Oracle® VM Server for SPARC 3.6 Developer's Guide

Part No: E93620
August 2018

ORACLE

Oracle VM Server for SPARC 3.6 Developer's Guide

Part No: E93620

Référence: E93620

Accès aux services de support Oracle

Les clients Oracle qui ont souscrit un contrat de support ont accès au support électronique via My Oracle Support. Pour plus d'informations, visitez le site `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` ou le site `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` si vous êtes malentendant.

Contents

Using This Documentation

- **Overview** – Provides detailed information and procedures that describe the Oracle VM Server for SPARC templates, the XML interface, and the Logical Domains Manager discovery and Virtual Domain Information APIs.
- **Audience** – System administrators who manage virtualization on SPARC servers
- **Required knowledge** – System administrators on these servers must have a working knowledge of UNIX systems and the Oracle Solaris operating system (Oracle Solaris OS)

Product Documentation Library

Documentation and resources for this product and related products are available at http://www.oracle.com/technetwork/documentation/vm-sparc-194287.html.

Feedback

Provide feedback about this documentation at http://www.oracle.com/goto/docfeedback.

♦ ♦ ♦ CHAPTER 1

Developing Oracle VM Server for SPARC Templates, XML Files, and Programs

Developing Oracle VM Server for SPARC Templates, XML Files, and Programs

This guide provides information about the following Oracle VM Server for SPARC features:

- **Oracle VM Server for SPARC template utilities.** Use these utilities to create templates that you can use to deploy domain configurations. See Chapter 2, “Using Oracle VM Server for SPARC Templates”.
- **Oracle VM Server for SPARC XML interface.** Use this XML interface to describe the configuration of logical domains. See Chapter 3, “Using the XML Interface With the Logical Domains Manager”.
- **Logical Domains Manager discovery API.** Use this API to write programs that can discover which domains run the Logical Domains Manager. See Chapter 4, “Logical Domains Manager Discovery”.
- **Virtual Domain Information API.** Use this API to write programs that provide information about virtual domains. See Chapter 5, “Using the Virtual Domain Information Command and API”.

Note - The features that are described in this book can be used with all of the supported system software and hardware platforms that are listed in *Oracle VM Server for SPARC 3.6 Installation Guide*. However, some features are only available on a subset of the supported system software and hardware platforms. For information about these exceptions, see “What’s New in This Release” in *Oracle VM Server for SPARC 3.6 Release Notes* and What's New in Oracle VM Server for SPARC Software (`http://www.oracle.com/technetwork/server-storage/vm/documentation/sparc-whatsnew-330281.html`).

♦ ♦ ♦ CHAPTER 2

Using Oracle VM Server for SPARC Templates

This chapter covers the following topics:

About Oracle VM Server for SPARC Templates

The Oracle VM Server for SPARC Templates commands enable you to create, deploy, and configure Oracle VM Server for SPARC templates for SPARC systems. These commands are based on the OVF template specification that includes disk image files and an XML descriptor of properties that is contained in an archive (`.ova`).

You run these commands in the control domain of an Oracle VM Server for SPARC system. In addition to running the commands on the command line, you can run them in a fully automated fashion or from other programs as part of a larger work flow.

You can use these templates to deploy and configure only guest domains, I/O domains, and root domains. However, you cannot use templates to deploy and configure I/O domains or root domains that are also the `primary` domain or to deploy and configure service domains.

The Oracle VM Server for SPARC template commands are:

- `ovmtadm` – Enables you to create, configure, deploy, and remove a template by initiating the `ovmtcreate`, `ovmtconfig`, and `ovmtdeploy` commands. See the `ovmtadm`(8) man page.
- `ovmtconfig` – Performs configuration actions on a domain by transferring property name-value pairs to applications and processes in the target domain, such as Oracle Solaris OS configuration mechanisms and first-boot scripts. The property name-value pairs are set by using the `ovmtprop` command.

Also, you can use this command to back mount a domain's ZFS file systems so that the control domain runs commands directly on those file systems. This method includes the capability to copy files, install and upgrade Oracle Solaris OS packages, and perform configuration actions. See the `ovmtconfig`(8) man page.

- `ovmtcreate` – Creates a template from an existing Oracle VM Server for SPARC domain. See the `ovmtcreate`(8) man page.
- `ovmtdeploy` – Creates a domain from an Oracle VM Server for SPARC template. See the `ovmtdeploy`(8) man page.
- `ovmtlibrary` – Manages a database and file-system-based repository for Oracle VM Server for SPARC templates by organizing files, and storing, retrieving, and editing information in the database. See the `ovmtlibrary`(8) man page.
- `ovmtprop` – Enables you to view and set Oracle Solaris OS properties in the domain that is deployed from a template. The property is specified as a name-value pair. This command is called from other scripts and programs to perform configuration actions and actions that might be property-driven. See the `ovmtprop`(8) man page.

For information about getting started with the Oracle VM Server for SPARC templates, see Getting Started with OVM Templates for SPARC - Part 3: Using Templates on SuperCluster (`https://blogs.oracle.com/cmt/getting-started-with-ovm-templates-for-sparc-part-3%3a-using-templates-on-supercluster`)

Installing the Oracle VM Server for SPARC Template Utilities

To install the Oracle VM Server for SPARC template utilities software package on the control domain, you must first download and install the Oracle VM Server for SPARC 3.6 software on the control domain. See Chapter 2, “Installing the Software” in *Oracle VM Server for SPARC 3.6 Installation Guide*.

Then, install the Oracle VM Server for SPARC template utilities software package, `ovmtutils`.

```
# pkg install -v pkg:/system/ldoms/ovmtutils
```

Oracle VM Server for SPARC Template Lifecycle

This section describes each stage of the template creation process, the actions that are taken, and how to use the Oracle VM Server for SPARC template utilities to assist in the process:

Note - The creation and development of templates with applications and first-boot scripts is an iterative process. Take care to synchronize all aspects of the configuration by using a source code management system to manage the scripts and properties.

The following describes the stages of the template-creation process, the actions that are taken, and how to use the Oracle VM Server for SPARC template utilities to assist in the process:

1. **Authoring a template.** While pre-built, generic templates are available, you can create a custom template from an existing domain. This domain must have all of the operating system components, application software, and other utilities that you want fully installed.

   Typically, the environment is configured as fully as possible, with only a small number of actions required to finalize the environment. Any domain settings such as memory, virtual CPU, virtual networking, and disks should reflect the deployment that you want.

   At this stage, you create one or more “first-boot” scripts. Include these scripts in the environment that performs the final configuration based on the properties that you supply. Be sure to record and describe these properties in a README file for each template.

   Note - If any first-boot scripts access domain variables, ensure that the `ovmtprop` utility is installed in the guest domain.

2. **Creating a template.** Before you create a template, ensure that the source domain environment is not configured so that it can be configured later by prescribed actions that are often part of the first-boot scripts.

   For example, perform the following steps:

   - Remove any application-specific configurations to be re-created later.
   - Use default values for configuration files.
   - Export any zpools other than the root file system so that they can be recognized by new domains.
   - Revert the operating system to an unconfigured state so that it is ready to accept configuration properties on its first boot after deployment from a template. Run the following command to remove any site-level customizations, unconfigure, and halt the operating system:

     ```
     # sysconfig unconfigure -g system -s --include-site-profile --destructive
     ```

   After you take these steps, run the `ovmtcreate` command to create a template from the domain.

Note - If you run this command and no properties are supplied at deployment time, the system runs the Oracle Solaris Interactive Installer on the next boot.

---

3. **Specifying the name of the template.** Use a consistent convention to identify the template, such as the following format:

   technology`.`*OS*`.`*application*`.`*architecture*`.`*build*`.ova`

   For example, the following template name is for a domain that runs build 2 of the Oracle Solaris 11.2 OS on a SPARC platform and that runs version 12.1.2 of the WebLogic server: `OVM_S11.2_WLS12.1.2_SPARC_B2.ova`

4. **Distributing the template.** The template is a single file with a `.ova` extension. The file contains the compressed disk images and the metadata that are required for deployment. The template also contains a manifest file of payload file checksums, which might be combined with an overall archive checksum to validate that the content has not been changed since distribution.

   You can distribute the template by using web-based services or maintain a central repository rather than duplicating templates.

5. **Deploying the template.** Because the template captures only those aspects of a system that are seen by the source domain, you must understand which services must be present to support the deployment of the template.

   The required services include the following:

   - One or more virtual switches to appropriate interfaces to which virtual networks from the template might be attached
   - Virtual disk services
   - Console services
   - Sufficient virtual CPU and memory to accommodate the template requirements

   While the `ovmtdeploy` utility can override many of these settings, the minimum values that are supplied with a template represent the baseline requirements.

   You can use the `ovmtdeploy` utility to automatically extract, decompress, and copy the virtual disks to deployment directories, and to build the various virtual devices that the template describes.

   At this point, you could start the domain but you might need to perform some manual configuration steps by using the domain console before the domain is fully functional.

6. **Automatically configuring the domain.** The configuration of a domain that is created by a template consists of several types of actions. For example, you might specify property name-value pairs to provide first-boot scripts with the information to configure. You might

also back-mount virtual disks to the control domain to perform actions on the domain file systems such as copying configuration files.

The `ovmtconfig` utility automates these domain-configuration activities and enables you to specify the actions to take and the properties to use to configure a domain by specifying one or more command scripts and property files.

To configure the Oracle Solaris OS, the `ovmtconfig` utility back-mounts the domain's root file system and creates an `sc_profile.xml` file from the supplied configuration scripts and properties. This profile enables the Oracle Solaris OS to configure itself on first boot.

7. **First configuration.** Following the successful configuration of the Oracle Solaris OS and first boot, you must configure any installed applications. During the configuration phase, the `ovmtconfig` utility passes configuration information to the deployed domain by using one of the following methods:

   - **Domain variables –** In addition to a local properties file, you can set domain variables by running the `ovmtconfig` utility in the control domain that can then be used by the `ovmtprop` utility in the guest domain. This method enables first-boot scripts to access the properties directly and provides configuration information directly to the guest domain after the configuration completes.

     For example, you might automate a change of a configuration aspect that does not have network access by using a supervisor script that runs `ovmtprop` in the guest and by running `ovmtconfig -v` from the control domain.

   - **Direct action –** The `ovmtconfig` utility back-mounts the guest domain file systems to the control domain and takes direct action on the files and file systems. Actions might include the creation of configuration files or the copying of system binaries. These actions are described in the scripts that you supply to the `ovmtconfig` utility. You can find scripts that perform command actions in the `/opt/ovmtutils/share/scripts` directory.

     Note - These actions do not typically include first-boot processes that are designed to run in the guest domain because such actions might affect the control domain.

     Use the `ovmtconfig -c` command to specify the commands to run.

Caution - Do not use unencrypted properties to pass sensitive information to the domain such as passwords. Properties other than those that are used to configure the Oracle Solaris OS are passed to the domain as `ldm` variables in clear text. These property values are visible to a user on the control domain who is authorized to execute `ldm` commands and to a user who is logged in to the deployed domain.

At this point, the domain should be fully configured and operational.

Oracle VM Server for SPARC Template Features

Creating SPARC OpenStack Images From a Source Domain

The `ovmtcreate` command has a new `-m` option that enables you to select either the `openstack` disk image format or the default `ovf` template format.

