris 10 *System Administration Guide: Security Services*.

You can enable BSM auditing in one of two ways. When you want to disable auditing, be sure you use the same method that you used in enabling. The two methods are:

- Use the `enable-bsm.fin` finish script in the Solaris Security Toolkit.

  The `enable-bsm.fin` script is not used by default by the `ldm_control-secure.driver`. You must enable the finish script in your chosen driver.
- Use the Solaris OS `bsmconv`(1M) command.

Here are the procedures for both methods.

▼ Use the `enable-bsm.fin` Finish Script

1. **Copy the `ldm_control-secure.driver` to *my-ldm.driver*, where *my-ldm.driver* is the name for your copy of the `ldm_control-secure.driver`.**
2. **Copy the `ldm_control-config.driver` to *my-ldm-config.driver*, where *my-ldm-config.driver* is the name for your copy of the `ldm_control-config.driver`.**

3. **Copy the** `ldm_control-hardening.driver` **to** *my-ldm-hardening.driver*, **where** *my-ldm-hardening.driver* **is the name for your copy of the** `ldm_control-hardening.driver`.

4. **Edit** *my-ldm.driver* **to refer to the new configuration and hardening drivers,** *my-ldm-control.driver* **and** *my-ldm-hardening.driver*, **respectively.**

5. **Edit** *my-ldm-hardening.driver*, **and remove the pound sign (**`#`**) from in front of the following line in the driver.**

```
enable-bsm.fin
```

6. **Execute** *my-ldm.driver*.

```
# /opt/SUNWjass/bin/jass-execute -d my-ldm.driver
```

7. **Reboot the Solaris OS for auditing to take effect.**

▼ Use the Solaris OS `bsmconv`(1M) Command

1. **Add** `vs` **in the** `flags:` **line of the** `/etc/security/audit_control` **file.**

2. **Run the** `bsmconv`**(1M) command.**

```
# /etc/security/bsmconv
```

   For more information about this command, refer to the Solaris 10 Reference Manual Collection or the man page.

3. **Reboot the Solaris Operating System for auditing to take effect.**

▼ Verify that BSM Auditing is Enabled

1. **Type the following command.**

```
# auditconfig -getcond
```

2. **Check that** `audit condition = auditing` **appears in the output.**

▼ Disable Auditing

You can disable auditing in one of two ways, depending on how you enabled it. See "Enabling and Using BSM Auditing" on page 172.

1. **Do one of the following.**
   - Undo the Solaris Security Toolkit hardening run which enabled BSM auditing.

```
# /opt/SUNWjass/bin/jass-execute -u
```

   - Use the Solaris OS `bsmunconv`(1M) command.

```
# /etc/security/bsmunconv
```

2. **Reboot the Solaris OS for the disabling of auditing to take effect.**

▼ Print Audit Output

- **Use one of the following to print BSM audit output.**
  - For example, use the Solaris OS commands `auditreduce`(1M) and `praudit`(1M) to print audit output.

```
# auditreduce -c vs | praudit
# auditreduce -c vs -a 20060502000000 | praudit
```

  - Use the Solaris OS `praudit -x` command to print XML output.

▼ Rotate Audit Logs

- **Use the Solaris OS `audit -n` command to rotate audit logs.**

CHAPTER 10

Using the XML Interface With the Logical Domains Manager

This chapter explains the Extensible Markup Language (XML) communication mechanism through which external user programs can interface with Logical Domains software. These basic topics are covered:

- Transport – How communications are initiated between an external program and the Logical Domains (LDoms) Manager.
- Protocol – Format of XML messages sent to and received from the Logical Domains Manager.
- Events – Logical Domains Manager alert notifications.

For various schemas to use with the Logical Domains Manager, see Appendix A.

XML Transport

External programs use the Extensible Messaging and Presence Protocol (XMPP) – RFC 3920 to communicate with the Logical Domains Manager. XMPP is supported for both local and remote connections and is on by default. To shut off a remote connection, set the `ldmd/xmpp_enable` SMF property to `false` and restart the Logical Domains Manager.

```
# svcadm disable ldmd
# svccfg -s ldom/ldmd setprop ldmd/xmpp_enabled=false
# svcadm refresh ldmd
# svcadm enable ldmd
```

XMPP

The Logical Domains Manager implements an XMPP server which can communicate with numerous available XMPP client applications and libraries. The LDoms Manager uses the following security mechanisms:

- Transport Layer Security (TLS) to secure the communication channel between the client and itself.
- Simple Authentication and Security Layer (SASL) for authentication. `PLAIN` is the only SASL mechanism supported. You must send in a user name and password to the server, so it can authorize you before allowing monitoring or management operations.

Local Connections

The LDoms Manager detects whether user clients are running on the same domain as itself and, if so, does a minimal XMPP handshake with that client. Specifically, the SASL authentication step after the setup of a secure channel through TLS is skipped. Authentication and authorization are done based on the credentials of the process implementing the client interface.

Clients can choose to implement a full XMPP client or to simply run a streaming XML parser, such as the `libxml2` Simple API for XML (SAX) parser. Either way the client has to handle an XMPP handshake to the point of TLS negotiation. Refer to the XMPP specification for the sequence needed.

XML Protocol

After communication initialization is complete, LDoms-defined XML messages are sent next. There are two general types of XML messages:

- Request and response messages use the `<LDM_interface>` tag. This type of XML message is used for communicating commands and getting results back from the LDoms Manager, analogous to executing commands using the command-line interface (CLI). This tag is also used for event registration and unregistration.
- Event messages use the `<LDM_event>` tag. This type of XML message is used to asynchronously report events posted by the LDoms Manager.

Request and Response Messages

The XML interface into LDoms has two different formats:

- One format for sending commands into the LDoms Manager
- Another format for LDoms Manager to respond on the status of the incoming message and the actions requested within that message.

The two formats share many common XML structures, but are separated in this discussion for a better understanding of the differences between them. This document also contains an XML Schema which details the combined incoming and outgoing XML (See "LDM_Event XML Schema" on page 206).

Requests

An incoming XML request to the LDoms Manager at its most basic level includes a description of a single command, operating on a single object. More complicated requests can handle multiple commands and multiple objects per command. Following is the structure of a basic XML command.

CODE EXAMPLE 10-1 Format of a Single Command Operating on a Single Object

```
<LDM_interface version="1.0">
  <cmd>
    <action>Place command here</action>
    <data version="3.0">
      <Envelope>
        <References/>
        <!-- Note a <Section> section can be here instead of <Content> -->
        <Content xsi:type="ovf:VirtualSystem_Type" id="Domain name">
          <Section xsi:type="ovf:ResourceAllocationSection_type">
            <Item>
              <rasd:OtherResourceType>
                LDom Resource Type
              </rasd:OtherResourceType>
              <gprop:GenericProperty
              key="Property name">
                Property Value
            </gprop:GenericProperty>
            </Item>
          </Section>
          <!-- Note: More Sections sections can be placed here -->
        </Content>
      </Envelope>
    </data>
    <!-- Note: More Data sections can be placed here -->
  </cmd>
  <!-- Note: More Commands sections can be placed here -->
</LDM_interface>
```

The <LDM_interface> *Tag*

All commands sent to the LDoms Manager must start with the <LDM_interface> tag. Any document sent into the LDoms Manager must have only one <LDM_interface> tag contained within it. The <LDM_interface> tag must include a version attribute as shown in CODE EXAMPLE 10-1.

The <cmd> *Tag*

Within the <LDM_interface> tag, the document must include at least one <cmd> tag. Each <cmd> section must have only one <action> tag. Use the <action> tag to describe the command to run. Each <cmd> tag must include at least one <data> tag to describe the objects on which the command is to operate.

The <data> *Tag*

Each <data> section contains a description of an object pertinent to the command specified. The format of the data section is based on the XML schema portion of the Open Virtualization Format (OVF) draft specification. That schema defines an <Envelope> section which contains a <References> tag (unused by LDoms) and <Content> and <Section> sections.

For LDoms, the <Content> section is used to identify and describe a particular domain. The domain name in the id= attribute of the <Content> node identifies the domain. Within the <Content> section are one or more <Section> sections describing resources of the domain as needed by the particular command.

If you only need to identify a domain name, then you do not need to use any <Section> tags. Conversely, if no domain identifier is needed for the command, then you do need to provide a <Section> section, describing the resources needed for the command, outside of a <Content> section, but still within the <Envelope> section.

A <data> section does not need to contain an <Envelope> tag in cases where the object information can be inferred. This situation mainly applies to requests for monitoring all objects applicable to an action, and event registration and unregistration requests.

To allow use of the OVF specification's schema to properly define all types of objects, two additional OVF types have been defined:

- <gprop:GenericProperty> tag (See "The GenericProperty XML Schema" on page 231.)
- <Binding> tag (See "Binding_Type XML Schema" on page 232.)

The <gprop:GenericProperty> tag was defined to handle any object's property for which the OVF specification does not have a definition. The property name is defined in the key= attribute of the node and the value of the property is the contents of the node. The <binding> tag is used in the list-bindings subcommand output to define resources that are bound to other resources.

Responses

An outgoing XML response closely matches the structure of the incoming request in terms of the commands and objects included, with the addition of a `<Response>` section for each object and command specified, as well as an overall `<Response>` section for the request. The `<Response>` sections provide status and message information as described in CODE EXAMPLE 10-2. Following is the structure of a response to a basic XML request.

CODE EXAMPLE 10-2 Format of a Response to a Single Command Operating on a Single Object

```
<LDM_interface version="1.0">
  <cmd>
    <action>Place command here</action>
    <data version="3.0">
      <Envelope>
        <References/>
        <!-- Note a <Section> section can be here instead of <Content> -->
        <Content xsi:type="ovf:VirtualSystem_Type" id="Domain name">
          <Section xsi:type="ovf:ResourceAllocationSection_type">
            <Item>
              <rasd:OtherResourceType>
                LDom Resource Type
              </rasd:OtherResourceType>
              <gprop:GenericProperty
              key="Property name">
                Property Value
            </gprop:GenericProperty>
            </Item>
          </Section>
          <!-- Note: More <Section> sections can be placed here -->
        </Content>
      </Envelope>
      <response>
        <status>success or failure</status>
        <resp_msg>Reason for failure</resp_msg>
      </response>
    </data>
    <!-- Note: More Data sections can be placed here -->
    <response>
      <status>success or failure</status>
      <resp_msg>Reason for failure</resp_msg>
    </response>
  </cmd>
  <!-- Note: More Command sections can be placed here -->
  <response>
    <status>success or failure</status>
```

CODE EXAMPLE 10-2 Format of a Response to a Single Command Operating on a Single Object *(Continued)*

```
    <resp_msg>Reason for failure</resp_msg>
  </response>
</LDM_interface>
```

Overall Response

This <response> section, which is the direct child of the <LDM_interface> section, indicates overall success or failure of the entire request. Unless the incoming XML document is malformed, the <response> section includes only a <status> tag. If this response status indicates success, all commands on all objects have succeeded. If this response status is a failure and there is no <resp_msg> tag, then one of the commands included in the original request failed. The <resp_msg> tag is used only to describe some problem with the XML document itself.

