Virtualiron

PROGRAMMING GUIDE

Version: V4.4

NOTICE

The information in this document is provided as is and without warranty of any kind and is subject to change without notice. Virtual Iron Incorporated assumes no responsibility, and shall have no liability of any kind arising from supply or use of this publication or any material contained herein.

Virtual Iron® Software, Inc.
900 Chelmsford Street
Tower I, Floor 2
Lowell, MA 01851
http://www.virtualiron.com

TRADEMARKS

00081308

CONTENTS

Contents

Contents

PREFACE

Virtual Iron® VI-Center™ consists of a management server and a graphical client. The *Virtual Iron Programming Guide* provides samples of Java code to support client applications in virtual data centers.

AUDIENCE

This guide is for experienced application developers administrators. It assumes that you:

- Have read the VI-Center Administration Guide
- Are familiar with the Virtual Iron Management Server
- Have knowledge of Java development

CONVENTIONS

This book is distributed as an Adobe Acrobat document. It uses the following conventions:

- **Bold** type is used to call out the names of controls, and to describe management actions when using the Virtual Iron® management client.
- Light blue text is used for hyper-text links.
- Mono-spaced darker blue type is used for command examples.
- *Italicized text* is used to define or emphasize key terms.

LICENSE INFORMATION

The following Third Party Software modules are incorporated in or shipped with the Virtual Iron® software and are subject to the following terms and conditions:

Linux®

Linux® is a registered trademark of Linus Torvalds.

Redistributing GPL Source Files

You can redistribute and/or modify General Public License (GPL) software under the terms of the GPL. You may obtain a copy of the source code corresponding to the binaries for the GPL Software (the “GPL Source Files”) by downloading the GPL Source Files from http://www.virtualiron.com/Support/Downloads/Open-Virtual-Iron/index.php. This offer to obtain a copy of the GPL Source Files is valid for three years from the date you acquired this software product.

1

VIRTUAL IRON'S SDK

INTRODUCTION

The Virtual Iron API provides a Java interface to the Virtual Iron Management Server (VIMS). By including the Virtual Iron jars with your client software, you can connect to the VIMS and control your data center with all the same capabilities that are available through the VIMS user interface. For example, you can

- Create, start, and stop virtual servers
- Live migrate a running virtual server to a more capable node
- Collect performance statistics for nodes and virtual servers
- Listen for system events.

Getting Started

To develop applications using code in the Virtual Iron SDK, you will need a development environment as described below.

Supported Development Environments:

- Java Development Kit—JDK 5 or JDK 6 (JRE is not sufficient)
- Eclipse (tested with 3.3 Europa)

After downloading the SDK, unpack the .zip archive to a location on your machine. The directory contains subdirectories docs/, src/examples, system/resources and scripts/.

- src/examples—Java SDK examples
- system/resources/lib—VI API client jars
- scripts—Jython-based runner script examples

The SDK client jars must have the same version as the VIMS. An easy way to verify this is to connect to the VIMS with a connect script:

In Linux:

```
$cd scripts
$ ./runner.sh --inputfile=connect.py --vivmgr=http://<VIMS machine> --
username=admin --password=<password>
```

In Windows:

```
$cd scripts
$ runner.bat --inputfile=connect.py --vivmgr=http://<VIMS machine> --username=admin
--password=<password>
```

API JavaDoc

The VIMS API javadoc in its preliminary form is accessible from the VIMS at:

http://<vims:port>/resources/doc

However, it is recommended that you follow the examples in this document first, as they are more complete. The javadoc may provide some guidance with additional configuration options that are not documented here.

2

BASIC CONCEPTS

DEVELOPER BASICS

The Java-based API uses an object-oriented model for configuration and operation. Managed objects represent physical and virtual entities in the VIMS, such as a Node or VirtualServer. After connecting to the VIMS, you can use the ConfigurationManager to create, find, or delete these objects. Each object has its own get or set methods to change its configuration, as well as some object-specific operations, such as *reboot*, *start*, and *shutdown*.