Use the `ovmtcreate -m openstack` command to create a single, uncompressed SPARC OpenStack-compatible disk image directly from the first virtual disk in a source domain. Note that this command does not create a complete template, which would include additional payload items such as additional disk images, an OVF metadata file and a manifest file. Also, this command does not encapsulate these components in an `.ova` tar file. Other metadata options are ignored, such as those that provide a description, specify boilerplate files, or specify minor and major versions.

The `ovmtcreate -m ovf` command creates a complete OVF template, which is the same as running the `ovmtcreate` command without using the `-m ovf` option.

Expanding Underlying Disk Devices

The `ovmtdeploy` command now expands the underlying disk devices to device extents during template deployment. This expansion operation occurs by default and supports only disk devices and not disk image files. You can use the `ovmtdeploy -x` command to disable the expansion operation at runtime.

Previously, the size of the resulting disk was determined by the size of the original source domain encapsulated in the template. So, deploying a template that contains a 20-Gbyte system disk to a disk device that has 600 Gbytes results in a disk formatted to a 20-Gbyte size. Now that the underlying disk device can be expanded, this same template deployment results in a disk formatted to its full 600-Gbyte size.

While the underlying disk device has been expanded, the guest domain OS might require that additional actions are performed to recognize and grow to the larger space. To perform these actions for the Oracle Solaris OS, run the `/opt/ovmtutils/share/scripts/`

`ovmt_s11_expand_disk.sh` script immediately following the deployment operation. See the `ovmtconfig`(8) man page.

Oracle VM Server for SPARC Template Examples

This section shows examples of the following Oracle VM Server for SPARC template tasks:

EXAMPLE 1 Creating an Oracle VM Server for SPARC Template

The following `ovmtcreate` command creates a template based on the `ldg1` domain called `ovmtcreate_example`. Note that the resulting template name has the `.ova` suffix.

```
# ovmtcreate -d ldg1 -o ovmtcreate_example
...

STAGE 1 - EXAMINING SYSTEM AND ENVIRONMENT
------------------------------------------
Performing platform & prerequisite checks
Checking user permissions
Checking for required packages
Checking for required services
Checking directory permissions

STAGE 2 - ANALYZING DOMAIN
--------------------------
Retrieving and processing attributes
Checking domain state
Getting domain resource settings
Discovering network topology
Discovering disk topology

STAGE 3 - ARCHIVE CREATION
--------------------------
Checking destination and current directory capacity
Compressing disk image
Creating XML configuration
Calculating manifest checksums
```

```
Creating archive file
Checking archive

PROCESS COMPLETED
-----------------
Started: Tue Jun 12 15:29:14 PDT 2018
Completed: Tue Jun 12 15:41:25 PDT 2018
Elapsed time: 0:12:11
```

EXAMPLE 2 Configuring Oracle VM Server for SPARC Template Properties

You can use the `ovmtconfig` and `ovmtprop` utilities to specify Oracle VM Server for SPARC template property values and Oracle Solaris OS property values, respectively.

- The following `ovmtconfig` command performs configuration operations directly on the `ldg1` domain's file system.

  The `-c` option specifies the `/opt/ovmtutils/share/scripts/ovmt_s11_scprofile.sh` script to set property values. The `-p` option specifies particular values for the `com.oracle.solaris.network.ipaddr` and `com.oracle.solaris.system.computer-name` properties.

  ```
  # ovmtconfig -v -d ldg1 -f -s \
  -c /opt/ovmtutils/share/scripts/ovmt_s11_scprofile.sh \
  -p com.oracle.solaris.network.ipaddr.0=10.153.118.211,\
  com.oracle.solaris.system.computer-name=system1
  ...

  STAGE 1/7 - EXAMINING SYSTEM AND ENVIRONMENT
  --------------------------------
  Checking operating system
  Checking platform type
  Checking user permissions
  Checking packages
  Checking host domain name
  Checking host domain type
  Checking services

  STAGE 2/7 - PROCESSING COMMAND LINE OPTIONS
  --------------------------------
  Parsing individual properties
  Creating consolidated properties list

  STAGE 3/7 - ANALYZING TARGET DOMAIN
  -----------------------
  Stopping domain ldg1
  ```

```
Analyzing domain disk topology for domain ldg1
Discovering 1 volumes for vDisks
Examining 1 backend devices
unbinding domain ldg1
Creating 1 virtual disks for back mount
Created virtual disk 0

STAGE 4/7 - PERFORMING BACKMOUNT
--------------------
Finding Solaris device for vdisks
Importing zpools for 1 Solaris devices
Detected conflicting zpool name, attempting rename
Getting boot file system for properties in 1 zpool
Setting properties in 1 zpools
Mounting ZFS file systems
Mounting ZFS found in zpool rpool_1

STAGE 5/7 - PERFORMING ACTIONS ON TARGET DOMAIN
-----------------------------------

STAGE 6/7 - UNMOUNTING AND RESTORING DOMAIN STATE
-------------------------------------
Rolling back commands DEBUG [20150819-07:02:42]: Rolling back 8 /usr/sbin/zfs unmount
 -f rpool_1/ROOT/solaris/var
completed

STAGE 7/7 - SETTING TARGET DOMAIN ENVIRONMENT
--------------------------------
Checking 2 properties to set as domain variables
Process completed
```

- The following `ovmtprop` command specifies Oracle Solaris OS property values.

  ```
  primary# ovmtprop set-prop com.oracle.solaris.system.computer-name=test ldg1
  ```

  Use the `ldm list -l` command to verify that the value for the `com.oracle.solaris.system.computer-name` property is `test`.

  ```
  primary# ldm list -l ldg1
  NAME             STATE      FLAGS   CONS    VCPU  MEMORY   UTIL  NORM  UPTIME
  ldg1             active     -n----  5000    8     8G       0.0%  0.0%  23h 23m
  ..
  VARIABLES
      auto-boot?=true
      boot-file=-k
  ```

```
        pm_boot_policy=disabled=0;ttfc=2000;ttmr=0;

    VMAPI TO GUEST
        com.oracle.solaris.fmri.count=0
        com.oracle.solaris.system.computer-name=test
```

EXAMPLE 3 Deploying Oracle VM Server for SPARC Templates

The following `ovmtdeploy` command creates a domain called `ldg1` by using the `ovmtcreate_example.ova` Oracle VM Server for SPARC template in the `/export/ovmtdeploy` directory.

```
# ovmtdeploy -d ldg1 -o /export/ovmtdeploy ovmtcreate_example.ova
...

STAGE 1 - EXAMINING SYSTEM AND ENVIRONMENT
------------------------------------------
Checking user privilege
Performing platform & prerequisite checks
Checking for required services
Named resourced available

STAGE 2 - ANALYZING ARCHIVE & RESOURCE REQUIREMENTS
---------------------------------------------------
Checking .ova format and contents
Validating archive configuration
Checking sufficient resources present
WARNING: Virtual switch primary-vsw0 already exists

STAGE 3 - EXTRACTING ARCHIVE
----------------------------
Extracting archive
Validating checksums
Decompressing disk image(s)

STAGE 4 - DEPLOYING DOMAIN
--------------------------
Creating domain and adding resources
Validating deployment
Domain created:
```

The `ldm list` output shows that you have created a new domain called `ldg1`.

```
# ldm list
NAME    STATE  FLAGS  CONS VCPU MEMORY UTIL NORM UPTIME
primary active -n-cv- UART 8    40G    1.4% 1.1% 6d 2h 18m
ldg1    active -n---- 5000 8    8G     41%  38%  28s
```

EXAMPLE 4 Managing the Oracle VM Server for SPARC Template Library

The `ovmtlibrary` command manages a database and file-system-based repository for Oracle VM Server for SPARC templates by organizing files and by storing, retrieving, and editing information in the database.

- The following command creates a template library in `export/user1/ovmtlibrary_example`:

```
# ovmtlibrary -c init -l /export/user1/ovmtlibrary_example
...

Init complete
```

- The following command stores the `sol-11_2-ovm-2-sparc.ova` template in the `export/user1/ovmtlibrary_example` library:

```
# ovmtlibrary -c store -d "ovmtlibrary example" -o http://system1.example.com/s11.2/
templates/sol-11_2-ovm-2-sparc.ova -l /export/user1/ovmtlibrary_example
...

Templates present in path "/export/user1/ovmtlibrary_example"

event id is 2

********************************************************************************
converted 'http://system1.example.com/s11.2/templates/sol-11_2-ovm-2-sparc.ova' (646)
 ->
'http://system1.example.com/s11.2/templates/sol-11_2-ovm-2-sparc.ova' (UTF-8)
--2015-08-18 16:37:17-- http://system1.example.com/s11.2/templates/sol-11_2-ovm-2-
sparc.ova
Resolving system1.example.com (system1.example.com)... 10.134.127.18
Connecting to system1.example.com (system1.example.com)|10.134.127.18|:80...
 connected.
HTTP request sent, awaiting response... 200 OK
Length: 1018341888 (971M) [text/plain]
Saving to: '/export/user1/ovmtlibrary_example/repository/templates/1/1/sol-11_2-
ovm-2-sparc.ova'

/export/user1/ovmtlibrary_example/repo 100%
[================================================================================>]
 971.17M 6.05MB/s in 5m 37s
2015-08-18 16:42:55 (2.88 MB/s) - '/export/user1/ovmtlibrary_example/repository/
templates/1/1/sol-11_2-ovm-2-sparc.ova' saved
[1018341888/1018341888]
```

```
********************************************************************************
Download complete
Extracting the ova file...
Extract complete
Decompress file System.img.gz
Store complete
```

- The following command lists the contents of the `export/user1/ovmtlibrary_example` library:

```
# ovmtlibrary -c list -l /export/user1/ovmtlibrary_example
...

Templates present in path "/export/user1/ovmtlibrary_example"

ID Name                         Version Description         Date
--------------------------------------------------------------------------------
1  sol-11_2-ovm-2-sparc         1       ovmtlibrary example 2015-08-18
```

- The following command shows a detailed listing of the `export/user1/ovmtlibrary_example` library:

```
# ovmtlibrary -c list -i 1 -o -l /export/user1/ovmtlibrary_example
...

Templates present in path "/export/user1/ovmtlibrary_example"

ID Name                           Type Path                       Size(bytes)
--------------------------------------------------------------------------------
1  sol-11_2-ovm-2-sparc.ova       ova /export/user1/ovmtlibrary_example/repository/
templates/1/1/sol-11_2-ovm-2-sparc.ova 1018341888
2  sol-11_2-ovm-sparc.ovf         ovf /export/user1/ovmtlibrary_example/repository/
templates/1/1/sol-11_2-ovm-sparc.ovf 3532
3  sol-11_2-ovm-sparc.mf          mf /export/user1/ovmtlibrary_example/repository/
templates/1/1/sol-11_2-ovm-sparc.mf 137
4  System.img                     img /export/user1/ovmtlibrary_example/repository/
templates/1/1/System.img 21474836480
```

- The following command deletes template ID 1 from the `export/user1/ovmtlibrary_example` library:

```
# ovmtlibrary -c delete -i 1 -l /export/user1/ovmtlibrary_example
```

♦ ♦ ♦ CHAPTER 3

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 Oracle VM Server for SPARC software. These basic topics are covered:

XML Transport

External programs can 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. The XML interface supports only version 1.2 of the Transport Layer Security (TLS) protocol.

To disable a remote connection, set the `ldmd/xmpp_enabled` SMF property to `false` and restart the Logical Domains Manager.

```
# svccfg -s ldom/ldmd setprop ldmd/xmpp_enabled=false
# svcadm refresh ldmd
# svcadm restart ldmd
```

Note - Disabling the XMPP server also prevents domain migration and the dynamic reconfiguration of memory.