Command Response

The <response> section under the <cmd> section alerts the user to success or failure of that particular command. The <status> tag shows if that command succeeds or fails. As with the overall response, if the command fails, the <response> section includes only a <resp_msg> tag if the contents of the <cmd> section of the request is malformed. Otherwise, the failed status means one of the objects the command ran against caused a failure.

Object Response

Finally, each <data> section in a <cmd> section also has a <response> section. This shows if the command being run on this particular object passes or fails. If the status of the response is SUCCESS, there is no <resp_msg> tag in the <response> section. If the status is FAILURE, there are one or more <resp_msg> tags in the <response> field, depending on the errors encountered when running the command against that object. Object errors can result from problems found when running the command, or a malformed or unknown object.

In addition to the <response> section, the <data> section can contain other information. This information is in the same format as an incoming <data> field, describing the object that caused a failure. See "The <data> Tag" on page 179. This additional information is especially useful in the following cases:

- When a command fails against a particular <data> section but passes for any additional <data> sections
- When an empty <data> section is passed into a command and fails for some domains but passes for others

Events

In lieu of polling, you can subscribe to receive event notifications of certain state changes that occur. There are three types of events to which you can subscribe, individually or collectively. See "Event Types" on page 184 for complete details.

Registration and Unregistration

Use an `<LDM_interface>` message to register for events. See "The `<LDM_interface>` Tag" on page 178. The action tag details the type of event for which to register or unregister and the `<data>` section is left empty.

CODE EXAMPLE 10-3 Example Event Registration Request Message

```
<LDM_interface version="1.0">
  <cmd>
    <action>reg-domain-events</action>
    <data version="3.0"/>
  </cmd>
</LDM_interface>
```

The Logical Domains Manager responds with an `<LDM_interface>` response message stating whether the registration or unregistration was successful.

CODE EXAMPLE 10-4 Example Event Registration Response Message

```
<LDM_interface version="1.0">
  <cmd>
    <action>reg-domain-events</action>
    <data version="3.0"/>
      <response>
        <status>success</status>
      </response>
    </data>
    <response>
      <status>success</status>
    </response>
  </cmd>
  <response>
    <status>success</status>
  </response>
</LDM_interface>
```

The action string for each type of event is listed in the events subsection.

The <LDM_event> Messages

Event messages have the same format as an incoming <LDM_interface> message with the exception that the start tag for the message is <LDM_event>. The action tag of the message is the action that was performed to trigger the event. The data section of the message describes the object associated with the event; the details depend on the type of event that occurred.

CODE EXAMPLE 10-5 Example <LDM_event> Notification

```
<LDM_event version='1.0'>
  <cmd>
    <action>Event command here</action>
    <data version='3.0'>
      <Envelope
        <References/>
        Content xsi:type='ovf:VirtualSystem_Type' ovf:id='ldg1'/>
          <Section xsi:type="ovf:ResourceAllocationSection_type"
            <Item>
              <rasd:OtherResourceType>
                LDom Resource Type
              </rasd:OtherResourceType>
              <gprop:GenericProperty
              key="Property name">
                Property Value
              </gprop:GenericProperty>
            </Item>
          </Section>
      </Envelope>
    </data>
  </cmd>
</LDM_event>
```

Event Types

The three types of events to which you can subscribe are:

- Domain events
- Resource events
- Hardware events

All the events correspond to Logical Domains Manager (`ldm`) subcommands.

Domain Events

Domain events describe what actions can be performed directly to a domain. The following table shows the domain events which can be listed in the `<action>` tag in the `<LDM_event>` message.

Domain Events	Domain Events
`add-domain`	`remove-domain`
`bind-domain`	`unbind-domain`
`start-domain`	`stop-domain`
`domain-reset`	`panic-domain`

These events always contain *only* a `<Content>` tag in the OVF data section that describes to which domain the event happened. To register for the domain events, send an `<LDM_interface>` message with the `<action>` tag set to **`reg-domain-events`**. Unregistering for these events requires an `<LDM_interface>` message with the action tag set to **`unreg-domain-events`**.

Resource Events

Resource events occur when resources are added, removed, or changed in any domain. The data section for some of these events contains the <Content> tag with a <Section> tag giving a service name in the OVF data section. The following table shows events which can be listed in the <action> tag in the <LDM_event> message.

Resource Events	Resource Events
add-vdiskserverdevice	remove-vdiskserverdevice
set-vdiskserverdevice	remove-vdiskserver
set-vconscon	remove-vconscon
set-vswitch	remove-vswitch
remove-vdpcs	

The remaining resource events always contain *only* the <Content> tag in the OVF data section that describes to which domain the event happened.

Resource Events	Resource Events	Resource Events
add-vcpu	add-crypto	add-memory
add-io	add-variable	add-vconscon
add-vdisk	add-vdiskserver	add-vnet
add-vswitch	add-vdpcs	add-vdpcc
set-vcpu	set-crypto	set-memory
set-variable	set-vnet	set-vconsole
set-vdisk	remove-vcpu	remove-crypto
remove-memory	remove-io	remove-variable
remove-vdisk	remove-vnet	remove-vdpcc

To register for the resource events, send an <LDM_interface> message with the <action> tag set to **reg-resource-events**. Unregistering for these events requires an <LDM_interface> message with the <action> tag set to **unreg-resource-events**.

Hardware Events

Hardware events pertain to changing the physical system hardware. In the case of LDoms software, the only hardware changes that can be made are those to the service processor (SP) when a user adds, removes, or sets an SP configuration. Currently, the only three events of this type are:

- `add-spconfig`
- `set-spconfig`
- `remove-spconfig`

The hardware events always contain *only* a `<Section>` tag in the OVF data section which describes which SP configuration to which the event is happening. To register for these events, send an `<LDM_interface>` message with the `<action>` tag set to **`reg-hardware-events`**. Unregistering for these events requires an `<LDM_interface>` message with the `<action>` tag set to **`unreg-hardware-events`**.

All Events

You can also register to listen for all three type of events without having to register for each one individually. To register for all three types of events simultaneously, send an `<LDM_interface>` message with the `<action>` tag set to **`reg-all-events`**. Unregistering for these events require an `<LDM_interface>` message with the `<action>` tag set to **`unreg-all-events`**.

Logical Domains Manager Actions

The commands specified in the <action> tag, with the exception of *-*-events commands, correspond to those of the LDoms command-line interface. For details about Logical Domains Manager (ldm) subcommands, refer to the *Logical Domains (LDoms) Manager 1.1 Man Page Guide* or the ldm man page.

Note – The XML interface does *not* support the verb or command *aliases* supported by the Logical Domains Manager CLI.

The supported strings in the <action> tag are as follows:

LDoms Actions	LDoms Actions	LDoms Actions
list-bindings	list-services	list-constraints
list-devices	add-domain	remove-domain
list-domain	start-domain	stop-domain
bind-domain	unbind-domain	add-io
remove-io	add-mau	set-mau
remove-mau	add-memory	set-memory
remove-memory	remove-reconf	add-spconfig
set-spconfig	remove-spconfig	list-spconfig
add-variable	set-variable	remove-variable
list-variable	add-vconscon	set-vconscon
remove-vconscon	set-vconsole	add-vcpu
set-vcpu	remove-vcpu	add-vdisk
remove-vdisk	add-vdiskserver	remove-vdiskserver
add-vdpcc	remove-vdpcc	add-vdpcs
remove-vdpcs	add-vdiskserverdevice	remove-vdiskserverdevice
add-vnet	set-vnet	remove-vnet
add-vswitch	set-vswitch	remove-vswitch
reg-domain-events	unreg-domain-events	reg-resource-events
unreg-resource-events	reg-hardware-events	unreg-hardware-events
reg-all-events	unreg-all-events	migrate-domain

Logical Domains Manager Resources and Properties

Following are the Logical Domains Manager resources and the properties that can be defined for each of those resources. The resources and properties are shown in **bold** type in the XML examples. These examples show resources, not binding output. The constraint output can be used to create input for the Logical Domains Manager actions. The exception to this is domain migration output. See "Domain Migration" on page 201. Each resource is defined in a `<Section>` OVF section and is specified by a `<rasd:OtherResourceType>` tag.

Logical Domain Information (`ldom_info`) Resource

CODE EXAMPLE 10-6 Example `ldom_info` XML Output

```
<Envelope>
  <References/>
  <Content xsi:type="ovf:VirtualSystem_Type" id="primary">
    <Section xsi:type="ovf:ResourceAllocationSection_type">
      <Item>
        <rasd:OtherResourceType>ldom_info</rasd:OtherResourceType>
        <rasd:Address>00:03:ba:d8:ba:f6</rasd:Address>
        <gprop:GenericPropertykey="hostid">83d8baf6</gprop:GenericProperty>
      </Item>
    </Section>
  </Content>
</Envelope>
```

The `ldom_info` resource is always contained within a `<Content>` section. The two properties within the `ldom_info` resource are optional properties:

- `<rasd:Address>` tag, which specifies the MAC address to be assigned to a domain.
- `<gprop:GenericPropertykey="hostid">` tag, which specifies the host ID to be assigned to the domain.

CPU (cpu) Resource

CODE EXAMPLE 10-7 Example cpu XML

```
<Envelope>
  <References/>
  <Content xsi:type="ovf:VirtualSystem_Type" id="ldg1">
    <Section xsi:type="ovf:VirtualHardwareSection_Type">
      <Item>
        <rasd:OtherResourceType>cpu</rasd:OtherResourceType>
        <rasd:AllocationUnits>4</rasd:AllocationUnits>
      </Item>
    </Section>
  </Content>
</Envelope>
```

A cpu resource is always contained within a <Content> section. The only property is the <rasd:AllocationUnits> tag, which specifies the number of virtual CPUs.

MAU (mau) Resource

Note – The mau resource is any LDoms-supported cryptographic unit on an LDoms-supported server. Currently, the two cryptographic units supported are the Modular Arithmetic Unit (MAU) and the Control Word Queue (CWQ).