Create object

The ConfigurationManager creates objects given the class and a unique name. Physical resources such as Nodes and Storage are discovered and are not explicitly created.

```
VirtualServer vs = (VirtualServer) ConfigurationManager.createObject(VirtualServer.class, "Web Server");
```

Examples of objects you may create are *VirtualServer*, *VirtualDataCenter*, and *EthernetNetwork*. Some objects are created automatically by the VIMS when they are discovered, such as *Node, Card, Port, StorageAreaNetworkDisk*, and *VirtualNetworkInterfaceCard*.

Delete object

To delete a managed object:

```
ConfigurationManager.deleteObject(VirtualServer.class, "Web Server");
```

When deleting an object, a VIMS rule may prevent the object from being deleted based on its state, such as deleting a node with running virtual servers. The virtual servers have to be stopped and removed first.

Find object

To find an object based on the object type and name:

```
VirtualServer vs = cm.findObject (VirtualServer, "My Web Server");
```

An object may also be found based on a handle. This is useful if the object may be deleted before it may be referenced. For instance, if a VirtualServer may be deleted, it is better to save the handle rather than the object reference for use later.

```
moh = vs.getHandle()
...
VirtualServer vs  = cm.findObject (moh);
```

Get all objects of specified Class

```
ArrayList <VirtualServer> vsList = ConfigurationManager.getObjects(VirtualServer.class);
```

Rename object

To rename an object:

```
VirtualServer vs = ...
vs.setName("New VirtualServer Name");
```

Check object status

To find an object's status, check its associated status event. To make more complex queries of an object's state refer to System Events.

```
Event event = vs.getStatusEvent();
if (event instanceof VirtualServerRunningEvent)
{
    // do something if VS is Running
}
```

3

CONNECTING TO A MANAGEMENT SERVER

CONNECTION SERVICE

To connect your client to the VIMS use the ConfigurationManager. The ConfigurationManager supports http, https, tcp and tcps connection protocols. The User Account username and password are configured on the VIMS. We recommend that you create a new User Account (apiuser) for the API calls separate from the admin account.

```
String url = "http://ms_hostname:8080";
String username = "apiuser";
String password = "Passw0rd";
ConfigurationManager cm = VirtualizationManager.connect(url,username,password);
```

Example

```
try {
   ConfigurationManager cm = VirtualizationManager.connect("http://host:8080", "apiuser", "Passw0rd");
}
catch (Exception e)
{
   // failed to connect
}
```

Chapter 3 CONNECTING TO A MANAGEMENT SERVER

4

JOB FRAMEWORK

MANAGING IN A MULTI-USER ENVIRONMENT

VI-Center uses a Job operations framework that supports a flexible approach to the reconfiguration of physical and virtual objects.

VI-Center is designed for use in an environment in which a number of users may have access to the same objects. VI-Center maintains an accurate and consistent view of the virtualization environment while users perform separate and simultaneous jobs.

Each configuration change is a *job*—a transaction performed by a single user. The steps that follow describe how resources are locked and released at the start and conclusion of each job.

What's in a Job?

A job is a configuration change that affects one or more physical or virtual objects. Examples of user operations that can be included in a job are:

- Renaming a virtual data center or other object
- Adding or deleting a virtual data center
- Adding VNICs or to a virtual server
- Moving a Virtual Server from one VDC to another
- Deleting a virtual server
- Changing the minimum and maximum values for a virtual server's memory and/or CPUs

Jobs and Resource Locking

A single job can contain one or many individual operations. Objects involved in a job are locked to all other users in the Virtualization environment until the job is completed or cancelled. This assures that a consistent and accurate view is maintained for all users.

The state of locked objects cannot be known until the locks are cleared. The state of the virtualization environment is always accurately reflected by the state of objects that are *not* locked.

The Virtual Iron job model provides a method to encapsulate one or more operations in a transaction. The operations are executed when you commit the job transaction. If an error occurs, the transaction is rolled back.

It is recommended that all set actions or operations that change the database in job transactions are encapsulated in a job. Since *get* operations do not change the database, they should not be encapsulated in a job.

Locks and Multiple User

A number of different users may perform jobs simultaneously, provided they are performed on different objects. For example, suppose User A has created *Finance-One* virtual data center and begins a job by moving nodes into another virtual data center. At the same time, User B modifies the resources of *Commodities* virtual data center. The objects remain locked until the jobs are completed.