XMPP Server

The Logical Domains Manager implements an XMPP server which can communicate with numerous available XMPP client applications and libraries. The Logical Domains Manager uses the following security mechanisms:

- Transport Layer Security 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 Logical Domains 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, Oracle VM Server for SPARC-defined XML messages are sent next. The two general types of XML messages are:

- Request and response messages, which use the `<LDM_interface>` tag. This type of XML message is used for communicating commands and getting results back from the Logical Domains Manager, analogous to executing commands using the command-line interface (CLI). This tag is also used for event registration and unregistration.
- Event messages, which use the `<LDM_event>` tag. This type of XML message is used to asynchronously report events posted by the Logical Domains Manager.

Request and Response Messages

The XML interface into Oracle VM Server for SPARC has two different formats:

- A format for sending commands into the Logical Domains Manager
- A format for Logical Domains 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 they are separated in this discussion for a better understanding of the differences between them.

Request Messages

An incoming XML request to the Logical Domains 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. The following example shows the structure of a basic XML command.

EXAMPLE 5 Format of a Single Command Operating on a Single Object

```
<LDM_interface version="1.7">
  <cmd>
    <action>Place command here</action>
    <options>Place options for certain commands here</options>
    <arguments>Place arguments for certain commands here</arguments>
    <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>

        </Content>
      </Envelope>
    </data>

  </cmd>
```

```
</LDM_interface>
```

<LDM_interface> Tag

All commands sent to the Logical Domains Manager must start with the <LDM_interface> tag. Any document sent into the Logical Domains Manager must have only one <LDM_interface> tag contained within it. The <LDM_interface> tag must include a version attribute as shown in Example 5, “Format of a Single Command Operating on a Single Object,” on page 25.

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 <cmd> tag can also have an <options> tag, which is used for options and flags that are associated with some commands. The following commands use options:

- The `ldm add-spconfig` command can use the `-r` *autosave-name* option.
- The `ldm add-vdsdev` command can use the `-f` option.
- The `ldm bind-domain` command can use the `-f` option.
- The `ldm cancel-operation` command can use the `migration` or `reconf` option.
- The `ldm list-bindings` command can use the `-e` option.
- The `ldm list-dependencies` command can use the `-r` option.
- The `ldm list-devices` command can use the `-B` option.
- The `ldm list-domain` command can use the `-e` option.
- The `ldm list-io` command can use the `-l` option.
- The `ldm list-netdev` command can use the `-b` option and the `-l` option.
- The `ldm list-rsrc-group` command can use the `-a` option.
- The `ldm list-spconfig` command can use the `-r` [*autosave-name*] option.
- The `ldm remove-domain` command can use the `-a` option.
- The `ldm remove-spconfig` command can use the `-r` option.
- The `ldm stop-domain` command can use the following tags to set the command arguments:
  - <force> represents the `-f` option.

- <halt> represents the -h option.
- <message> represents the -m option.
- <quick> represents the -q option.
- <reboot> represents the -r option.
- <timeout> represents the -t option.

  Note that the tags must not have any content value. However, the -t and -m options must have a non-null value, for example, <timeout>10</timeout> or <message>Shutting down now</message>.

The following XML example fragment shows how to use the <options> tag to pass the -l option to the ldm list-io command to generate a long listing:

```
<cmd>
    <action>list-io</action>
    <options>-l</options>
    <data version="3.0">
    </data>
</cmd>
```

The following XML example fragment shows how to use the <arguments> tag to pass a reboot request with a reboot message to the ldm stop-domain command:

```
<action>stop-domain</action>
<arguments>
    <reboot/>
    <message>my reboot message</message>
</arguments>
```

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 Oracle VM Server for SPARC) and <Content> and <Section> sections.

For Oracle VM Server for SPARC, 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 need to identify only 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 need to provide a <Section> section describing the resources needed for the command, placed outside 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.

Two additional OVF types enable the use of the OVF specification's schema to properly define all types of objects:

- <gprop:GenericProperty> tag
- <Binding> tag

The <gprop:GenericProperty> tag handles 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 ldm list-bindings command output to define resources that are bound to other resources.

Response Messages

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. The following example shows the structure of a response to a basic XML request.

EXAMPLE 6 Format of a Response to a Single Command Operating on a Single Object

```
<LDM_interface version="1.7">
  <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>
      </Content>
    </Envelope>
    <response>
      <status>success or failure</status>
      <resp_msg>Reason for failure</resp_msg>
    </response>
  </data>

  <response>
    <status>success or failure</status>
    <resp_msg>Reason for failure</resp_msg>
  </response>
</cmd>

<response>
  <status>success or failure</status>
  <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 whether 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 that the command ran against caused a failure.

Object Response

Finally, each <data> section in a <cmd> section also has a <response> section. This section shows whether 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 27. 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

Event Messages

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 32 for complete details.

Registration and Unregistration

Use an <LDM_interface> message to register for events. See "<LDM_interface> Tag" on page 26. The <action> tag details the type of event for which to register or unregister and the <data> section is left empty.

EXAMPLE 7 Example Event Registration Request Message

```
<LDM_interface version="1.3">
```

```
  <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.

EXAMPLE 8 Example Event Registration Response Message

```
<LDM_interface version="1.3">
  <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.

<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.

EXAMPLE 9 Example <LDM_event> Notification

```
<LDM_event version='1.1'>
  <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

You can subscribe to the following event types:

- Domain events
- Hardware events
- Progress events
- Resource events

All the events correspond to `ldm` subcommands.

Domain Events

Domain events describe which actions can be performed directly to a domain. The following domain events can be specified in the `<action>` tag in the `<LDM_event>` message:

- `add-domain`
- `bind-domain`
- `domain-reset`
- `domain-state-change`
- `migrate-domain`
- `panic-domain`
- `remove-domain`
- `start-domain`
- `stop-domain`

- `unbind-domain`

These events always contain *only* a `<Content>` tag in the OVF `<data>` section that describes the domain to which the event happened. To register for the domain events, send an `<LDM_interface>` message with the `<action>` tag set to `reg-domain-events`. To unregister for these events, send an `<LDM_interface>` message with the `<action>` tag set to `unreg-domain-events`.

EXAMPLE 10 `domain-state-change` Event With `transition` Soft State

```
<LDM_event version="1.1">
  <cmd>
    <action>domain-state-change</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_info</rasd:OtherResourceType>
              <gprop:GenericProperty
                key="domain-soft-state">transition</gprop:GenericProperty>
            </Item>
          </Section>
        </Content>
      </Envelope>
    </data>
  </cmd>
</LDM_event>
```

EXAMPLE 11 `domain-state-change` Event With `normal` Soft State

```
<LDM_event version="1.1">
  <cmd>
    <action>domain-state-change</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_info</rasd:OtherResourceType>
              <gprop:GenericProperty
                key="domain-soft-state">normal</gprop:GenericProperty>
            </Item>
          </Section>
```

```
        </Content>
      </Envelope>
    </data>
  </cmd>
</LDM_event>
```

Hardware Events

Hardware events pertain to changing the physical system hardware. In the case of Oracle VM Server for SPARC software, the only hardware changes are those to the service processor (SP) when you add, remove, or set 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`. To unregister for these events, send an `<LDM_interface>` message with the `<action>` tag set to `unreg-hardware-events`.

Progress Events

Progress events are issued for long-running commands, such as a domain migration. These events report the amount of progress that has been made during the life of the command. At this time, only the `migration-process` event is reported.

Progress events always contain only a `<Section>` tag in the OVF `<data>` section that describes the SP configuration affected by the event. To register for these events, send an `<LDM_interface>` message with the `<action>` tag set to `reg-hardware-events`. To unregister for these events, send an `<LDM_interface>` message with the `<action>` tag set to `unreg-hardware-events`.

The `<data>` section of a progress event consists of a `<content>` section that describes the affected domain. This `<content>` section uses an `ldom_info` `<Section>` tag to update progress. The following generic properties are shown in the `ldom_info` section:

- `--progress` – Percentage of the progress made by the command

- `--status` – Command status, which can be one of `ongoing`, `failed`, or `done`
- `--source` – Machine that is reporting the progress

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 providing a service name in the OVF `<data>` section.

The following events can be specified in the `<action>` tag in the `<LDM_event>` message:

- `add-vdiskserverdevice`
- `remove-vdiskserverdevice`
- `set-vdiskserverdevice`
- `remove-vdiskserver`
- `set-vconscon`
- `remove-vconscon`
- `set-vswitch`
- `remove-vswitch`

The following resource events always contain *only* the `<Content>` tag in the OVF `<data>` section that describes the domain to which the event happened:

- `add-vcpu`
- `add-memory`
- `add-io`
- `add-variable`
- `add-vconscon`
- `add-vdisk`
- `add-vdiskserver`
- `add-vnet`
- `add-vsan`
- `add-vswitch`
- `remove-io`
- `remove-memory`
- `remove-variable`
- `remove-vcpu`

- `remove-vdisk`
- `remove-vnet`
- `set-memory`
- `set-variable`
- `set-vconsole`
- `set-vcpu`
- `set-vdisk`
- `set-vnet`

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`.

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 `ldm` command-line interface. For details about `ldm` subcommands, see the `ldm`(8) 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:

- `add-domain`
- `add-io`
- `add-memory`
- `add-spconfig`

- add-variable
- add-vconscon
- add-vcpu
- add-vdisk
- add-vdiskserver
- add-vdiskserverdevice
- add-vhba
- add-vnet
- add-vsan
- add-vsan-dev
- add-vswitch
- bind-domain
- cancel-operation
- list-bindings
- list-constraints
- list-dependencies
- list-devices
- list-domain
- list-hba
- list-io
- list-netdev
- list-netstat
- list-rsrc-group
- list-services
- list-spconfig
- list-variable
- list-vmapi
- list-vsan
- migrate-domain
- reg-all-events
- reg-domain-events
- reg-hardware-events
- reg-resource-events
- remove-domain

- remove-io
- remove-memory
- remove-reconf
- remove-spconfig
- remove-variable
- remove-vconscon
- remove-vcpu
- remove-vdisk
- remove-vdiskserver
- remove-vdiskserverdevice
- remove-vhba
- remove-vmapi
- remove-vnet
- remove-vsan
- remove-vsan-dev
- remove-vswitch
- rescan-vhba
- set-domain
- set-memory
- set-spconfig
- set-variable
- set-vconscon
- set-vconsole
- set-vcpu
- set-vhba
- set-vmapi
- set-vnet
- set-vsan
- set-vswitch
- start-domain
- stop-domain
- unbind-domain
- unreg-all-events
- unreg-domain-events
- unreg-hardware-events

- `unreg-resource-events`

Logical Domains Manager Resources and Properties

This section provides examples of 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 except domain migration output. See "Domain Migration" on page 66. Each resource is defined in a `<Section>` OVF section and is specified by a `<rasd:OtherResourceType>` tag.