CODE EXAMPLE 10-8 Example mau XML

```
<Envelope>
  <References/>
  <Content xsi:type="ovf:VirtualSystem_Type" id="ldg1">
    <Section xsi:type="ovf:VirtualHardwareSection_Type">
      <Item>
        <rasd:OtherResourceType>mau</rasd:OtherResourceType>
        <rasd:AllocationUnits>1</rasd:AllocationUnits>
      </Item>
    </Section>
  </Content>
</Envelope>
```

A mau resource is always contained within a <Content> section. The only property is the <rasd:AllocationUnits> tag, which signifies the number of MAUs or other cryptographic units.

Memory (memory) Resource

CODE EXAMPLE 10-9 Example memory XML

```
<Envelope>
  <References/>
  <Content xsi:type="ovf:VirtualSystem_Type" id="ldg1">
    <Section xsi:type="ovf:VirtualHardwareSection_Type">
      <Item>
        <rasd:OtherResourceType>memory</rasd:OtherResourceType>
        <rasd:AllocationUnits>4G</rasd:AllocationUnits>
      </Item>
    </Section>
  </Content>
</Envelope>
```

A memory resource is always contained within a <Content> section. The only property is the <rasd:AllocationUnits> tag, which signifies the amount of memory.

Virtual Disk Server (vds) Resource

CODE EXAMPLE 10-10 Example vds XML

```
<Envelope>
  <References/>
  <Content xsi:type="ovf:VirtualSystem_Type" id="ldg1">
    <Section xsi:type="ovf:VirtualHardwareSection_Type">
      <Item>
        <rasd:OtherResourceType>vds</rasd:OtherResourceType>
        <gprop:GenericProperty key="service_name">vdstmp<
        /gprop:GenericProperty>
      </Item>
    </Section>
  </Content>
</Envelope>
```

A virtual disk server (vds) resource can be in a <Content> section as part of a domain description, or it can appear on its own in an <Envelope> section. The only property is the <gprop:GenericProperty> tag with a key of "service_name" and which contains the name of the vds resource being described.

Virtual Disk Server Volume (vds_volume) Resource

CODE EXAMPLE 10-11 Example vds_volume XML

```
<Envelope>
  <References/>
    <Section xsi:type="ovf:VirtualHardwareSection_Type">
      <Item>
        <rasd:OtherResourceType>vds_volume</rasd:OtherResourceType>
        <gprop:GenericProperty key="vol_name">vdsdev0</gprop:GenericProperty>
        <gprop:GenericProperty key="service_name">primary-vds0<
        /gprop:GenericProperty>
        <gprop:GenericProperty key="block_dev">
          opt/SUNWldm/domain_disks/testdisk1</gprop:GenericProperty>
        <gprop:GenericProperty key="vol_opts">ro<
        /gprop:GenericProperty>
        <gprop:GenericProperty key="mpgroup">mpgroup-name<
        /gprop:GenericProperty>
      </Item>
    </Section>
</Envelope>
```

A vds_volume resource can be in a <Content> section as part of a domain description, or it can appear on its own in an <Envelope> section. It must have <gprop:GenericProperty> tags with the following keys:

- vol_name - Name of the volume
- service_name - Name of the virtual disk server to which this volume is to be bound
- block_dev - File or device name to be associated with this volume

Optionally, a vds_volume resource can also have the following properties:

- vol_opts - One or more of the following, comma-separated, within one string: {ro,slice,excl}
- mpgroup - Name of the multipath (failover) group

Disk (`disk`) Resource

CODE EXAMPLE 10-12 Example `disk` XML

```
<Envelope>
  <References/>
  <Content xsi:type="ovf:VirtualSystem_Type" id="ldg1">
    <Section xsi:type="ovf:VirtualHardwareSection_Type">
      <Item>
        <rasd:OtherResourceType>disk</rasd:OtherResourceType>
        <gprop:GenericProperty key="vdisk_name">vdisk0</gprop:GenericProperty>
        <gprop:GenericProperty key="service_name">primary-vds0<
        /gprop:GenericProperty>
        <gprop:GenericProperty key="vol_name">vdsdev0</gprop:GenericProperty>
        <gprop:GenericProperty key="timeout">60</gprop:GenericProperty>
      </Item>
    </Section>
  </Content>
</Envelope>
```

A `disk` resource is always contained within a `<Content>` section. It must have `<gprop:GenericProperty>` tags with the following keys:

- `vdisk_name` - Name of the virtual disk
- `service_name` - Name of the virtual disk server to which this virtual disk is to be bound
- `vol_name` - Virtual disk service device with which this virtual disk is to be associated

Optionally, the `disk` resource can also have the following property:

- `timeout` - Timeout value in seconds for establishing a connection between a virtual disk client (`vdc`) and a virtual disk server (`vds`). If there are multiple virtual disk (`vdisk`) paths, then the `vdc` can try to connect to a different `vds`, and the timeout ensures that a connection to any `vds` is established within the specified amount of time.

Virtual Switch (vsw) Resource

CODE EXAMPLE 10-13 Example vsw XML

```
<Envelope>
  <References/>
  <Content xsi:type="ovf:VirtualSystem_Type" id="ldg1">
    <Section xsi:type="ovf:VirtualHardwareSection_Type">
      <Item>
        <rasd:OtherResourceType>vsw</rasd:OtherResourceType>
        <gprop:GenericProperty key="service_name">vsw1-ldg1<
        /gprop:GenericProperty>
        <gprop:GenericProperty key="dev-path">bge0</gprop:GenericProperty>
        <rasd:Address>00:14:4f:fc:00:01</rasd:Address>
        <gprop:GenericProperty key="mode">sc</gprop:GenericProperty>
        <gprop:GenericProperty key="pvid">12345678</gprop:GenericProperty>
        <gprop:GenericProperty key="vid">87654321</gprop:GenericProperty>
      </Item>
    </Section>
  </Content>
</Envelope>
```

A vsw resource can be either in a <Content> section as part of a domain description, or it can appear on its own in an <Envelope> section. It must have <gprop:GenericProperty> tags with the following keys:

- service_name - Name to be assigned to the virtual switch.
- dev-path - Path of the network device to be associated with this virtual switch

Optionally, the vsw resource can also have the following properties:

- <rasd:Address> - Assigns a MAC address to the virtual switch
- pvid - Port virtual local area network (VLAN) identifier (ID) indicates the VLAN of which the virtual network needs to be a member, in untagged mode.
- vid - Virtual local area network (VLAN) identifier (ID) indicates the VLAN of which a virtual network and virtual switch need to be a member, in tagged mode.
- mode - sc for SunCluster heartbeat support.

Network (network) Resource

CODE EXAMPLE 10-14 Example network XML

```
<Envelope>
  <References/>
  <Content xsi:type="ovf:VirtualSystem_Type" id="ldg1">
    <Section xsi:type="ovf:VirtualHardwareSection_Type">
      <Item>
        <rasd:OtherResourceType>network</rasd:OtherResourceType>
        <gprop:GenericProperty key="vnet_name">ldg1-vnet0<
        /gprop:GenericProperty>
        <gprop:GenericProperty key="service_name">primary-vsw0<
        /gprop:GenericProperty>
        <rasd:Address>00:14:4f:fc:00:01</rasd:Address>
      </Item>
    </Section>
  </Content>
</Envelope>
```

A network resource is always contained within a <Content> section. It must have <gprop:GenericProperty> tags with the following keys:

- vnet_name - Name of the virtual network (vnet)
- service_name - Name of the virtual switch (vswitch) to which this virtual network is to be bound

Optionally, the network resource can also have the following properties:

- <rasd:Address> - Assigns a MAC address to the virtual switch
- pvid - Port virtual local area network (VLAN) identifier (ID) indicates the VLAN of which the virtual network needs to be a member, in untagged mode.
- vid - Virtual local area network (VLAN) identifier (ID) indicates the VLAN of which a virtual network and virtual switch need to be a member, in tagged mode.
- mode - hybrid to enable hybrid I/O for that virtual network.

Virtual Console Concentrator (vcc) Resource

CODE EXAMPLE 10-15 Example vcc XML

```
<Envelope>
  <References/>
  <Content xsi:type="ovf:VirtualSystem_Type" id="ldg1">
    <Section xsi:type="ovf:VirtualHardwareSection_Type">
      <Item>
        <rasd:OtherResourceType>vcc</rasd:OtherResourceType>
        <gprop:GenericProperty key="service_name">vcc1</gprop:GenericProperty>
        <gprop:GenericProperty key="min_port">6000</gprop:GenericProperty>
        <gprop:GenericProperty key="max_port">6100</gprop:GenericProperty>
      </Item>
    </Section>
  </Content>
</Envelope>
```

A vcc resource can be either in a <Content> section as part of a domain description, or it can appear on its own in an <Envelope> section. It can have <gprop:GenericProperty> tags with the following keys:

- service_name - Name to be assigned to the virtual console concentrator service
- min_port - Minimum port number to be associated with this vcc
- max_port - Maximum port number to be associated with this vcc

Variable (`var`) Resource

CODE EXAMPLE 10-16 Example `var` XML

```
<Envelope>
  <References/>
  <Content xsi:type="ovf:VirtualSystem_Type" id="ldg1">
    <Section xsi:type="ovf:VirtualHardwareSection_Type">
      <Item>
        <rasd:OtherResourceType>var</rasd:OtherResourceType>
        <gprop:GenericProperty key="name">test_var</gprop:GenericProperty>
        <gprop:GenericProperty key="value">test1</gprop:GenericProperty>
      </Item>
    </Section>
  </Content>
</Envelope>
```

A `var` resource is always contained within a `<Content>` section. It can have `<gprop:GenericProperty>` tags with the following keys:

- `name` - Name of the variable
- `value` - Value of the variable

Physical I/O Device (`physio_device`) Resource

CODE EXAMPLE 10-17 Example `physio_device` XML

```
<Envelope>
  <References/>
  <Content xsi:type="ovf:VirtualSystem_Type" id="ldg1">
    <Section xsi:type="ovf:VirtualHardwareSection_Type">
      <Item>
        <rasd:OtherResourceType>physio_device</rasd:OtherResourceType>
        <gprop:GenericProperty key="name">pci@780</gprop:GenericProperty>
      </Item>
    </Section>
  </Content>
</Envelope>
```

A `physio_device` resource is always contained within a `<Content>` section. The only property is the `<gprop:GenericProperty>` tag with the following key property value:

- `name` - Name of the I/O device being described

SP Configuration (spconfig) Resource

CODE EXAMPLE 10-18 Example spconfig XML

```
<Envelope>
    <Section xsi:type="ovf:ResourceAllocationSection_type">
      <Item>
        <rasd:OtherResourceType>spconfig</rasd:OtherResourceType>
        <gprop:GenericProperty key="spconfig_name">primary<
        /gprop:GenericProperty>
        <gprop:GenericProperty key="spconfig_status">current<
        /gprop:GenericProperty>
      </Item>
    </Section>
</Envelope>
```

A service processor (SP) configuration (spconfig) resource always appears on its own in an <Envelope> section. It can have <gprop:GenericProperty> tags with the following keys

- spconfig_name - Name of a configuration to be stored on the SP
- spconfig_status - The current status of a particular SP configuration. This property is used in the output of an ldm list-spconfig command.

Virtual Data Plane Channel Service (vdpcs) Resource

CODE EXAMPLE 10-19 Example vdpcs XML

```
<Envelope>
  <References/>
  <Content xsi:type="ovf:VirtualSystem_Type" id="ldg1">
    <Section xsi:type="ovf:VirtualHardwareSection_Type">
      <Item>
        <rasd:OtherResourceType>vdpcs</rasd:OtherResourceType>
        <gprop:GenericProperty key="service_name">dg1-vdpcs<
        /gprop:GenericProperty>
      </Item>
    </Section>
  </Content>
</Envelope>
```

This resource is only of interest in a Netra DPS environment. A `vdpcs` resource can be either in a `<Content>` section as part of a domain description, or it can appear on its own in an `<Envelope>` section. The only property is the `<gprop:GenericProperty>` tag with the following key property value:

- `service_name` - Name of the virtual data plane channel service (`vdpcs`) resource being described

Virtual Data Plane Channel Client (vdpcc) Resource

CODE EXAMPLE 10-20 Example vdpcc XML

```
<Envelope>
  <References/>
  <Content xsi:type="ovf:VirtualSystem_Type" id="ldg1">
    <Section xsi:type="ovf:VirtualHardwareSection_Type">
      <Item>
        <rasd:OtherResourceType>vdpcc</rasd:OtherResourceType>
        <gprop:GenericProperty key="vdpcc_name">vdpcc</gprop:GenericProperty>
        <gprop:GenericProperty key="service_name">ldg1-vdpcs<
        /gprop:GenericProperty>
      </Item>
    </Section>
  </Content>
</Envelope>
```

This resource is only of interest in a Netra DPS environment. A virtual data plane channel client resource is always contained within a <Content> section. It can have <gprop:GenericProperty> tags with the following keys:

- vdpcc_name - Name of the virtual data plane channel client (vdpcc)
- service_name - Name of the virtual data plane channel service vdpcs to which this vdpcc is to be bound

Console (`console`) Resource

CODE EXAMPLE 10-21 Example `console` XML

```
<Envelope>
  <References/>
  <Content xsi:type="ovf:VirtualSystem_Type" id="ldg1">
    <Section xsi:type="ovf:VirtualHardwareSection_Type">
      <Item>
        <rasd:OtherResourceType>console</rasd:OtherResourceType>
        <gprop:GenericProperty key="port">6000</gprop:GenericProperty>
        <gprop:GenericProperty key="service_name">vcc2</gprop:GenericProperty>
        <gprop:GenericProperty key="group">group-name<
        /gprop:GenericProperty>
      </Item>
    </Section>
  </Content>
</Envelope>
```

A console `resource` is always contained within a `<Content>` section. It can have `<gprop:GenericProperty>` tags with the following keys:

- port - Port to which to change this virtual console (`console`)
- `service_name` - Virtual console concentrator (`vcc`) service to which to bind this `console`
- `group` - Name of the group to which to bind this `console`

Domain Migration

This example shows what is contained in the <data> section for a migrate-domain subcommand.

CODE EXAMPLE 10-22 Example migrate-domain <data> Section

```
<Envelope>
  <References/>
  <Content xsi:type="ovf:VirtualSystem_Type" ovf:id="ldg1"/>
  <Content xsi:type="ovf:VirtualSystem_Type" ovf:id="ldg1"/>
    <Section xsi:type="ovf:ResourceAllocationSection_Type">
      <Item>
        <rasd:OtherResourceType>ldom_info</rasd:OtherResourceType>
        <gprop:GenericProperty key=
"target">target-host</gprop:GenericProperty>
        <gprop:GenericProperty key=
"username">user-name</gprop:GenericProperty>
        <gprop:GenericProperty key="password">password</gprop:GenericProperty>
      <Item>
    </Section>
  </Content>
</Envelope>
```

Where:

- First <Content> node (without an <ldom_info> section) is the source domain to migrate.
- Second <Content> node (with an <ldom_info> section) is the target domain to which to migrate. The source and target domain names can be the same.
- The <ldom_info> section for the target domain describes the machine to which to migrate and the details needed to migrate to that machine:
  - target-host is the target machine to which to migrate.
  - user-name is the login user name for the target machine. Must be SASL 64-bit encoded.
  - password is the password to use for logging into the target machine. Must be SASL 64-bit encoded.

Note – The Logical Domains Manager uses sasl_decode64() to decode the target user name and password and uses sasl_encode64() to encode these values. SASL 64 encoding is equivalent to base64 encoding.

APPENDIX **A**

XML Schemas

This appendix provides various XML schemas for your use with the Logical Domains Manager.

`LDM_interface` XML Schema

This schema is a snapshot of the Open Virtualization Format (OVF) Draft Specification version 0.98

CODE EXAMPLE A-1 `LDM_interface` XML Schema

```
<?xml version="1.0"?>
xs:schema
     xmlns:ovf="/var/opt/SUNWldom/envelope"
     xmlns:xs="http://www.w3.org/2001/XMLSchema">
     <xs:import namespace="/var/opt/SUNWldom/envelope" schemaLocation=
"ovf-envelope.xsd"/>

     <xs:annotation>
          <xs:documentation>
               Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
               Use is subject to license terms.
          </xs:documentation>
     </xs:annotation>

<!--
==================
Type Definitions
==================
-->
  <xs:simpleType name="statusStringType">
    <xs:restriction base="xs:string">
```

CODE EXAMPLE A-1 `LDM_interface` XML Schema *(Continued)*

```
      <xs:enumeration value="success"/>
      <xs:enumeration value="failure"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="responseType">
    <xs:sequence>
      <xs:element name="status" type="statusStringType"/>
      <xs:element name="resp_msg" type="xs:string"
        minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <!-- LDM interface document -->
  <xs:element name="LDM_interface">
    <xs:complexType>
      <xs:sequence>

        <!-- START cmd -->
        <xs:element name="cmd" minOccurs="1" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="action" type="xs:string"
                minOccurs="0"/>

              <!-- START data -->
              <xs:element name="data" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:choice minOccurs="1" maxOccurs="unbounded">

                    <!--OVF Evelope Version 0.9 -->
                    <xs:element name="Envelope" type="ovf:Envelope_Type"/>
                    <!-- DATA response -->
                    <xs:element name="response" type="responseType"
                      minOccurs="0" maxOccurs="1"/>
                  </xs:choice>
                  <xs:attribute name="version" type="xs:string" use="required"/>
                </xs:complexType>
              </xs:element> <!-- END data -->

              <!-- CMD response -->
              <xs:element name="response" type="responseType"
                minOccurs="0" maxOccurs="1"/>

            </xs:sequence>
          </xs:complexType>
        </xs:element> <!-- END cmd -->
```

CODE EXAMPLE A-1 `LDM_interface` XML Schema *(Continued)*

```
        <!-- DOCUMENT response -->
        <xs:element name="response" type="responseType"
          minOccurs="0" maxOccurs="1"/>

      </xs:sequence>
      <xs:attribute name="version" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element> <!-- LDM interface document -->

</xs:schema>
```

LDM_Event XML Schema

CODE EXAMPLE A-2 LDM_Event XML Schema

```
<?xml version="1.0"?>
<xs:schema
    xmlns:ovf="/var/opt/SUNWldom/envelope"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:import namespace="/var/opt/SUNWldom/envelope" schemaLocation=
"ovf-envelope.xsd"/>

    <xs:annotation>
      <xs:documentation>
        Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
        Use is subject to license terms.
      </xs:documentation>
    </xs:annotation>

  <!-- LDM interface document -->
  <xs:element name="LDM_event">
    <xs:complexType>
      <xs:sequence>

       <!-- START cmd -->
       <xs:element name="cmd" minOccurs="1" maxOccurs="unbounded">
         <xs:complexType>
           <xs:sequence>
             <xs:element name="action" type="xs:string"
               minOccurs="0"/>

             <!-- START data -->
             <xs:element name="data" minOccurs="0" maxOccurs="unbounded">
               <xs:complexType>
                 <xs:choice minOccurs="1" maxOccurs="unbounded">

                 <!--OVF Evelope Version 0.9 -->
                 <xs:element name="Envelope" type="ovf:Envelope_Type"/>

                 </xs:choice>
                 <xs:attribute name="version" type="xs:string" use="required"/>
               </xs:complexType>
             </xs:element> <!-- END data -->

            </xs:sequence>
          </xs:complexType>
```

CODE EXAMPLE A-2 `LDM_Event` XML Schema *(Continued)*

```
        </xs:element> <!-- END cmd -->

      </xs:sequence>
      <xs:attribute name="version" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element> <!-- LDM interface document -->

</xs:schema>
```

The `ovf-envelope.xsd` Schema

CODE EXAMPLE A-3 The `ovf-envelope.xsd` Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  targetNamespace="/var/opt/SUNWldom/envelope"
  xmlns:ovf="/var/opt/SUNWldom/envelope"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <!-- Include virtual hardware schema -->
  <xs:include schemaLocation="./ovf-section.xsd"/>
  <xs:include schemaLocation="./cim-virtualhardware.xsd"/>
  <xs:include schemaLocation="./ovf-core.xsd"/>

  <!-- Root element of a OVF package-->
  <xs:element name="Envelope" type="ovf:Envelope_Type"/>

  <xs:complexType name="Envelope_Type">
    <xs:sequence>
      <!--- References to all external files -->
      <xs:element name="References" type="ovf:References_Type"/>

      <!-- Package level meta-data -->
      <xs:element name="Section" type="ovf:Section_Type" minOccurs="0"
maxOccurs="unbounded"/>

      <!-- Content. A virtual machine or a vService -->
      <xs:element name="Content" type="ovf:Entity_Type" minOccurs="0"
maxOccurs="unbounded"/>

      <xs:any namespace="##targetNamespace" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
      <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="signed" type="xs:boolean" use="optional"/>
    <xs:attribute name="manifest" type="xs:boolean" use="optional"/>
    <xs:anyAttribute namespace="##any"/>
  </xs:complexType>