Domain Information (`ldom_info`) Resource

The following example shows the optional properties of the `ldom_info` resource:

EXAMPLE 12 Example `ldom_info` XML Output

The following example shows values specified for several `ldom_info` properties such as `uuid`, `hostid`, and `Address`.

```
<Envelope>
  <References/>
  <Content xsi:type="ovf:VirtualSystem_Type" id="primary">
    <Section xsi:type="ovf:ResourceAllocationSection_type">
      <Item>
        <rasd:OtherResourceType>ldom_info</rasd:OtherResourceType>
        <uuid>c2c3d93b-a3f9-60f6-a45e-f35d55c05fb6</uuid>
        <rasd:Address>00:03:ba:d8:ba:f6</rasd:Address>
        <gprop:GenericProperty key="hostid">83d8baf6</gprop:GenericProperty>
        <gprop:GenericProperty key="master">plum</gprop:GenericProperty>
        <gprop:GenericProperty key="failure-policy">reset</gprop:GenericProperty>
        <gprop:GenericProperty key="extended-mapin-space">on</gprop:GenericProperty>
        <gprop:GenericProperty key="progress">45%</gprop:GenericProperty>
        <gprop:GenericProperty key="status">ongoing</gprop:GenericProperty>
        <gprop:GenericProperty key="source">system1</gprop:GenericProperty>
        <gprop:GenericProperty key="rc-add-policy"></gprop:GenericProperty>
        <gprop:GenericProperty key="perf-counters">global</gprop:GenericProperty>
      </Item>
    </Section>
  </Content>
```

```
</Envelope>
```

The `ldom_info` resource is always contained within a `<Content>` section. The following properties within the `ldom_info` resource are optional properties:

- `<uuid>` tag, which specifies the UUID of the domain.
- `<rasd:Address>` tag, which specifies the MAC address to be assigned to a domain.
- `<gprop:GenericProperty key="extended-mapin-space">` tag, which specifies whether extended mapin space is enabled (`on`) or disabled (`off`) for the domain. The default value is `off`.
- `<gprop:GenericProperty key="failure-policy">` tag, which specifies how slave domains should behave should the master domain fail. The default value is `ignore`. Following are the valid property values:
  - `ignore` ignores failures of the master domain (slave domains are unaffected).
  - `panic` panics any slave domains when the master domain fails.
  - `reset` resets any slave domains when the master domain fails.
  - `stop` stops any slave domains when the master domain fails.
- `<gprop:GenericProperty key="hostid">` tag, which specifies the host ID to be assigned to the domain.
- `<gprop:GenericProperty key="master">` tag, which specifies up to four comma-separated master domain names.
- `<gprop:GenericProperty key="progress">` tag, which specifies the percentage of progress made by the command.
- `<gprop:GenericProperty key="source">` tag, which specifies the machine reporting on the progress of the command.
- `<gprop:GenericProperty key="status">` tag, which specifies the status of the command (`done`, `failed`, or `ongoing`).
- `<gprop:GenericProperty key="rc-add-policy">` tag, which specifies whether to enable or disable the direct I/O and SR-IOV I/O virtualization operations on any root complex that might be added to the specified domain. Valid values are `iov` and no value (`rc-add-policy=`).
- `<gprop:GenericProperty key="perf-counters">` tag, which specifies the performance register sets to access (`global`, `htstrand`, `strand`).

  If the platform does not have the performance access capability, the `perf-counters` property value is ignored.
- `<gprop:GenericProperty key="boot-policy">` tag, which specifies the verified boot policy.
- `<gprop:GenericProperty key="shutdown-group">` tag, which specifies the shutdown group number for a domain.

CPU (cpu) Resource

Note that the allocation units property, <rasd:AllocationUnits>, for the cpu resource always specifies the number of virtual CPUs and not the number of cores.

A cpu resource is always contained within a <Content> section.

EXAMPLE 13 cpu XML Section Output from the ldm list-bindings Command

The following example shows the XML output for the <cpu> section by using the ldm list-bindings command.

```
<?xml version="1.0"?>
<LDM_interface
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ovf="./schemas/envelope"
xmlns:rasd="./schemas/CIM_ResourceAllocationSettingData"
xmlns:vssd="./schemas/CIM_VirtualSystemSettingData"
xmlns:gprop="./schemas/GenericProperty"
xmlns:bind="./schemas/Binding"
version="1.3"
xsi:noNamespaceSchemaLocation="./schemas/combined-v3.xsd">
  <cmd>
    <action>list-bindings</action>
    <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>
              <uuid>1e04cdbd-472a-e8b9-ba4c-d3eee86e7725</uuid>
              <rasd:Address>00:21:28:f5:11:6a</rasd:Address>
              <gprop:GenericProperty key="hostid">0x8486632a</gprop:GenericProperty>
              <failure-policy>fff</failure-policy>
              <wcore>0</wcore>
              <extended-mapin-space>0</extended-mapin-space>
              <cpu-arch>native</cpu-arch>
              <rc-add-policy/>
              <gprop:GenericProperty key="state">active</gprop:GenericProperty>
            </Item>
          </Section>
          <Section xsi:type="ovf:VirtualHardwareSection_Type">
            <Item>
              <rasd:OtherResourceType>cpu</rasd:OtherResourceType>
```

```
              <rasd:AllocationUnits>8</rasd:AllocationUnits>
              <bind:Binding>
                <Item>
                  <rasd:OtherResourceType>cpu</rasd:OtherResourceType>
                  <gprop:GenericProperty key="vid">0</gprop:GenericProperty>
                  <gprop:GenericProperty key="pid">0</gprop:GenericProperty>
                  <gprop:GenericProperty key="cid">0</gprop:GenericProperty>
                  <gprop:GenericProperty key="strand_percent">100</gprop:
GenericProperty>
                  <gprop:GenericProperty key="util_percent">1.1%</gprop:GenericProperty>
                  <gprop:GenericProperty key="normalized_utilization">0.1%</gprop:
GenericProperty>
                </Item>
          </Section>
        </Content>
      </Envelope>
    </data>
  </cmd>
</LDM_interface>
```

EXAMPLE 14 `cpu` XML Section Output from the `ldm list-domain` Command

The following example shows the XML output for the `<cpu>` section by using the `ldm list-domain` command.

```
<?xml version="1.0"?>
<LDM_interface
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ovf="./schemas/envelope"
xmlns:rasd="./schemas/CIM_ResourceAllocationSettingData"
xmlns:vssd="./schemas/CIM_VirtualSystemSettingData"
xmlns:gprop="./schemas/GenericProperty"
xmlns:bind="./schemas/Binding"
version="1.3"
xsi:noNamespaceSchemaLocation="./schemas/combined-v3.xsd">
  <cmd>
    <action>list-domain</action>
    <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>
              <gprop:GenericProperty key="state">active</gprop:GenericProperty>
              <gprop:GenericProperty key="flags">-n-cv-</gprop:GenericProperty>
              <gprop:GenericProperty key="utilization">0.7%</gprop:GenericProperty>
```

```
              <gprop:GenericProperty key="uptime">3h</gprop:GenericProperty>
              <gprop:GenericProperty key="normalized_utilization">0.1%</gprop:
GenericProperty>
            </Item>
          </Section>
        </Content>
      </Envelope>
    </data>
  </cmd>
</LDM_interface>
```

Memory (`memory`) Resource

A memory resource is always contained within a `<Content>` section. The only property is the `<rasd:AllocationUnits>` tag, which signifies the amount of memory.

EXAMPLE 15 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>
```

Virtual SAN (`vsan`) Resource

A virtual SAN (`vsan`) resource can be in a `<Content>` section. It must use `<gprop:GenericProperty>` tags with the following properties:

- `service_name` – Specifies the name of the virtual SAN
- `vsan_mask` – Specifies whether the `mask` property value is `on` or `off`
- `vol_name` – Specify zero or more instances of this property based on the value of the `mask` property:

- `mask=on`**.** Specify zero or more entries. Each entry must specify the worldwide number (WWN) of the device.
- `mask=off`**.** Specify `*` to indicate that all devices are included.

EXAMPLE 16 Creating a Virtual SCSI Host Bus Adapter

The following XML input represents the `ldm add-vhba id=14 timeout=27 my-vhba my-vsan primary` command, which creates the `my-vhba` virtual SCSI HBA on the `primary` domain. The `my-vhba` virtual SCSI HBA connects to the `my-vsan` virtual SAN. The `my-vhba` is created with an ID of `14` and a timeout of 27 seconds.

```
<LDM_interface ...>
<cmd>
  <action>add-vhba</action>
  <data version="3.0">
    <Envelope>
      <Content ovf:id="primary" xsi:type="ovf:VirtualSystem_Type">
        <Section type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>vhba</rasd:OtherResourceType>
            <gprop:GenericProperty key="vhba_name">my-vhba</gprop:GenericProperty>
            <gprop:GenericProperty key="service_name">my-vsan</gprop:GenericProperty>
            <gprop:GenericProperty key="timeout">27</gprop:GenericProperty>
            <gprop:GenericProperty key="id">14</gprop:GenericProperty>
          </Item>
        </Section>
      </Content>
    </Envelope>
  </data>
</cmd>
</LDM_interface>
```

EXAMPLE 17 Modifying the Behavior of a Virtual SCSI Host Bus Adapter

The following XML input represents the `ldm set-vhba timeout=20 my-vhba primary` command, which sets the `timeout` property value to `20` seconds for the `my-vhba` virtual SCSI HBA on the `primary` domain.

```
<LDM_interface ...>
<cmd>
  <action>set-vhba</action>
  <data version="3.0">
    <Envelope>
      <Content ovf:id="primary" xsi:type="ovf:VirtualSystem_Type">
        <Section type="ovf:VirtualHardwareSection_Type">
```

```
          <Item>
            <rasd:OtherResourceType>vhba</rasd:OtherResourceType>
            <gprop:GenericProperty key="vhba_name">my-vhba</gprop:GenericProperty>
            <gprop:GenericProperty key="timeout">20</gprop:GenericProperty>
          </Item>
        </Section>
      </Content>
    </Envelope>
  </data>
</cmd>
</LDM_interface>
```

EXAMPLE 18 Removing a Virtual SCSI Host Bus Adapter

The following XML input represents the `ldm remove-vhba my-vhba primary` command, which removes the `my-vhba` virtual SCSI HBA from the `primary` domain.

```
<LDM_interface ...>
<cmd>
  <action>remove-vhba</action>
  <data version="3.0">
    <Envelope>
      <Content ovf:id="primary" xsi:type="ovf:VirtualSystem_Type">
        <Section type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>vhba</rasd:OtherResourceType>
            <gprop:GenericProperty key="vhba_name">my-vhba</gprop:GenericProperty>
          </Item>
        </Section>
      </Content>
    </Envelope>
  </data>
</cmd>
</LDM_interface>
```

EXAMPLE 19 Creating a Virtual Storage Area Network

The following XML input represents the `ldm add-vsan /pci@340/pci@1/pci@0/pci@6/SUNW,emlxs@0/fp@0,0 my-vsan primary` command, which creates the `my-vsan` virtual SAN on the `primary` domain. The `my-vsan` virtual SAN is associated with the `/pci@340/pci@1/pci@0/pci@6/SUNW,emlxs@0/fp@0,0` SCSI HBA initiator port.

```
<LDM_interface ...>
<cmd>
  <action>add-vsan</action>
```

```
  <data version="3.0">
    <Envelope>
      <References/>
      <Content xsi:type="ovf:VirtualSystem_Type" ovf:id="primary">
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>vsan</rasd:OtherResourceType>
            <gprop:GenericProperty key="vsan_iport">
            /pci@340/pci@1/pci@0/pci@6/SUNW,emlxs@0/fp@0,0
            </gprop:GenericProperty>
            <gprop:GenericProperty key="service_name">my-vsan</gprop:GenericProperty>
          </Item>
        </Section>
      </Content>
    </Envelope>
  </data>
</cmd>
</LDM_interface>
```

EXAMPLE 20 Creating a Virtual Storage Area Network With `mask=on`

The following XML input represents the `ldm add-vsan /pci@340/pci@1/pci@0/pci@6/SUNW,emlxs@0/fp@0,0 mask=on my-vsan primary` command, which creates the `my-vsan` virtual SAN on the `primary` domain. The `my-vsan` virtual SAN is associated with the `/pci@340/pci@1/pci@0/pci@6/SUNW,emlxs@0/fp@0,0` SCSI HBA initiator port and sets `mask=on`.