  <xs:complexType name="References_Type">
    <xs:sequence>
      <xs:element name="File" type="ovf:File_Type" minOccurs="0" maxOccurs=
"unbounded"/>
```

CODE EXAMPLE A-3 The `ovf-envelope.xsd` Schema *(Continued)*

```
      <xs:any namespace="##targetNamespace" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
      <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
<xs:anyAttribute namespace="##any"/>
  </xs:complexType>

  <!--Type for an external reference to a resource -->
  <xs:complexType name="File_Type">
    <xs:sequence>
      <xs:any namespace="##targetNamespace" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
      <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>

    <!-- Reference key used in other parts of the package -->
    <xs:attribute name="id" type="xs:string" use="required"/>
    <!-- Same as using a single part element -->
    <xs:attribute name="href" type="xs:string" use="required"/>
    <!-- Size in bytes of the files (if known) -->
    <xs:attribute name="size" type="xs:integer" use="optional"/>
    <!-- Estimated size in bytes of the files (if a good guess is known) -->
    <xs:attribute name="estSize" type="xs:integer" use="optional"/>
    <!-- Compression type (gzip or bzip2) -->
    <xs:attribute name="compression" type="xs:string" use="optional"/>
    <!-- Chunk size (except of last chunk) -->
    <xs:attribute name="chunkSize" type="xs:long" use="optional"/>

    <xs:anyAttribute namespace="##any"/>
  </xs:complexType>

  <!-- Base class for an entity -->
  <xs:complexType name="Entity_Type" abstract="true">
    <xs:sequence>
      <xs:element name="Info" type="ovf:Info_Type" minOccurs="0" maxOccurs=
"unbounded"/>
      <xs:element name="Section" type="ovf:Section_Type" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:string" use="required"/>
  </xs:complexType>

  <!-- A Virtual Machine Entity -->
  <xs:complexType name="VirtualSystem_Type">
<xs:complexContent>
```

CODE EXAMPLE A-3 The `ovf-envelope.xsd` Schema *(Continued)*

```
      <xs:extension base="ovf:Entity_Type"> </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <!-- A Composite Service -->
  <xs:complexType name="VirtualSystemCollection_Type">
    <xs:complexContent>
      <xs:extension base="ovf:Entity_Type">
       <xs:sequence>
        <xs:element name="Content" type="ovf:Entity_Type" minOccurs="0"
maxOccurs="unbounded"/>
        <xs:any namespace="##targetNamespace" processContents="lax" minOccurs=
"0" maxOccurs="unbounded"/>
        <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
       </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
```

The `ovf-section.xsd` Schema

CODE EXAMPLE A-4 The `ovf-section.xsd` Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  targetNamespace="/var/opt/SUNWldom/envelope"
  xmlns:ovf="/var/opt/SUNWldom/envelope"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:import namespace="http://www.w3.org/XML/1998/namespace"
    schemaLocation="http://www.w3.org/2001/xml.xsd"/>

  <!-- The base class for a section. Subclassing this is the most common form of
extensibility -->
  <xs:complexType name="Section_Type" abstract="true">
    <xs:sequence>
      <!-- The info element specifies the meaning of the section. This is
typically shown if the section is not understood by the importer -->
      <xs:element name="Info" type="ovf:Info_Type" minOccurs="0" maxOccurs=
"unbounded"/>
    </xs:sequence>
  <!-- Whether the import should fail or not, if the section is not understood -->
    <xs:attribute name="required" type="xs:boolean" use="optional"/>
    <xs:anyAttribute namespace="##any"/>
    <!-- Subtypes defines more specific elements -->
  </xs:complexType>

  <!-- A basic type for a localizable string -->
  <xs:complexType name="Info_Type">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute ref="xml:lang"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:schema>
```

The `ovf-core.xsd` Schema

CODE EXAMPLE A-5 The `ovf-core.xsd` Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  targetNamespace="/var/opt/SUNWldom/envelope"
  xmlns:ovf="/var/opt/SUNWldom/envelope"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:include schemaLocation="ovf-section.xsd"/>
  <xs:import namespace="http://www.w3.org/XML/1998/namespace"
    schemaLocation="http://www.w3.org/2001/xml.xsd"/>

  <!-- A user defined annotation on an entity -->
  <xs:complexType name="AnnotationSection_Type">
    <xs:complexContent>
      <xs:extension base="ovf:Section_Type">
        <xs:sequence>
          <!-- Several localized annotations can be included -->
          <xs:element name="Annotation" type="ovf:Info_Type" minOccurs="0"
maxOccurs="unbounded"/>
          <xs:any namespace="##targetNamespace" processContents="lax"
minOccurs="0" maxOccurs="unbounded"/>
          <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:anyAttribute namespace="##any"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <!-- Product information about a virtual appliance -->
  <xs:complexType name="ProductSection_Type">
    <xs:complexContent>
      <xs:extension base="ovf:Section_Type">
        <xs:sequence>
          <xs:element name="Product" type="ovf:Info_Type" minOccurs="0"
maxOccurs="unbounded"/>
          <xs:element name="Vendor" type="ovf:Info_Type" minOccurs="0"
maxOccurs="unbounded"/>
          <xs:element name="Version" type="xs:string" minOccurs="0"/>
          <xs:element name="Full-version" type="xs:string" minOccurs="0"/>
          <xs:element name="ProductUrl" type="xs:string" minOccurs="0"/>
          <xs:element name="VendorUrl" type="xs:string" minOccurs="0"/>
          <xs:element name="AppUrl" type="xs:string" minOccurs="0"/>
```

CODE EXAMPLE A-5 The `ovf-core.xsd` Schema *(Continued)*

```
          <xs:any namespace="##targetNamespace" processContents="lax"
minOccurs="0" maxOccurs="unbounded"/>
          <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:anyAttribute namespace="##any"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <!-- Configuration parameters that can be passed to the virtual machine for
application-level configuration -->
  <xs:complexType name="PropertySection_Type">
    <xs:complexContent>
      <xs:extension base="ovf:Section_Type">
        <xs:sequence>
          <xs:element name="Property" maxOccurs="unbounded">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="Description" type="ovf:Info_Type" minOccurs=
"0" maxOccurs="unbounded"/>
                <xs:any namespace="##targetNamespace" processContents="lax"
minOccurs="0" maxOccurs="unbounded"/>
                <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
              </xs:sequence>
              <xs:attribute name="key" type="xs:string"/>
              <xs:attribute name="type" type="xs:string"/>
              <xs:attribute name="configurableByUser" type="xs:boolean" use=
"optional"/>
              <xs:attribute name="configurableAtRuntime" type="xs:boolean" use=
"optional"/>
              <xs:attribute name="defaultValue" type="xs:string" use=
"optional"/>
              <xs:anyAttribute namespace="##any"/>
            </xs:complexType>
          </xs:element>
          <xs:any namespace="##targetNamespace" processContents="lax"
minOccurs="0" maxOccurs="unbounded"/>
          <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
        </xs:sequence>
        <!-- A comma-separated list of transports that are supported by the
virtual machine to access the OVF environment. -->
        <xs:attribute name="transport" type="xs:string" use="optional"/>
        <xs:anyAttribute namespace="##any"/>
      </xs:extension>
```

CODE EXAMPLE A-5 The `ovf-core.xsd` Schema *(Continued)*

```
    </xs:complexContent>
  </xs:complexType>

  <!-- Provides descriptions for the logical networks used within the package.
These descriptions are typically used as an aid when the package is deployed. -->
  <xs:complexType name="NetworkSection_Type">
    <xs:complexContent>
      <xs:extension base="ovf:Section_Type">
        <xs:sequence>
          <xs:element name="Network" maxOccurs="unbounded">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="Description"
type="ovf:Info_Type" minOccurs="0" maxOccurs="unbounded"/>
                <xs:any namespace="##targetNamespace" processContents="lax"
minOccurs="0" maxOccurs="unbounded"/>
                <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
              </xs:sequence>
              <xs:attribute name="name" type="xs:string" use="required"/>
              <xs:anyAttribute namespace="##any"/>
            </xs:complexType>
          </xs:element>
          <xs:any namespace="##targetNamespace" processContents="lax"
minOccurs="0" maxOccurs="unbounded"/>
          <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:anyAttribute namespace="##any"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <!-- Provides meta-information description of the virtual disks in the package
-->
  <xs:complexType name="DiskSection_Type">
    <xs:complexContent>
      <xs:extension base="ovf:Section_Type">
        <xs:sequence>
          <xs:element name="Disk" type="ovf:VirtualDiskDesc_Type" minOccurs="0"
maxOccurs="unbounded"/>
          <xs:any namespace="##targetNamespace" processContents="lax"
minOccurs="0" maxOccurs="unbounded"/>
          <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:anyAttribute namespace="##any"/>
```

CODE EXAMPLE A-5 The `ovf-core.xsd` Schema *(Continued)*

```
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <!-- Disk -->
  <xs:complexType name="VirtualDiskDesc_Type">
    <!-- A logical ID for the virtual disk within this package -->
    <xs:attribute name="diskId" type="xs:string" use="required"/>
    <!-- A file reference to the virtual disk file. If this is not specified a
blank virtual disk is created of the given size -->
    <xs:attribute name="fileRef" type="xs:string" use="optional"/>
    <!-- Capacity in bytes. The capacity can be specified as either a size or as
a reference to a property using $(property_name) -->
    <xs:attribute name="capacity" type="xs:string" use="required"/>
    <!-- Format of the disk. The format is an URL that identifies the disk type,
e.g., http://www.vmware.com/format/vmdk.html#sparse -->
    <xs:attribute name="format" type="xs:string" use="required"/>
    <!-- Populated size of disk. This is an estimation of how much storage the
disk needs if backed by a non pre-allocated (aka. sparse) disk. This size does
not take the meta-data into account used by a sparse disk. -->
    <xs:attribute name="populatedSize" type="xs:long" use="optional"/>
    <!-- Reference to a potential parent disk -->
    <xs:attribute name="parentRef" type="xs:string" use="optional"/>
  </xs:complexType>

  <!-- CPU Architecture requirements for the guest software. -->
  <xs:complexType name="CpuCompatibilitySection_Type">
    <xs:complexContent>
      <xs:extension base="ovf:Section_Type">
        <xs:sequence>
          <xs:element name="Level" maxOccurs="unbounded">
            <xs:complexType>
              <xs:attribute name="level" type="xs:int" use="optional"/>
              <xs:attribute name="eax" type="xs:string" use="optional"/>
              <xs:attribute name="ebx" type="xs:string" use="optional"/>
              <xs:attribute name="ecx" type="xs:string" use="optional"/>
              <xs:attribute name="edx" type="xs:string" use="optional"/>
            </xs:complexType>
          </xs:element>
          <xs:any namespace="##targetNamespace" processContents="lax"
minOccurs="0" maxOccurs="unbounded"/>
          <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="Vendor" type="xs:string"/>
        <xs:anyAttribute namespace="##any"/>
      </xs:extension>
    </xs:complexContent>
```

CODE EXAMPLE A-5 The `ovf-core.xsd` Schema *(Continued)*

```
  </xs:complexType>

  <!-- Specification of the operating system installed in the guest -->
  <xs:complexType name="OperatingSystemSection_Type">
    <xs:complexContent>
      <xs:extension base="ovf:Section_Type">
        <xs:sequence>
          <xs:element name="Description" type="ovf:Info_Type" minOccurs="0"
maxOccurs="unbounded"/>
          <xs:any namespace="##targetNamespace" processContents="lax"
minOccurs="0" maxOccurs="unbounded"/>
          <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
        </xs:sequence>
        <!-- The IDs are the enumeration used in CIM_OperatingSystem_Type -->
        <xs:attribute name="id" type="xs:string"/>
        <xs:anyAttribute namespace="##any"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <!-- End-User License Agreement -->
  <xs:complexType name="EulaSection_Type">
    <xs:complexContent>
      <xs:extension base="ovf:Section_Type">
        <xs:sequence>
          <!-- Contains the license agreement in plain text. Several different
locales can be specified -->
          <xs:element name="License" type="ovf:Info_Type" minOccurs="1"
maxOccurs="unbounded"/>
          <xs:any namespace="##targetNamespace" processContents="lax"
minOccurs="0" maxOccurs="unbounded"/>
          <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:anyAttribute namespace="##any"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <!-- For a VirtualSystemCollection, this section is used to specify the order
in which the contained entities are to be powered on. --
  <xs:complexType name="StartupSection_Type">
    <xs:complexContent>
      <xs:extension base="ovf:Section_Type">
        <xs:sequence>
          <xs:element name="item" minOccurs="0" maxOccurs="unbounded">
```

CODE EXAMPLE A-5 The `ovf-core.xsd` Schema *(Continued)*

```
            <xs:complexType>
              <!-- Id of entity in collection -->
              <xs:attribute name="id" type="xs:string"/>
              <!-- Startup order. Entities are started up starting with
lower-numbers first. Items with same order identifier may be started up
concurrently or in any order. The order is reversed for shutdown. -->
              <xs:attribute name="order" type="xs:int"/>
              <!-- Delay in seconds to wait for the power on to complete -->
              <xs:attribute name="startDelay" type="xs:int"/>
              <!-- Whether to resume power-on sequence, once the guest reports
ok. -->
              <xs:attribute name="waitingForGuest" type="xs:boolean"/>
              <!-- Delay in seconds to wait for the power on to complete -->
              <xs:attribute name="stopDelay" type="xs:int"/>
              <!-- Stop action to use. Valid values are: 'powerOn? (default),
'none?. -->
              <xs:attribute name="startAction" type="xs:string"/>
              <!-- Stop action to use. Valid values are: 'powerOff' (default),
'guestShutdown', 'suspend'. -->
              <xs:attribute name="stopAction" type="xs:string"/>
              <xs:anyAttribute namespace="##any"/>
            </xs:complexType>
          </xs:element>
          <xs:any namespace="##targetNamespace" processContents="lax"
minOccurs="0" maxOccurs="unbounded"/>
          <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
        </xs:sequence>
        <!-- A comma-separated list of transports that the virtual machine
supports to provide feedback. -->
        <xs:anyAttribute namespace="##any"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <!-- If this section is present, it indicates that the virtual machine needs
to be initially booted to install and configure the software. -->
  <xs:complexType name="InstallSection_Type">
    <xs:complexContent>
      <xs:extension base="ovf:Section_Type">
        <xs:sequence>
          <xs:any namespace="##targetNamespace" processContents="lax"
minOccurs="0" maxOccurs="unbounded"/>
          <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
        </xs:sequence>
```

CODE EXAMPLE A-5 The `ovf-core.xsd` Schema *(Continued)*

```
        <!-- A comma-separated list of transports that the virtual machine
supports to provide feedback. -->
        <xs:attribute name="transport" type="xs:string"/>
        <xs:anyAttribute namespace="##any"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
```

The ovf-virtualhardware.xsc Schema

CODE EXAMPLE A-6 The ovf-virtualhardware.xsc Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  targetNamespace="/var/opt/SUNWldom/envelope"
  xmlns:ovf="/var/opt/SUNWldom/envelope"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:vssd="/var/opt/SUNWldom/CIM_VirtualSystemSettingData
  xmlns:rasd="/var/opt/SUNWldom/CIM_ResourceAllocationSettingData">

  <xs:import namespace="http://www.w3.org/XML/1998/namespace" schemaLocation=
"http://www.w3.org/2001/xml.xsd"/>

  <xs:include schemaLocation="ovf-section.xsd"/>

  <xs:import namespace="/var/opt/SUNWldom/CIM_VirtualSystemSettingData"
schemaLocation="cim-vssd.xsd"/>
  <xs:import namespace="/var/opt/SUNWldom/CIM_ResourceAllocationSettingData"
schemaLocation="cim-rasd.xsd"/>

  !-- Specifies the virtual hardware for a virtual machine -->
  <xs:complexType name="VirtualHardwareSection_Type">
    <xs:complexContent>
      <xs:extension base="ovf:Section_Type">
        <xs:sequence>
          <xs:element name="System"
type="vssd:CIM_VirtualSystemSettingData_Type" minOccurs="0"/>
          <xs:element name="Item"
type="rasd:CIM_ResourceAllocationSettingData_Type" minOccurs="0" maxOccurs=
"unbounded"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <!-- Specifies a section for resource constraints on a VirtualSystemCollection
-->
  <xs:complexType name="ResourceAllocationSection_Type">
    <xs:complexContent>
      <xs:extension base="ovf:Section_Type">
        <xs:sequence>
          <xs:element name="Item"
```

CODE EXAMPLE A-6 The `ovf-virtualhardware.xsc` Schema *(Continued)*

```
type="rasd:CIM_ResourceAllocationSettingData_Type" minOccurs="0" maxOccurs=
"unbounded"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
```

The cim-rasd.xsd Schema

CODE EXAMPLE A-7 The cim-rasd.xsd Schema

```
<?xml version='1.0' encoding='utf-8'?>
<xs:schema
  targetNamespace="/var/opt/SUNWldom/CIM_ResourceAllocationSettingData"
  xmlns:class="/var/opt/SUNWldom/CIM_ResourceAllocationSettingData"
  xmlns:cim="/var/opt/SUNWldom/common"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:import namespace="/var/opt/SUNWldom/common" schemaLocation=
"cim-common.xsd"/>

  <xs:element name="Caption" nillable="true" type="cim:cimString"/>

  <xs:element name="Description" nillable="true" type="cim:cimString"/

  <xs:element name="InstanceId" nillable="true" type="cim:cimString"/>

  <xs:element name="ResourceType" nillable="true">
    <xs:complexType>
      <xs:simpleContent>
        <xs:restriction base="xs:anyType">
          <xs:simpleType>
            <xs:union>
              <xs:simpleType>
                <xs:restriction base="xs:unsignedShort">
                  <xs:enumeration value="1"/> <!-- Other -->
                  <xs:enumeration value="2"/> <!-- Computer System -->
                  <xs:enumeration value="3"/> <!-- Processor-->
                  <xs:enumeration value="4"/> <!-- Memory-->
                  <xs:enumeration value="5"/> <!-- IDE Controller -->
                  <xs:enumeration value="6"/> <!-- Parallel SCSI HBA -->
                  <xs:enumeration value="7"/> <!-- FC HBA -->
                  <xs:enumeration value="8"/> <!-- iSCSI HBA -->
                  <xs:enumeration value="9"/> <!-- IB HCA -->
                  <xs:enumeration value="10"/> <!-- Ethernet Adapter -->
                  <xs:enumeration value="11"/> <!-- Other Network Adapter -->
                  <xs:enumeration value="12"/> <!-- I/O Slot -->
                  <xs:enumeration value="13"/> <!-- I/O Device -->
                  <xs:enumeration value="14"/> <!-- Floppy Drive -->
                  <xs:enumeration value="15"/> <!-- CD Drive -->
                  <xs:enumeration value="16"/> <!-- DVD drive -->
                  <xs:enumeration value="17"/> <!-- Disk Drive -->
                  <xs:enumeration value="18"/> <!-- Tape Drive -->
```

CODE EXAMPLE A-7 The `cim-rasd.xsd` Schema *(Continued)*

```
                    <xs:enumeration value="19"/> <!-- Storage Extent -->
                    <xs:enumeration value="20"/> <!-- Other storage device -->
                    <xs:enumeration value="21"/> <!-- Serial port -->
                    <xs:enumeration value="22"/> <!-- Parallel port -->
                    <xs:enumeration value="23"/> <!-- USB Controller -->
                    <xs:enumeration value="24"/> <!-- Graphics controller -->
                    <xs:enumeration value="25"/> <!-- IEEE 1394 Controller -->
                    <xs:enumeration value="26"/> <!-- Partitionable Unit -->
                    <xs:enumeration value="27"/> <!-- Base Partitionable Unit -->
                    <xs:enumeration value="28"/> <!-- Power Supply -->
                    <xs:enumeration value="29"/> <!-- Cooling Device -->
                    <xs:enumeration value="29"/> <!-- Cooling Device -->
                    <xs:enumeration value="31"/> <!-- PS2 Controller -->
                    <xs:enumeration value="32"/> <!-- SIO Controller -->
                    <xs:enumeration value="33"/> <!-- Keyboard -->
                    <xs:enumeration value="34"/> <!-- Pointing Device -->
                  </xs:restriction>
                </xs:simpleType>
                <xs:simpleType>
                  <xs:restriction base="xs:unsignedShort">
                    <xs:minInclusive value="30"/>
                    <xs:maxInclusive value="32769"/>
                  </xs:restriction>
                </xs:simpleType>
                <xs:simpleType>
                  <xs:restriction base="xs:unsignedShort">
                    <xs:minInclusive value="32768"/>
                    <xs:maxInclusive value="65535"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:union>
            </xs:simpleType>
            <xs:anyAttribute namespace="##any"/>
          </xs:restriction>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>

    <xs:element name="OtherResourceType" nillable="true" type="cim:cimString"/>

    <xs:element name="ResourceSubType" nillable="true" type="cim:cimString"/>

    <xs:element name="PoolID" nillable="true" type="cim:cimString"/>

    <xs:element name="ConsumerVisibility" nillable="true">
      <xs:complexType>
        <xs:simpleContent>
```