```
<LDM_interface ...>
<cmd>
  <action>add-vsan</action>
  <data version="3.0">
    <Envelope>
      <References/>
      <Content xsi:type="ovf:VirtualSystem_Type" ovf:id="primary">
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>vsan</rasd:OtherResourceType>
            <gprop:GenericProperty key="vsan_iport">
            /pci@340/pci@1/pci@0/pci@6/SUNW,emlxs@0/fp@0,0</gprop:GenericProperty>
            <gprop:GenericProperty key="service_name">my-vsan</gprop:GenericProperty>
            <gprop:GenericProperty key="vsan_mask">on</gprop:GenericProperty>
          </Item>
        </Section>
      </Content>
    </Envelope>
  </data>
</cmd>
```

```
</LDM_interface>
```

EXAMPLE 21 Modifying a Virtual Storage Area Network to Set `mask=on`

The following XML input represents the `ldm set-vsan mask=on my-vsan` command, which sets the `mask` property value to `on` on the `my-vsan` virtual SAN.

```
<LDM_interface ...>
<cmd>
  <action>set-vsan</action>
  <data version="3.0">
    <Envelope>
      <References/>
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>vsan</rasd:OtherResourceType>
            <gprop:GenericProperty key="service_name">my-vsan</gprop:GenericProperty>
            <gprop:GenericProperty key="vsan_mask">on</gprop:GenericProperty>
          </Item>
        </Section>
    </Envelope>
  </data>
</cmd>
</LDM_interface>
```

EXAMPLE 22 Removing a Virtual Storage Area Network

The following XML input represents the `ldm remove-vsan my-vsan primary` command, which removes the `my-vsan` virtual SAN.

```
<LDM_interface ...>
<cmd>
  <action>remove-vsan</action>
    <data version="3.0">
      <Envelope>
        <References/>
          <Section xsi:type="ovf:VirtualHardwareSection_Type">
            <Item>
              <rasd:OtherResourceType>vsan</rasd:OtherResourceType>
              <gprop:GenericProperty key="service_name">my-vsan</gprop:GenericProperty>
            </Item>
          </Section>
        </Envelope>
     </data>
  </cmd>
</LDM_interface>
```

EXAMPLE 23 Listing a Virtual Storage Area Network Information

The following XML input represents the `ldm list-vsan my-vsan` command, which shows information about the `my-vsan` virtual SAN.

```
<LDM_interface ...>
<cmd>
  <action>list-vsan</action>
    <data version="3.0">
      <Envelope>
        <References/>
          <Section xsi:type="ovf:ResourceAllocationSection_Type">
            <Item>
              <rasd:OtherResourceType>vsan</rasd:OtherResourceType>
              <gprop:GenericProperty key="service_name">my-vsan</gprop:GenericProperty>
            </Item>
          </Section>
      </Envelope>
    </data>
</cmd>
</LDM_interface>
```

The following example XML output shows that the `mask` property is set to `on` and that the `naa.1234` and `naa.5678` physical devices are assigned to the `my-vsan` virtual SAN:

```
<?xml version="1.0"?>
<LDM_interface ...>
<cmd>
  <action>list-vsan</action>
  <data version="3.0">
    <Envelope>
      <References/>
      <Content xsi:type="ovf:VirtualSystem_Type" ovf:id="primary">
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>vsan</rasd:OtherResourceType>
            <gprop:GenericProperty key="service_name">my-vsan</gprop:GenericProperty>
            <gprop:GenericProperty key="vsan_mask">on</gprop:GenericProperty>
            <gprop:GenericProperty key="vol_name">naa.1234</gprop:GenericProperty>
            <gprop:GenericProperty key="vol_name">naa.5678</gprop:GenericProperty>
          </Item>
        </Section>
      </Content>
    </Envelope>
  <response>
    <status>success</status>
  </response>
</data>
```

```
  <response>
    <status>success</status>
  </response>
</cmd>
<response>
  <status>success</status>
</response>
</LDM_interface>
```

EXAMPLE 24 Viewing the Physical SCSI Host Bus Adapter Initiator Ports in All Domains in the System

The following XML input represents the `ldm list-hba` command, which shows the physical SCSI HBA initiator ports on all domains.

```
<LDM_interface ...>
<cmd>
  <action>list-hba</action>
  <data version="3.0">
  </data>
</cmd>
</LDM_interface>
```

This example XML output shows information about the physical SCSI HBA initiator ports in all domains on the system. In this example, it shows information about the ten SCSI HBA initiator ports in the `primary` domain and the three virtual SAN initiator ports in the `ldg0_mask_on` domain.

```
<?xml version="1.0"?>
<LDM_interface ...>
<cmd>
  <action>list-hba</action>
  <data version="3.0">
    <Envelope>
      <References/>
      <Content xsi:type="ovf:VirtualSystem_Type" ovf:id="primary">
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>hba</rasd:OtherResourceType>
            <gprop:GenericProperty key="vsan_iport">
            /pci@6c0/pci@1/pci@0/pci@c/pci@0/pci@c/scsi@0/iport@1
            </gprop:GenericProperty>
            <gprop:GenericProperty key="hba_nlun">1</gprop:GenericProperty>
          </Item>
        </Section>
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
          <Item>
```

```
      <rasd:OtherResourceType>hba</rasd:OtherResourceType>
      <gprop:GenericProperty key="vsan_iport">
      /pci@6c0/pci@1/pci@0/pci@c/pci@0/pci@c/scsi@0/iport@2
      </gprop:GenericProperty>
      <gprop:GenericProperty key="hba_nlun">1</gprop:GenericProperty>
    </Item>
  </Section>
  <Section xsi:type="ovf:VirtualHardwareSection_Type">
    <Item>
      <rasd:OtherResourceType>hba</rasd:OtherResourceType>
      <gprop:GenericProperty key="vsan_iport">
      /pci@6c0/pci@1/pci@0/pci@c/pci@0/pci@c/scsi@0/iport@4
      </gprop:GenericProperty>
      <gprop:GenericProperty key="hba_nlun">1</gprop:GenericProperty>
    </Item>
  </Section>
  <Section xsi:type="ovf:VirtualHardwareSection_Type">
    <Item>
      <rasd:OtherResourceType>hba</rasd:OtherResourceType>
      <gprop:GenericProperty key="vsan_iport">
      /pci@680/pci@1/pci@0/pci@4/SUNW,emlxs@0,1/fp@0,0
      </gprop:GenericProperty>
      <gprop:GenericProperty key="hba_nlun">0</gprop:GenericProperty>
    </Item>
  </Section>
  <Section xsi:type="ovf:VirtualHardwareSection_Type">
    <Item>
      <rasd:OtherResourceType>hba</rasd:OtherResourceType>
      <gprop:GenericProperty key="vsan_iport">
      /pci@680/pci@1/pci@0/pci@4/SUNW,emlxs@0/fp@0,0
      </gprop:GenericProperty>
      <gprop:GenericProperty key="hba_nlun">52</gprop:GenericProperty>
    </Item>
  </Section>
  <Section xsi:type="ovf:VirtualHardwareSection_Type">
    <Item>
      <rasd:OtherResourceType>hba</rasd:OtherResourceType>
      <gprop:GenericProperty key="vsan_iport">
      /pci@340/pci@1/pci@0/pci@6/SUNW,emlxs@0,1/fp@0,0
      </gprop:GenericProperty>
      <gprop:GenericProperty key="hba_nlun">0</gprop:GenericProperty>
    </Item>
  </Section>
  <Section xsi:type="ovf:VirtualHardwareSection_Type">
    <Item>
      <rasd:OtherResourceType>hba</rasd:OtherResourceType>
      <gprop:GenericProperty key="vsan_iport">
      /pci@340/pci@1/pci@0/pci@6/SUNW,emlxs@0/fp@0,0
```

```
            </gprop:GenericProperty>
            <gprop:GenericProperty key="hba_nlun">52</gprop:GenericProperty>
          </Item>
        </Section>
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>hba</rasd:OtherResourceType>
            <gprop:GenericProperty key="vsan_iport">
            /pci@300/pci@1/pci@0/pci@4/pci@0/pci@c/scsi@0/iport@2
            </gprop:GenericProperty>
            <gprop:GenericProperty key="hba_nlun">1</gprop:GenericProperty>
          </Item>
        </Section>
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>hba</rasd:OtherResourceType>
            <gprop:GenericProperty key="vsan_iport">
            /pci@300/pci@1/pci@0/pci@4/pci@0/pci@c/scsi@0/iport@4
            </gprop:GenericProperty>
            <gprop:GenericProperty key="hba_nlun">1</gprop:GenericProperty>
          </Item>
        </Section>
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>hba</rasd:OtherResourceType>
            <gprop:GenericProperty key="vsan_iport">
            /pci@300/pci@1/pci@0/pci@4/pci@0/pci@c/scsi@0/iport@1
            </gprop:GenericProperty>
            <gprop:GenericProperty key="hba_nlun">1</gprop:GenericProperty>
          </Item>
        </Section>
      </Content>
    </Envelope>
    <response>
      <status>success</status>
    </response>
</data>
<data version="3.0">
    <Envelope>
      <References/>
      <Content xsi:type="ovf:VirtualSystem_Type" ovf:id="ldg0_mask_on">
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>hba</rasd:OtherResourceType>
            <gprop:GenericProperty key="vsan_iport">
            /virtual-devices@100/channel-devices@200/scsi@0/iport@0
            </gprop:GenericProperty>
            <gprop:GenericProperty key="hba_nlun">1</gprop:GenericProperty>
```

```
          </Item>
        </Section>
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>hba</rasd:OtherResourceType>
            <gprop:GenericProperty key="vsan_iport">
            /virtual-devices@100/channel-devices@200/scsi@1/iport@0
            </gprop:GenericProperty>
            <gprop:GenericProperty key="hba_nlun">52</gprop:GenericProperty>
          </Item>
        </Section>
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>hba</rasd:OtherResourceType>
            <gprop:GenericProperty key="vsan_iport">/iscsi</gprop:GenericProperty>
            <gprop:GenericProperty key="hba_nlun">0</gprop:GenericProperty>
          </Item>
        </Section>
      </Content>
    </Envelope>
    <response>
      <status>success</status>
    </response>
  </data>
  <response>
    <status>success</status>
  </response>
</cmd>
<response>
  <status>success</status>
</response>
</LDM_interface>
```

EXAMPLE 25 Adding a Physical Device to a Virtual Storage Area Network

The following XML input represents the `ldm add-vsan-dev my-vsan naa.1234` command, which adds the `naa.1234` volume to the `my-vsan` virtual SAN.

```
<LDM_interface ...>
<cmd>
  <action>add-vsan-dev</action>
  <data version="3.0">
    <Envelope>
      <References/>
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>vsan_dev</rasd:OtherResourceType>
```

```
            <gprop:GenericProperty key="service_name">my-vsan</gprop:GenericProperty>
            <gprop:GenericProperty key="vol_name">naa.1234</gprop:GenericProperty>
          </Item>
        </Section>
    </Envelope>
  </data>
</cmd>
</LDM_interface>
```

EXAMPLE 26 Removing a Physical Device From a Virtual Storage Area Network

The following XML input represents the `ldm remove-vsan-dev my-vsan naa.1234` command, which removes the `naa.1234` device from the `my-vsan` virtual SAN.

```
<LDM_interface ...>
<cmd>
  <action>remove-vsan-dev</action>
  <data version="3.0">
    <Envelope>
      <References/>
      <Section xsi:type="ovf:VirtualHardwareSection_Type">
        <Item>
          <rasd:OtherResourceType>vsan_dev</rasd:OtherResourceType>
          <gprop:GenericProperty key="service_name">my-vsan</gprop:GenericProperty>
          <gprop:GenericProperty key="vol_name">naa.1234</gprop:GenericProperty>
        </Item>
      </Section>
    </Envelope>
  </data>
</cmd>
</LDM_interface>
```

Virtual Disk Server (`vds`) Resource

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`, which contains the name of the `vds` resource being described.

EXAMPLE 27 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>
```

Virtual Disk Server Volume (`vds_volume`) Resource

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

EXAMPLE 28 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>
```

Disk (`disk`) Resource

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 `timeout` property, which is the 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`. The timeout ensures that a connection to any `vds` is established within the specified amount of time.

EXAMPLE 29 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>
```

Virtual Switch (`vsw`) Resource

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 a `<gprop:GenericProperty>` tag with the `service_name` key, which is the name to be assigned to the virtual switch.

Optionally, the vsw resource can also have the following properties:

- <rasd:Address> – Assigns a MAC address to the virtual switch
- default-vlan-id – Specifies the default virtual local area network (VLAN) to which a virtual network device or virtual switch needs to be a member, in tagged mode. The first VLAN ID (*vid1*) is reserved for the default-vlan-id.
- dev_path – Path of the network device to be associated with this virtual switch
- id – Specifies the ID of a new virtual switch device. By default, ID values are generated automatically, so set this property if you need to match an existing device name in the OS.
- inter_vnet_link – Specifies whether to assign LDC channels for inter-vnet communication. Valid values are on, off, and auto. The default value is auto.
- linkprop – Specifies that the virtual device gets physical link state updates (the default value of phys-state). When the value is blank, the virtual device does not get physical link state updates.
- mode – sc for Oracle Solaris Cluster heartbeat support.
- 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.
- mtu – Specifies the maximum transmission unit (MTU) of a virtual switch, virtual network devices that are bound to the virtual switch, or both. Valid values are in the range of 1500-16000. The ldm command issues an error if an invalid value is specified.
- 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.
- vsw-relay-mode – Specifies how to exchange the network traffic between domains.

EXAMPLE 30 Example vsw XML

```
<Envelope>
  <References/>
  <Content xsi:type="ovf:VirtualSystem_Type" id="ldg2">
    <Section xsi:type="ovf:VirtualHardwareSection_Type">
      <Item>
        <rasd:OtherResourceType>vsw</rasd:OtherResourceType>
        <rasd:Address>00:14:4f:fb:ec:00</rasd:Address>
        <gprop:GenericProperty key="service_name">test-vsw1</gprop:GenericProperty>
        <gprop:GenericProperty key="inter_vnet_link">auto</gprop:GenericProperty>
        <gprop:GenericProperty key="default-vlan-id">1</gprop:GenericProperty>
        <gprop:GenericProperty key="pvid">1</gprop:GenericProperty>
        <gprop:GenericProperty key="mtu">1500</gprop:GenericProperty>
        <gprop:GenericProperty key="dev_path">switch@0</gprop:GenericProperty>
        <gprop:GenericProperty key="id">0</gprop:GenericProperty>
      </Item>
    </Section>
```

```
  </Content>
</Envelope>
```

Network (`network`) Resource

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 network
- `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.
- `linkprop` – Specifies that the virtual device gets physical link state updates (the default value of `phys-state`). When the value is blank, the virtual device does not get physical link state updates.
- `custom` – Specifies whether to enable or disable custom settings for the maximum number of VLANs and MAC addresses that can be assigned to a virtual network device from a trusted host. The default value is `disable`.
- `custom/max-mac-addrs` – Specifies the maximum number of MAC addresses that can be assigned to a virtual network device from a trusted host. The default value is 4096.
- `custom/max-vlans` – Specifies the maximum number of VLANs that can be assigned to a virtual network device from a trusted host. The default value is 4096.
- `alt-mac-addrs` – Specifies a comma-separated list of alternate MAC addresses.
- `auto-alt-mac-addrs` – Specifies the number of automatic alternate MAC addresses.
- `pvlan` – Specifies a private VLAN.
- `mtu` – Specifies the maximum transmission unit.
- `maxbw` – Specifies the maximum bandwidth limit for the specified port.
- `id` – Specifies the ID of the network device.
- `allowed-dhcp-cids` – Specifies a comma-separated list of MAC addresses or host names.
- `allowed-ips` – Specifies a comma-separated list of IP addresses.

- `cos` – Specifies the class of service priority that is associated with outbound packets on the link.
- `priority` – Specifies the relative priority of the link. The priority is used to schedule packet processing within the system.
- `protection` – Specifies the types of protection in the form of a bit-wise OR of the protection types.

EXAMPLE 31 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>
        <rasd:Address>auto-allocated</rasd:Address>
        <gprop:GenericProperty key="linkprop">phys-state</gprop:GenericProperty>
        <gprop:GenericProperty key="custom">enable</gprop:GenericProperty>
        <gprop:GenericProperty key="custom/max-mac-addrs">4096</gprop:GenericProperty>
        <gprop:GenericProperty key="custom/max-vlans">12</gprop:GenericProperty>
        <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>
```

Virtual Console Concentrator (`vcc`) Resource

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`

EXAMPLE 32 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>
```

Variable (`var`) Resource

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

EXAMPLE 33 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>
```

Physical I/O Device (`physio_device`) Resource

A `physio_device` resource is always contained within a `<Content>` section. This resource can be modified by using the `add-io`, `set-io`, `remove-io`, `create-vf`, `destroy-vf`, and `set-domain` subcommands.

EXAMPLE 34 Example `physio_device` XML

The following examples show how to perform actions on virtual functions, physical functions, and root complexes.

- The following XML example fragment shows how to use the `ldm add-io` command to add the `/SYS/MB/NET0/IOVNET.PF0.VF0` virtual function to the `ldg1` domain.

```
<LDM_interface version="1.3">
  <cmd>
    <action>add-io</action>
    <data version="3.0">
      <Envelope>
        <References/>
        <Content xsi:type="ovf:VirtualSystem_Type" ovf:id="ldg1">
          <Section xsi:type="ovf:VirtualHardwareSection_Type">
            <Item>
              <rasd:OtherResourceType>physio_device</rasd:OtherResourceType>
              <gprop:GenericProperty key="name">
              /SYS/MB/NET0/IOVNET.PF0.VF0</gprop:GenericProperty>
            </Item>
          </Section>
        </Content>
      </Envelope>
    </data>
  </cmd>
</LDM_interface>
```

- The following XML example fragment shows how to use the `ldm set-io` command to set the `iov_bus_enable_iov` property value to `on` for the `pci_1` root complex.

```
<LDM_interface version="1.3">
  <cmd>
    <action>set-io</action>
    <data version="3.0">
      <Envelope>
        <References/>
          <Section xsi:type="ovf:VirtualHardwareSection_Type">
            <Item>
              <rasd:OtherResourceType>physio_device</rasd:OtherResourceType>
              <gprop:GenericProperty key="name">pci_1</gprop:GenericProperty>
              <gprop:GenericProperty key="iov_bus_enable_iov">
              on</gprop:GenericProperty>
            </Item>
        </Section>
```

```
      </Envelope>
    </data>
  </cmd>
</LDM_interface>
```

- The following XML example fragment shows how to use the `ldm set-io` command to set the `unicast-slots` property value to `6` for the `/SYS/MB/NET0/IOVNET.PF1` physical function.

```
<LDM_interface version="1.3">
  <cmd>
    <action>set-io</action>
    <data version="3.0">
      <Envelope>
        <References/>
          <Section xsi:type="ovf:VirtualHardwareSection_Type">
            <Item>
              <rasd:OtherResourceType>physio_device</rasd:OtherResourceType>
              <gprop:GenericProperty key="name">
              /SYS/MB/NET0/IOVNET.PF1</gprop:GenericProperty>
              <gprop:GenericProperty key="unicast-slots">6</gprop:GenericProperty>
            </Item>
          </Section>
      </Envelope>
    </data>
  </cmd>
</LDM_interface>
```

- The following XML example fragment shows how to use the `ldm create-vf` command to create the `/SYS/MB/NET0/IOVNET.PF1.VF0` virtual function with the following property values.
  - `unicast-slots=6`
  - `pvid=3`
  - `mtu=1600`

```
<LDM_interface version="1.3">
  <cmd>
    <action>create-vf</action>
    <data version="3.0">
      <Envelope>
        <References/>
          <Section xsi:type="ovf:VirtualHardwareSection_Type">
            <Item>
              <rasd:OtherResourceType>vf_device</rasd:OtherResourceType>
```

```
              <gprop:GenericProperty key="iov_pf_name">
              /SYS/MB/NET0/IOVNET.PF1</gprop:GenericProperty>
              <gprop:GenericProperty key="unicast-slots">6</gprop:GenericProperty>
              <gprop:GenericProperty key="pvid">3</gprop:GenericProperty>
              <gprop:GenericProperty key="mtu">1600</gprop:GenericProperty>
            </Item>
          </Section>
      </Envelope>
    </data>
  </cmd>
</LDM_interface>
```

- The following XML example fragment shows how to use the `ldm create-vf` command to create the number of virtual functions specified by the `iov_pf_repeat_count_str` value (`3`) with the `/SYS/MB/NET0/IOVNET.PF1` physical function. You cannot specify other property values when you create multiple virtual functions with the `iov_pf_repeat_count_str` property.

```
<LDM_interface version="1.3">
  <cmd>
    <action>create-vf</action>
    <data version="3.0">
      <Envelope>
        <References/>
          <Section xsi:type="ovf:VirtualHardwareSection_Type">
            <Item>
              <rasd:OtherResourceType>vf_device</rasd:OtherResourceType>
              <gprop:GenericProperty key="iov_pf_name">
              /SYS/MB/NET0/IOVNET.PF1</gprop:GenericProperty>
              <gprop:GenericProperty key="iov_pf_repeat_count_str">
              3</gprop:GenericProperty>
            </Item>
          </Section>
      </Envelope>
    </data>
  </cmd>
</LDM_interface>
```

- The following XML example fragment shows how to use the `ldm set-io` command to add a user-assigned name to an existing virtual function.

  The `iov_vf_name` XML property refers to the name of an existing virtual function. The `iov_vf_tag` property refers to the user-assigned name.

Note - The `iov_vf_tag` property is used only when you assign a new name, whereas the `iov_vf_name` property is used always to refer to an existing virtual function, as in an `ldm destroy-vf` command.