CODE EXAMPLE A-7 The `cim-rasd.xsd` Schema *(Continued)*

```
        <xs:restriction base="xs:anyType">
          <xs:simpleType>
            <xs:union>
              <xs:simpleType>
                <xs:restriction base="xs:unsignedShort">
                  <xs:enumeration value="0"/>
                  <xs:enumeration value="2"/>
                  <xs:enumeration value="3"/>
                  <xs:enumeration value="4"/>
                </xs:restriction>
              </xs:simpleType>
              <xs:simpleType>
                <xs:restriction base="xs:unsignedShort">
                  <xs:minInclusive value="5"/>
                  <xs:maxInclusive value="32768"/>
                </xs:restriction>
              </xs:simpleType>
              <xs:simpleType>
                <xs:restriction base="xs:unsignedShort">
                  <xs:minInclusive value="32767"/>
                  <xs:maxInclusive value="65535"/
                </xs:restriction>
              </xs:simpleType>
            </xs:union>
          </xs:simpleType>
          <xs:anyAttribute namespace="##any"/>
        </xs:restriction>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>

  <xs:element name="HostResource" nillable="true" type="xs:anyType"/>
  <xs:element name="AllocationUnits" nillable="true" type="cim:cimString"/>
  <xs:element name="VirtualQuantity" nillable="true" type=
"cim:cimUnsignedLong"/>
  <xs:element name="Reservation" nillable="true" type="cim:cimUnsignedLong"/>
  <xs:element name="Limit" nillable="true" type="cim:cimUnsignedLong"/>
  <xs:element name="Weight" nillable="true" type="cim:cimUnsignedInt"/>
  <xs:element name="AutomaticAllocation" nillable="true" type=
"cim:cimBoolean"/>
  <xs:element name="AutomaticDeallocation" nillable="true" type=
"cim:cimBoolean"/>
  <xs:element name="Parent" nillable="true" type="cim:cimString"/>
  <xs:element name="Connection" nillable="true" type="cim:cimString"/>
  <xs:element name="Address" nillable="true" type="cim:cimString"/
  <xs:element name="MappingBehavior" nillable="true">
    <xs:complexType>
```

CODE EXAMPLE A-7 The `cim-rasd.xsd` Schema *(Continued)*

```
      <xs:simpleContent>
        <xs:restriction base="xs:anyType">
          <xs:simpleType>
            <xs:union>
              <xs:simpleType>
                <xs:restriction base="xs:unsignedShort">
                  <xs:enumeration value="0"/>
                  <xs:enumeration value="1"/>
                  <xs:enumeration value="2"/>
                  <xs:enumeration value="3"/>
                  <xs:enumeration value="4"/>
                </xs:restriction>
              </xs:simpleType>
              <xs:simpleType>
                <xs:restriction base="xs:unsignedShort">
                  <xs:minInclusive value="5"/>
                  <xs:maxInclusive value="32768"/>
                </xs:restriction>
              </xs:simpleType>
              <xs:simpleType>
                <xs:restriction base="xs:unsignedShort"
                  <xs:minInclusive value="32767"/>
                  <xs:maxInclusive value="65535"/>
                </xs:restriction>
              </xs:simpleType>
            </xs:union>
          </xs:simpleType>
          <xs:anyAttribute namespace="##any"/>
        </xs:restriction>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="AddressOnParent" nillable="true" type="cim:cimString"/>

  <xs:element name="BusNumber" nillable="true" type="cim:cimUnsignedShort"/>

  <xs:complexType name="CIM_ResourceAllocationSettingData_Type">
    <xs:sequence>
      <xs:element ref="class:Caption" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="class:Description" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="class:InstanceId" minOccurs="0"/>
      <xs:element ref="class:ResourceType" minOccurs="0"/>
      <xs:element ref="class:OtherResourceType" minOccurs="0"/>
      <xs:element ref="class:ResourceSubType" minOccurs="0"/>
      <xs:element ref="class:PoolID" minOccurs="0"/>
      <xs:element ref="class:ConsumerVisibility" minOccurs="0"/>
```

CODE EXAMPLE A-7 The `cim-rasd.xsd` Schema *(Continued)*

```
      <xs:element ref="class:HostResource" maxOccurs="unbounded" minOccurs=
"0"/>
      <xs:element ref="class:AllocationUnits" minOccurs="0"/>
      <xs:element ref="class:VirtualQuantity" minOccurs="0"/>
      <xs:element ref="class:Reservation" minOccurs="0"/>
      <xs:element ref="class:Limit" minOccurs="0"/>
      <xs:element ref="class:Weight" minOccurs="0"/>
      <xs:element ref="class:AutomaticAllocation" minOccurs="0"/>
      <xs:element ref="class:AutomaticDeallocation" minOccurs="0"/>
      <xs:element ref="class:Parent" minOccurs="0"/>
      <xs:element ref="class:Connection" maxOccurs="unbounded" minOccurs="0"/>
      <xs:element ref="class:Address" minOccurs="0"/>
      <xs:element ref="class:MappingBehavior" minOccurs="0"/>
      <xs:element ref="class:AddressOnParent" minOccurs="0"/>
      <xs:element ref="class:BusNumber" minOccurs="0"/>
      <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/
    </xs:sequence>
    <xs:anyAttribute namespace="##any"/>
  </xs:complexType>

  <xs:element name="CIM_ResourceAllocationSettingData"
type="class:CIM_ResourceAllocationSettingData_Type"/>
</xs:schema>
```

The cim-vssd.xsd Schema

CODE EXAMPLE A-8 The cim-vssd.xsd Schema

```
<?xml version='1.0' encoding='utf-8'?>
<xs:schema
  targetNamespace="/var/opt/SUNWldom/CIM_VirtualSystemSettingData"
  xmlns:class="/var/opt/SUNWldom/CIM_VirtualSystemSettingData"
  xmlns:cim="/var/opt/SUNWldom/common"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:import namespace="/var/opt/SUNWldom/common"
    schemaLocation="cim-common.xsd"/>

  <xs:element name="Caption" nillable="true" type="cim:cimString"/>

  <xs:element name="Description" nillable="true" type="cim:cimString"/>

  <xs:element name="InstanceId" nillable="true" type="cim:cimString"/>

  <xs:element name="VirtualSystemIdentifier" nillable="true" type=
"cim:cimString"/>

  <xs:element name="VirtualSystemType" nillable="true" type="cim:cimString"/>

  <xs:complexType name="CIM_VirtualSystemSettingData_Type">
    <xs:sequence>
      <xs:element ref="class:Caption" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="class:Description" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="class:InstanceId" minOccurs="0"/>
      <xs:element ref="class:VirtualSystemIdentifier" minOccurs="0"/>
      <xs:element ref="class:VirtualSystemType" minOccurs="0"/>
      <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any"/>
  </xs:complexType>

  <xs:element name="CIM_VirtualSystemSettingData"
type="class:CIM_VirtualSystemSettingData_Type"/>

</xs:schema>
```

The cim-common.xsd Schema

CODE EXAMPLE A-9 The cim-common.xsd Schema

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema
  targetNamespace="/var/opt/SUNWldom/common"
  xmlns:cim="/var/opt/SUNWldom/common"
  xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">

  <!-- The following are runtime attribute definitions -->
  <xs:attribute name="Key" type="xs:boolean"/>

  <xs:attribute name="Version" type="xs:string"/>

  <!-- The following section defines the extended WS-CIM datatypes -->
  <xs:complexType name="cimDateTime">
    <xs:choice>
      <xs:element name="CIM_DateTime" type="xs:string" nillable="true"/>
      <xs:element name="Interval" type="xs:duration"/>
      <xs:element name="Date" type="xs:date"/>
      <xs:element name="Time" type="xs:time"/>
      <xs:element name="Datetime" type="xs:dateTime"/>
    </xs:choice>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
  </xs:complexType>

  <xs:complexType name="cimUnsignedByte">
    <xs:simpleContent>
      <xs:extension base="xs:unsignedByte">
        <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="cimByte">
    <xs:simpleContent>
      <xs:extension base="xs:byte">
        <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="cimUnsignedShort">
    <xs:simpleContent>
      <xs:extension base="xs:unsignedShort">
```

CODE EXAMPLE A-9 The `cim-common.xsd` Schema *(Continued)*

```
        <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="cimShort">
    <xs:simpleContent>
      <xs:extension base="xs:short">
        <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="cimUnsignedInt">
    <xs:simpleContent>
      <xs:extension base="xs:unsignedInt">
        <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="cimInt">
    <xs:simpleContent>
      <xs:extension base="xs:int">
        <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="cimUnsignedLong">
    <xs:simpleContent>
      <xs:extension base="xs:unsignedLong">
        <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="cimLong">
    <xs:simpleContent>
      <xs:extension base="xs:long">
        <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="cimString">
    <xs:simpleContent>
```

CODE EXAMPLE A-9 The `cim-common.xsd` Schema *(Continued)*

```
      <xs:extension base="xs:string">
        <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="cimBoolean">
    <xs:simpleContent>
      <xs:extension base="xs:boolean">
        <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="cimFloat">
    <xs:simpleContent>
      <xs:extension base="xs:float">
        <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="cimDouble">
    <xs:simpleContent>
      <xs:extension base="xs:double">
        <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="cimChar16">
    <xs:simpleContent>
        <xs:restriction base="cim:cimString">
          <xs:maxLength value="1"/>
          <xs:anyAttribute namespace="##any" processContents="lax"/>
        </xs:restriction>
      </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="cimBase64Binary">
    <xs:simpleContent>
      <xs:extension base="xs:base64Binary">
        <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
```

CODE EXAMPLE A-9 The `cim-common.xsd` Schema *(Continued)*

```
  <xs:complexType name="cimHexBinary">
    <xs:simpleContent>
      <xs:extension base="xs:hexBinary">
        <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="cimReference">
    <xs:sequence>
      <xs:any namespace="##other" maxOccurs="unbounded"/>
    </xs:sequence>
    xs:anyAttribute namespace="##any" processContents="lax"/>
  </xs:complexType>

  <!-- The following datatypes are used exclusively to define metadata fragments
--
  <xs:attribute name="qualifier" type="xs:boolean"/>