```
<LDM_interface xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:ovf=".
/schemas/envelope" xmlns:rasd="./schemas/CIM_ResourceAllocationSettingData" xmlns:
vssd="./schemas/CIM_VirtualSystemSettingData" xmlns:gprop="./schemas/GenericProperty"
 xmlns:bind="./schemas/Binding" version="1.6" xsi:noNamespaceSchemaLocation="./
schemas/combined-v3.xsd">
  <cmd>
    <action>set-io</action>
    <data version="3.0">
      <Envelope>
        <References/>
          <Section xsi:type="ovf:VirtualHardwareSection_Type">
            <Item>
              <rasd:OtherResourceType>physio_device</rasd:OtherResourceType>
              <gprop:GenericProperty
                key="name">my_new_vf</gprop:GenericProperty>
              <gprop:GenericProperty
                key="iov_vf_tag">/SYS/MB/NET2/IOVNET.PF1.VF0</gprop:GenericProperty>
            </Item>
          </Section>
      </Envelope>
    </data>
  </cmd>
</LDM_interface>
```

SP Configuration (`spconfig`) Resource

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.
- `spconfig_aux_status` – The auxiliary status of a particular SP configuration. If the running configuration is a degraded configuration, this property is set to `degraded`.

EXAMPLE 35 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>
          key="spconfig_aux_status">degraded</gprop:GenericProperty>
      </Item>
    </Section>
</Envelope>
```

DRM Policy Configuration (`policy`) Resource

A DRM policy (`policy`) resource appears in an `<Envelope>` section and can have `<gprop:GenericProperty>` tags with the following keys:

- `policy_name` – Name of the DRM policy
- `policy_enable` – Specifies whether the DRM policy is enabled or disabled
- `policy_priority` – Priority of the DRM policy
- `policy_vcpu_min` – Minimum number of virtual CPU resources for a domain
- `policy_vcpu_max` – Maximum number of virtual CPU resources for a domain
- `policy_util_lower` – Lower utilization level at which policy analysis is triggered
- `policy_util_upper` – Upper utilization level at which policy analysis is triggered
- `policy_tod_begin` – Effective start time of the DRM policy
- `policy_tod_end` – Effective stop time of the DRM policy
- `policy_sample_rate` – The sample rate, which is the cycle time in seconds
- `policy_elastic_margin` – Amount of buffer between the upper and lower CPU utilization bounds
- `policy_attack` – Maximum amount of a resource to be added during any one resource control cycle
- `policy_decay` – Maximum amount of a resource to be removed during any one resource control cycle

EXAMPLE 36 Example `policy` XML

```
<Envelope>
```

```
   <Section xsi:type="ovf:VirtualHardwareSection_Type">
     <Item>
       <rasd:OtherResourceType>policy</rasd:OtherResourceType>
       <gprop:GenericProperty key="policy_name">test-policy</gprop:GenericProperty>
       <gprop:GenericProperty key="policy_enable">on</gprop:GenericProperty>
       <gprop:GenericProperty key="policy_priority">1</gprop:GenericProperty>
       <gprop:GenericProperty key="policy_vcpu_min">12</gprop:GenericProperty>
       <gprop:GenericProperty key="policy_vcpu_max">13</gprop:GenericProperty>
       <gprop:GenericProperty key="policy_util_lower">8</gprop:GenericProperty>
       <gprop:GenericProperty key="policy_util_upper">9</gprop:GenericProperty>
       <gprop:GenericProperty key="policy_tod_begin">07:08:09</gprop:GenericProperty>
       <gprop:GenericProperty key="policy_tod_end">09:08:07</gprop:GenericProperty>
       <gprop:GenericProperty key="policy_sample_rate">1</gprop:GenericProperty>
       <gprop:GenericProperty key="policy_elastic_margin">8</gprop:GenericProperty>
       <gprop:GenericProperty key="policy_attack">8</gprop:GenericProperty>
       <gprop:GenericProperty key="policy_decay">9</gprop:GenericProperty>
     </Item>
   </Section>
</Envelope>
```

Console (`console`) Resource

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`
- `enable-log` – Enable or disable virtual console logging for this console

EXAMPLE 37 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>
        <gprop:GenericProperty key="enable-log">on</gprop:GenericProperty>
      </Item>
    </Section>
```

```
  </Content>
</Envelope>
```

Domain Migration

This example shows what is contained in the <data> section for a ldm migrate-domain command.

- 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 properties in the <ldom_info> section for the target domain describe the machine to which to migrate and the details needed to migrate to that machine:
  - The target property specifies the name of the target machine.
  - The username property specifies the login user name for the target machine. The user name must be SASL 64-bit encoded.
  - The password property specifies the password to use for logging into the target machine. The password must be SASL 64-bit encoded.
  - The certificate property specifies whether to use SSL certificates for the migration operation.
  - The dry_run property specifies whether to perform a dry run of the migration operation.
  - The force property specifies whether to force the migration.
  - The spconfig property specifies the name of the configuration to save to the SP on the source machine and the target machine following the migration.

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.

EXAMPLE 38 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">system2</gprop:GenericProperty>
```

```
        <gprop:GenericProperty key="username">user1</gprop:GenericProperty>
        <gprop:GenericProperty key="password">user1-password</gprop:GenericProperty>
        <gprop:GenericProperty key="certificate"></gprop:GenericProperty>
        <gprop:GenericProperty key="dry_run"></gprop:GenericProperty>
        <gprop:GenericProperty key="force"></gprop:GenericProperty>
        <gprop:GenericProperty key="spconfig">system2cfg</gprop:GenericProperty>
      <Item>
    </Section>
  </Content>
</Envelope>
```

XML List Examples

The examples in this section show how to use input XML to perform list commands.

EXAMPLE 39 Example Blacklisted CPU XML

The example input XML for the `ldm list-devices -B` command for blacklisted cores is:

```
<cmd>
    <action>list-devices</action>
    <options>-B</options>
    <data version="3.0">
       <Envelope>
         <Section xsi:type="ovf:VirtualHardwareSection_Type">
           <Item>
             <rasd:OtherResourceType>core</rasd:OtherResourceType>
           </Item>
         </Section>
       </Envelope>
    </data>
</cmd>
```

The following example XML example shows the output of the `ldm list-devices -B` command:

```
<cmd>
  <action>list-devices</action>
  <options>-B</options>
  <data version="3.0">
    <Envelope>
      <References/>
      <Section xsi:type="ovf:VirtualHardwareSection_Type">
```

```
        <Item>
          <rasd:OtherResourceType>core</rasd:OtherResourceType>
          <gprop:GenericProperty key="cid">1</gprop:GenericProperty>
          <gprop:GenericProperty key="status">blacklisted</gprop:GenericProperty>
        </Item>
...
      </Section>
    </Envelope>
...
</cmd>
```

EXAMPLE 40 Example Blacklisted Memory XML

The example input XML for the `ldm list-devices -B` command for blacklisted memory is:

```
<cmd>
  <action>list-devices</action>
  <options>-B</options>
  <data version="3.0">
    <Envelope>
      <Section xsi:type="ovf:VirtualHardwareSection_Type">
        <Item>
          <rasd:OtherResourceType>memory</rasd:OtherResourceType>
        </Item>
      </Section>
    </Envelope>
  </data>
</cmd>
```

The following example XML example shows the output of the `ldm list-devices -B` command:

```
<cmd>
  <action>list-devices</action>
  <options>-B</options>
  <data version="3.0">
    <Envelope>
      <References/>
      <Section xsi:type="ovf:VirtualHardwareSection_Type">
        <Item>
          <rasd:OtherResourceType>memory</rasd:OtherResourceType>
          <gprop:GenericProperty key="pa">0xa30000000</gprop:GenericProperty>
          <gprop:GenericProperty key="size">2G</gprop:GenericProperty>
          <gprop:GenericProperty key="status">blacklisted</gprop:GenericProperty>
        </Item>
        <Item>
          <rasd:OtherResourceType>memory</rasd:OtherResourceType>
```

```
          <gprop:GenericProperty key="pa">0xbb0000000</gprop:GenericProperty>
          <gprop:GenericProperty key="size">2G</gprop:GenericProperty>
          <gprop:GenericProperty key="status">evac_pending</gprop:GenericProperty>
          <gprop:GenericProperty key="domain">ldg1</gprop:GenericProperty>
        </Item>
      </Section>
    </Envelope>
    <response>
      <status>success</status>
    </response>
  </data>
  <response>
    <status>success</status>
  </response>
</cmd>
<response>
  <status>success</status>
</response>
```

EXAMPLE 41 Using the `list-bindings -e` Command With XML

The following XML example shows the `ldm list-bindings -e` command output:

```
<cmd>
  <action>list-bindings</action>
  <options>-e</options>
  <data version="3.0">
    <Envelope>
      <References/>
      <Content xsi:type="ovf:VirtualSystem_Type" ovf:id="primary">
        ...
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>vldc</rasd:OtherResourceType>
            <gprop:GenericProperty key="service_name">primary-vldc3
              </gprop:GenericProperty>
            <gprop:GenericProperty key="client">SP</gprop:GenericProperty>
            <gprop:GenericProperty key="desc">spds</gprop:GenericProperty>
            <gprop:GenericProperty key="ldc">0x14</gprop:GenericProperty>
            <bind:Binding/>
          </Item>
        </Section>
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>vldc</rasd:OtherResourceType>
            <gprop:GenericProperty key="client">SP</gprop:GenericProperty>
            <gprop:GenericProperty key="desc">sunvts</gprop:GenericProperty>
```

```
            <gprop:GenericProperty key="ldc">0x6</gprop:GenericProperty>
            <bind:Binding/>
          </Item>
        </Section>
...
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>vldcc</rasd:OtherResourceType>
            <gprop:GenericProperty key="name">ds</gprop:GenericProperty>
            <gprop:GenericProperty key="service">primary-vldc0@primary
              </gprop:GenericProperty>
            <gprop:GenericProperty key="desc">domain-services</gprop:GenericProperty>
            <gprop:GenericProperty key="ldc">0x2</gprop:GenericProperty>
            <bind:Binding/>
          </Item>
        </Section>
...
</cmd>
```

EXAMPLE 42 Using the `list-netdev -b` Command With XML

The following XML example can be used to run the `ldm list-netdev -b primary` command. This command lists information about all the network devices the `primary` domain:

```
<cmd>
  <action>list-netdev</action>
  <options>-b</options>
  <data version="3.0">
    <Envelope>
      <References/>
      <Content xsi:type="ovf:VirtualSystem_Type" ovf:id="primary"></Content>
    </Envelope>
  </data>
</cmd>
```

The following XML example output is the result of the previous input:

```
<cmd>
  <action>list-netdev</action>
  <options>-b</options>
  <data version="3.0">
    <Envelope>
      <References/>
      <Content xsi:type="ovf:VirtualSystem_Type" ovf:id="primary">
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>network_device</rasd:OtherResourceType>
            <gprop:GenericProperty key="name">aggr0</gprop:GenericProperty>
```

```
            <gprop:GenericProperty key="class">AGGR</gprop:GenericProperty>
            <gprop:GenericProperty key="media">ETHER</gprop:GenericProperty>
            <gprop:GenericProperty key="state">down</gprop:GenericProperty>
            <gprop:GenericProperty key="speed">0M</gprop:GenericProperty>
            <gprop:GenericProperty key="over">net2,net3</gprop:GenericProperty>
          </Item>
        </Section>
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>network_device</rasd:OtherResourceType>
            <gprop:GenericProperty key="name">net0</gprop:GenericProperty>
            <gprop:GenericProperty key="class">PHYS</gprop:GenericProperty>
            <gprop:GenericProperty key="media">ETHER</gprop:GenericProperty>
            <gprop:GenericProperty key="state">up</gprop:GenericProperty>
            <gprop:GenericProperty key="speed">1000M</gprop:GenericProperty>
            <gprop:GenericProperty key="over">igb0</gprop:GenericProperty>
            <gprop:GenericProperty key="loc">/SYS/MB/NET0</gprop:GenericProperty>
          </Item>
        </Section>
...
      </Content>
    </Envelope>
...
  </data>
...
</cmd>
```

EXAMPLE 43 Using the `list-netstat` Command With XML

The following XML example can be used to run the `ldm list-netstat -u K -o net0 ldg1` command. This command lists information about the `net0` network device in kilobytes for the `ldg1` domain:

```
<cmd>
  <action>list-netstat</action>
  <data version="3.0">
    <Envelope>
      <References/>
      <Content xsi:type="ovf:VirtualSystem_Type" ovf:id="ldg1">
        <Section xsi:type="ovf:ResourceAllocationSection_Type">
          <Item>
            <gprop:GenericProperty key="network_device">net0</gprop:GenericProperty>
            <gprop:GenericProperty key="byte_unit">K</gprop:GenericProperty>
          </Item>
        </Section>
      </Content>
    </Envelope>
```

```
  </data>
</cmd>
```

The following XML example output is the result of the previous input:

```
<cmd>
  <action>list-netstat</action>
  <data version="3.0">
    <Envelope>
      <References/>
      <Content xsi:type="ovf:VirtualSystem_Type" ovf:id="ldg1">
        <Section xsi:type="ovf:VirtualHardwareSection_Type">
          <Item>
            <rasd:OtherResourceType>network_stat</rasd:OtherResourceType>
            <gprop:GenericProperty key="name">net0</gprop:GenericProperty>
            <gprop:GenericProperty key="ipackets">509.68K</gprop:GenericProperty>
            <gprop:GenericProperty key="rbytes">43856.31K</gprop:GenericProperty>
            <gprop:GenericProperty key="opackets">4.58K</gprop:GenericProperty>
            <gprop:GenericProperty key="obytes">413.03K</gprop:GenericProperty>
          </Item>
        </Section>
      </Content>
    </Envelope>
...
</cmd>
```

EXAMPLE 44 Using the `list-io` Command With XML

The following XML example can be used to run the `ldm list-io` command:

```
<cmd>
    <action>list-io</action>
    <data version="3.0"></data>
</cmd>
```

The following XML example output is the result of the previous input:

```
<cmd>
  <action>list-io</action>
  <data version="3.0">
    <Envelope>
      <References/>
      <Section xsi:type="ovf:VirtualHardwareSection_Type">
        <Item>
          <rasd:OtherResourceType>BUS</rasd:OtherResourceType>
          <gprop:GenericProperty key="dev">pci@400</gprop:GenericProperty>
          <gprop:GenericProperty key="alias">/SYS/MB/CMP0/PEX</gprop:GenericProperty>
          <gprop:GenericProperty key="status">IOV</gprop:GenericProperty>
```

```
          <gprop:GenericProperty key="domain">primary</gprop:GenericProperty>
          <gprop:GenericProperty key="type">BUS</gprop:GenericProperty>
          <gprop:GenericProperty key="bus">/SYS/MB/CMP0/PEX</gprop:GenericProperty>
        </Item>
        ...
      </Section>
    </Envelope>
  </data>
 ...
</cmd>
```

EXAMPLE 45 Using the `list-io -l` Command With XML

The following XML example can be used to run the `ldm list-io -l` command:

```
<cmd>
    <action>list-io</action>
    <options>-l</options>
    <data version="3.0"></data>
</cmd>
```

The following XML example output is the result of the previous input:

```
<cmd>
  <action>list-io</action>
    <options>-l</options>
    <data version="3.0">
      <Envelope>
        <References/>
        <Section xsi:type="ovf:VirtualHardwareSection_Type"
..
          <Item>
            <rasd:OtherResourceType>PCIE</rasd:OtherResourceType>
            <gprop:GenericProperty key="dev">pci@400/pci@2/pci@0/pci@8
              </gprop:GenericProperty>
            <gprop:GenericProperty key="alias">/SYS/MB/PCIE0</gprop:GenericProperty>
            <gprop:GenericProperty key="status">OCC</gprop:GenericProperty>
            <gprop:GenericProperty key="domain">primary</gprop:GenericProperty>
            <gprop:GenericProperty key="type">PCIE</gprop:GenericProperty>
            <gprop:GenericProperty key="bus">/SYS/MB/CMP0/PEX</gprop:GenericProperty>
            <gprop:GenericProperty key="subdev">SUNW,emlxs@0/fp/disk
              </gprop:GenericProperty>
            <gprop:GenericProperty key="subdev">SUNW,emlxs@0/fp/tape
              </gprop:GenericProperty>
            <gprop:GenericProperty key="subdev">SUNW,emlxs@0/fp@0,0
              </gprop:GenericProperty>
            <gprop:GenericProperty key="subdev">SUNW,emlxs@0,1/fp/disk
              </gprop:GenericProperty>
```

```
            <gprop:GenericProperty key="subdev">SUNW,emlxs@0,1/fp/tape
              </gprop:GenericProperty>
            <gprop:GenericProperty key="subdev">SUNW,emlxs@0,1/fp@0,0
              </gprop:GenericProperty>
          </Item>
...
            <rasd:OtherResourceType>VF</rasd:OtherResourceType>
            <gprop:GenericProperty key="dev">pci@500/pci@2/pci@0/pci@0/network@0,86
              </gprop:GenericProperty>
            <gprop:GenericProperty key="alias">/SYS/MB/PCIE5/IOVNET.PF0.VF3
              </gprop:GenericProperty>
            <gprop:GenericProperty key="status"/>
            <gprop:GenericProperty key="domain"/>
            <gprop:GenericProperty key="type">VF</gprop:GenericProperty>
            <gprop:GenericProperty key="class">NETWORK</gprop:GenericProperty>
            <gprop:GenericProperty key="bus">/SYS/MB/CMP1/PEX</gprop:GenericProperty>
            <gprop:GenericProperty key="user-assigned-name"/>
            <gprop:GenericProperty key="class:mac-addr">00:14:4f:f8:7f:88
              </gprop:GenericProperty>
            <gprop:GenericProperty key="class:alt-mac-addrs">
              00:14:4f:f9:27:75,00:14:4f:f8:b7:a4</gprop:GenericProperty>
            <gprop:GenericProperty key="class:mtu">1500</gprop:GenericProperty>
          </Item>
        </Section>
      </Envelope>
    </data>
...
</cmd>
```

XML Schemas

The XML schemas that are used by the Logical Domains Manager are located in the `/opt/SUNWldm/bin/schemas` directory. The file names are as follows:

- `cim-common.xsd` – `cim-common.xsd` schema
- `cim-rasd.xsd` – `cim-rasd.xsd` schema
- `cim-vssd.xsd` – `cim-vssd.xsd` schema
- `cli-list-constraint-v3.xsd` – `cli-list-constraint-v3.xsd` schema
- `combined-v3.xsd` – `LDM_interface` XML schema
- `event-v3.xsd` – `LDM_Event` XML schema
- `ldmd-binding.xsd` – `Binding_Type` XML schema
- `ldmd-property.xsd` – `GenericProperty` XML schema

- `ovf-core.xsd` – `ovf-core.xsd` schema
- `ovf-envelope.xsd` – `ovf-envelope.xsd` schema
- `ovf-section.xsd` – `ovf-section.xsd` schema
- `ovf-strings.xsd` – `ovf-strings.xsd` schema
- `ovfenv-core.xsd` – `ovfenv-core.xsd` schema
- `ovfenv-section.xsd` – `ovfenv-section.xsd` schema

♦ ♦ ♦ CHAPTER 4

Logical Domains Manager Discovery

This chapter provides information about discovering the Logical Domains Manager running on systems on a subnet.

Discovering Systems Running the Logical Domains Manager

Logical Domains Managers can be discovered on a subnet by using multicast messages. The `ldmd` daemon is able to listen on a network for a specific multicast packet. If that multicast message is of a certain type, `ldmd` replies to the caller. This enables `ldmd` to be discovered on systems that are running Oracle VM Server for SPARC.

Multicast Communication

This discovery mechanism uses the same multicast network that is used by the `ldmd` daemon to detect collisions when automatically assigning MAC addresses. To configure the multicast socket, you must supply the following information:

```
#define MAC_MULTI_PORT  64535
#define MAC_MULTI_GROUP  "239.129.9.27"
```

By default, *only* multicast packets can be sent on the subnet to which the machine is attached. You can change the behavior by setting the `ldmd/hops` SMF property for the `ldmd` daemon.

Message Format

The discovery messages must be clearly marked so as not to be confused with other messages. The following multicast message format ensures that discovery messages can be distinguished by the discovery listening process:

```
#include <netdb.h> /* Used for MAXHOSTNAMELEN definition */
#define MAC_MULTI_MAGIC_NO 92792004
#define MAC_MULTI_VERSION 1

enum {
 SEND_MSG = 0,
 RESPONSE_MSG,
 LDMD_DISC_SEND,
 LDMD_DISC_RESP,
};

typedef struct {
 uint32_t version_no;
 uint32_t magic_no;
 uint32_t msg_type;
 uint32_t resv;
 union {
  mac_lookup_t  Mac_lookup;
  ldmd_discovery_t Ldmd_discovery;
 } payload;
#define lookup  payload.Mac_lookup
#define discovery payload.Ldmd_discovery
} multicast_msg_t;

#define LDMD_VERSION_LEN 32

typedef struct {
 uint64_t mac_addr;
 char  source_ip[INET_ADDRSTRLEN];
} mac_lookup_t;

typedef struct {
 char  ldmd_version[LDMD_VERSION_LEN];
 char  hostname[MAXHOSTNAMELEN];
 struct in_addr ip_address;
 int  port_no;
} ldmd_discovery_t;
```

▼ How to Discover Logical Domains Managers Running on Your Subnet

1. **Open a multicast socket.**

   Ensure that you use the port and group information specified in “Multicast Communication” on page 77.

2. **Send a `multicast_msg_t` message over the socket.**

   The message should include the following:

   - Valid value for `version_no`, which is `1` as defined by `MAC_MULTI_VERSION`
   - Valid value for `magic_no`, which is `92792004` as defined by `MAC_MULTI_MAGIC_NO`
   - `msg_type` of `LDMD_DISC_SEND`

3. **Listen on the multicast socket for responses from Logical Domains Managers.**

   The responses must be a `multicast_msg_t` message with the following information:

   - Valid value for `version_no`
   - Valid value for `magic_no`
   - `msg_type` set to `LDMD_DISC_RESP`
   - Payload consisting of a `ldmd_discovery_t` structure, which contains the following information:
     - `ldmd_version` – Version of the Logical Domains Manager running on the system
     - `hostname` – Host name of the system
     - `ip_address` – IP address of the system
     - `port_no` – Port number being used by the Logical Domains Manager for communications, which should be XMPP port `6482`

   When listening for a response from Logical Domains Managers, ensure that any auto-allocation MAC collision-detection packets are discarded.

♦ ♦ ♦ CHAPTER 5

Using the Virtual Domain Information Command and API

This chapter describes the Virtual Domain Information command and API.

Using Virtual Domain Information Command

The `virtinfo` command enables you to gather information about a running virtual domain.

The following list shows some of the information that you can gather about a virtual domain by using the command or API:

- Domain type (implementation, control, guest, I/O, service, root)
- Domain name determined by the Virtual Domain Manager
- Universally unique identifier (UUID) of the domain
- Network node name of the domain's control domain
- Chassis serial number on which the domain is running

For information about the `virtinfo` command, see the `virtinfo`(8) man page.

Using the Virtual Domain Information API

You can also use the Virtual Domain Information API to create programs to gather information related to virtual domains. See the `libv12n`(3LIB) and `v12n`(3EXT) man pages.

Index