  <xs:complexType name="qualifierString">
    <xs:simpleContent>
      <xs:extension base="cim:cimString">
        <xs:attribute ref="cim:qualifier" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="qualifierBoolean">
    <xs:simpleContent>
      <xs:extension base="cim:cimBoolean">
        <xs:attribute ref="cim:qualifier" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="qualifierUInt32">
    <xs:simpleContent>
      <xs:extension base="cim:cimUnsignedInt">
        <xs:attribute ref="cim:qualifier" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="qualifierSInt64">
    <xs:simpleContent>
      <xs:extension base="cim:cimLong">
        <xs:attribute ref="cim:qualifier" use="required"/>
```

CODE EXAMPLE A-9 The `cim-common.xsd` Schema *(Continued)*

```
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
<!--
  <xs:complexType name="qualifierSArray">
    <xs:complexContent>
      <xs:extension base="cim:qualifierString"/>
    </xs:complexContent>
  </xs:complexType>
-->
  <!-- The following element is to be used only for defining metadata -->
  <xs:element name=" DefaultValue" type="xs:anySimpleType"/>
</xs:schema>
```

The `GenericProperty` XML Schema

This schema is an extension to the Open Virtualization Format (OVF) schema.

CODE EXAMPLE A-10 The `GenericProperty` XML Schema

```
<?xml version='1.0' encoding='utf-8'?>
<xs:schema
  targetNamespace="/var/opt/SUNWldom/GenericProperty"
  xmlns:class="/var/opt/SUNWldom/GenericProperty"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:complexType name="GenericProperty_Type" type="xs:string">
  <xs:attribute name="key" type="xs:string" use="required"/>

  </xs:complexType>
  <xs:element name="GenericProperty" type="class:GenericProperty_Type"/>

</xs:schema>
```

Binding_Type XML Schema

This schema is an extension to the Open Virtualization Format (OVF) schema.

CODE EXAMPLE A-11 Binding_Type XML Schema

```
<?xml version='1.0' encoding='utf-8'?>
<xs:schema
  targetNamespace="/var/opt/SUNWldom/Binding"
  xmlns:class="/var/opt/SUNWldom/Binding"
xmlns:rasd="/var/opt/SUNWldom/CIM_ResourceAllocationSettingData"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:import namespace="/var/opt/SUNWldom/CIM_ResourceAllocationSettingData"
      schemaLocation="cim-rasd.xsd"/>

  <xs:complexType name="Binding_Type">
    <xs:sequence>
      <xs:element name="Item"
        type="rasd:CIM_ResourceAllocationSettingData_Type"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Glossary

This list defines terminology, abbreviations, and acronyms in the Logical Domains documentation.

A

ALOM CMT Advanced Lights Out Manager chip multithreading, which runs on the system controller and allows you to monitor and control your CMT server

API Application Programming Interface

`auditreduce`**(1M)** merge and select audit records from audit trail files

auditing Using the Solaris OS BSM to identify the source of security changes

authorization Setting up authorization using the Solaris OS RBAC

B

`bge` Broadcom Gigabit Ethernet driver on Broadcom BCM57xx devices

BSM Basic Security module

`bsmconv`**(1M)** enable the BSM

`bsmunconv`**(1M)** disable the BSM

C

CD compact disc

CLI command-line interface

compliance Determining if a system's configuration is in compliance with a predefined security profile

`config` Name of logical domain configuration saved on the system controller

CMT chip multithreading

constraints To the Logical Domains Manager, constraints are one or more resources you want to have assigned to a particular domain. You either receive all the resources you ask to be added to a domain or you get none of them, depending upon the available resources.

control domain Domain that creates and manages other logical domains and services

CPU central processing unit

CWQ Control Word Queue; cryptographic unit for Sun UltraSPARC T2-based platforms

D

DHCP Dynamic Host Configuration Protocol

DMA Direct Memory Access is the ability to directly transfer data between the memory and a device (for example, a network card) without involving the CPU.

DMP Dynamic Multipathing (Veritas)

DPS data plane software

DR dynamic reconfiguration

`drd`**(1M)** dynamic reconfiguration daemon for Logical Domains Manager (Solaris 10 OS)

DS Domain Services module (Solaris 10 OS)

DVD digital versatile disc

E

`e1000g` driver for Intel PRO/1000 Gigabit family of network interface controllers

EFI extensible firmware interface

ETM Encoding Table Management module (Solaris 10 OS)

F

FC_AL Fiber Channel Arbitrated Loop

FMA Fault Management Architecture

`fmd`**(1M)** fault manager daemon (Solaris 10 OS)

`format`**(1M)** disk partitioning and maintenance utility

`fmthard`**(1M)** populate label on hard disks

FTP File Transfer Protocol

G

Gb gigabit

guest domain Uses services from the I/O and service domains and is managed by the control domain.

GLDv3 Generic LAN Driver version 3.

H

hardening Modifying Solaris OS configuration to improve security

HDD hard disk drive

hypervisor Firmware layer interposed between the operating system and the hardware layer

I

I/O domain Domain that has direct ownership of and direct access to physical I/O devices and that shares those devices to other logical domains in the form of virtual devices

IB Infiniband

IDE Integrated Drive Electronics

IDR Interim Diagnostics Release

ILOM Integrated Lights Out Manager

`io` I/O devices, such as internal disks and PCIe controllers and their attached adapters and devices

`ioctl` input/output control call

IP Internet Protocol

IPMP Internet Protocol Network Multipathing

ISO International Organization for Standardization

K

`kaio` kernel asynchronous input/output

KB kilobyte

KU kernel update

L

LAN local-area network

LDAP Lightweight Directory Access Protocol

LDC logical domain channel

`ldm`**(1M)** Logical Domain Manager utility

`ldmd` Logical Domains Manager daemon

`lofi` loopback file

logical domain Discrete logical grouping with its own operating system, resources, and identity within a single computer system

Logical Domains (LDoms) Manager Provides a CLI to create and manage logical domains and allocate resources to domains

LUN logical unit number

M

MAC media access control address, which LDoms can automatically assign or you can assign manually

MAU Modular Arithmetic Unit; the cryptographic device for Sun UltraSPARC T1–based platforms

MB megabyte

MD machine description in the server database

`mem`, `memory` memory unit - default size in bytes, or specify gigabytes (`G`), kilobytes (`K`), or megabytes (`M`). Virtualized memory of the server that can be allocated to guest domains.

`metadb`**(1M)** create and delete replicas of the SVM metadevice state database

`metaset`**(1M)** configure disk sets

`mhd`**(7I)** multihost disk control operations

MIB Management Information Base

minimizing Installing the minimum number of core Solaris OS package necessary

MMF Multimode fiber

MMU memory management unit

`mpgroup` multipathing group name for virtual disk failover

`mtu` maximum transmission unit

N

NAT Network Address Translation

`ndpsldcc` Netra DPS Logical Domain Channel Client. *See also* `vdpcc`.

`ndpsldcs` Netra DPS Logical Domain Channel Service. *See also* `vdpcs`.

NFS network file system

NIS Network Information Services

NIU Network Interface Unit (Sun SPARC Enterprise T5120 and T5220 servers)

NTS network terminal server

NVRAM non-volatile random-access memory

`nxge` Driver for Sun x8 Express 1/10G Ethernet Adapter

O

OS operating system

OVF Open Virtualization Format

P

PA physical address

PCI peripheral component interconnect bus

PCIe PCI Express bus

PCI-X PCI Extended bus

`pcpu` physical CPU

`physio` physical input/output

PICL Platform Information and Control Library

`picld`**(1M)** PICL daemon

PM Power management of virtual CPUs

`praudit`**(1M)** print contents of an audit trail file

PRI priority

R

RA real address

RAID Redundant Array of Inexpensive Disks

RBAC Role-Based Access Control

RPC Remote Procedure Call

S

SASL Simple Authentication and Security Layer

SAX Simple API for XML parser, which traverses an XML document. The SAX parser is event-based and used mostly for streaming data.

SC system controller, also known as service processor

SCSI Small Computer System Interface

service domain Logical domain that provides devices, such as virtual switches, virtual console connectors, and virtual disk servers to other logical domains

SMA System Management Agent

SMF Service Management Facility

SNMP Simple Network Management Protocol

SP service processor, also known as system controller

SSH Secure Shell

`ssh`**(1)** Secure Shell command

`sshd`**(1M)** Secure Shell daemon

SunVTS Sun Validation Test Suite

`svcadm`**(1M)** manipulate service instances

SVM Solaris Volume Manager

T

TCP Transmission Control Protocol

TLS Transport Layer Security

U

UDP User Diagram Protocol

UFS UNIX File System

unicast Communication that takes place over a network between a single sender and a single receiver.

USB Universal Serial Bus

`uscsi`**(7D)** user SCSI command interface

UTP unshielded twisted pair

V

`var` variable

VBSC virtual blade system controller

`vcc`, `vconscon` virtual console concentrator service with a specific port range to assign to the guest domains

`vcons`, `vconsole` virtual console for accessing system level messages. A connection is achieved by connecting to `vconscon` service in the control domain at a specific port.

`vcpu` virtual central processing unit. Each of the cores of a server are represented as virtual CPUs. For example, an 8-core Sun Fire T2000 Server has 32 virtual CPUs that can be allocated between the logical domains.

`vdc` virtual disk client

`vdpcc` virtual data plane channel client in a Netra DPS environment

`vdpcs` virtual data plane channel service in a Netra DPS environment

`vdisk` virtual disks are generic block devices backed by different types of physical devices, volumes, or files.

`vds`, `vdiskserver` virtual disk server allows you to import virtual disks into a logical domain.

`vdsdev`, `vdiskserverdevice` virtual disk server device is exported by the virtual disk server. The device can be an entire disk, a slice on a disk, a file, or a disk volume.

VLAN virtual local area network

`vldc` virtual logical domain channel service

`vldcc` virtual logical domain channel client

`vnet` virtual network device implements a virtual Ethernet device and communicates with other `vnet` devices in the system using the virtual network switch (`vswitch`).

`vntsd`**(1M)** virtual network terminal server daemon for Logical Domains consoles (Solaris 10 OS)

`volfs`**(7FS)** Volume Management file system

`vsw`, `vswitch` virtual network switch that connects the virtual network devices to the external network and also switches packets between them.

VTOC volume table of contents

VxDMP Veritas Dynamic Multipathing

VxVM Veritas Volume Manager

W

WAN wide-area network

X

XFP eXtreme Fast Path

XML Extensible Markup Language

XMPP Extensible Messaging and Presence Protocol

Z

ZFS Zettabyte File System (Solaris 10 OS)

`zpool`**(1M)** ZFS storage pool

ZVOL ZFS Volume Emulation Driver