Prior to completing a job, a lock can be cleared in two ways:

- By logging out the user who initiated the lock. This action can be performed by the user, or by the virtualization environment administrator.
- By direct action of the virtualization environment administrator.

Job Failure and Rollback

Job operations are validated by the VI-Center as they are added to the Job tab. *The failure of any operation cancels the entire job and all other operations it specifies.* The state of the virtualization environment is rolled back to what it had been prior to the start of the job. All locks in the operation are released.

What are <job.begin> and <job.commit> tags in examples?

To distinguish which commands should be run in a job in the examples, the command is wrapped in these tags:

```
<job.begin>
  operation
<job.commit>
```

Jobs and Events

When a job operation fails, one or more events may be generated.

```
job.commit();
StatusEvent event = job.getStatusEvent();
if (event instanceOf JobFailureEvent)
{
   //job failed, do something
}
```

Job States

While a Job is in progress, in can be in any of the states defined in Table .

Job States

Job State	Meaning
In Progress	A Job is running.
Complete	The Job has completed.
Failed	The Job has Failed. The virtualization environment has been rolled-back to its previous state and all locks have been released.
Aborting	The Job has been Aborted via console Abort command. The virtualization environment has been rolled-back to its previous state and all locks have been released.

PERFORMING JOB OPERATIONS

The sections that follow explain how to perform job operations within the virtualization environment™.

Example: Start VS Job

```
// simple job example
// create a job, add operation and commit
ConfigurationManager cm = ...
 try {
   // create a unique job based on VIMS localtime
   String jobName = java.lang.Long.toString(cm.getLocalTime());
   Job job = (Job) cm.createJob(jobName);
   // begin transaction
   job.begin();
   // start VS operation
   vs.start();
   job.addOperationDescription("Start VirtualServer", vs, vs, vs);
   // commit job - VS is now started
   job.commit();
}
catch (IllegalOperationException e)
{
   // if VIMS rule fails, resource locked, etc., abort
   job.abort();
}
catch (Exception e)
{
   // if anything else fails, abort
   job.abort();
}
```

Job

Method	Description
String getDescription()	Job description
ArrayList getOperationDescriptions()	Array list of job operation descriptions
long getStartTime()	Start time of when job creation started
long getEndTime()	End time of when job completed
String getUsername()	Get user name job was run by
Array getAssociatedEvents()	Get all events associated with this job

Example

To examine the existing job log for running jobs:

```
Foundry foundry = cm.getFoundryContext();
JobLog jobLog = foundry.getJobLog();
for (Job job : jobLog.getJobs())
 {
   //
}
```

5

PHYSICAL ENVIRONMENT

NODE INFORMATION

After a managed node PXE-boots and is successfully discovered, the VIMS creates the node objects as well as all discovered physical storage. All physical storage, including attached SCSI disks and network SAN storage arrays and LUNs are also discovered.

Do not create the nodes or storage through the API. Nodes can be moved, renamed, rebooted or deleted with the API, but not created.

Physical storage configuration including SAN zoning, LUN masking, etc., is outside the scope of the VIMS API.

The following node information is collected by the VIMS during node discovery.

Method	Description
long getAvailableMemory()	Available physical memory in MB
String getBIOSReleaseDate()	BIOS release date
String getBIOSVendor()	BIOS vendor
String getBIOSVersion ()	BIOS version
ArrayList getCapabilities()	CPU Flags
String getDeviceInformation()	Returns node discovery output in XML
boolean getHaltOnErrorFlag()	Node won't restart on error. Default: false.
long getMemory()	Node physical memory in MB
long getMemoryOverhead()	dom0 memory overhead in MB

Method	Description
int getNumberOfPopulatedProcessorSockets()	Number of populated processor sockets on node motherboard
int getNumberOfProcessorSockets()	Number of processor sockets on node motherboard
int getNumberOfThreadsPerCore()	Number of threads per processor core
int getProcessorSpeed()	Processor speed in KHz
boolean getProtectedFlag()	If true, node is excluded from LiveMaintenance and LiveCapacity operations
ArrayList getTotals()	
ArrayList getActivatedOpticalDisks()	
ArrayList Card getCards()	Get all interface cards in the node
ArrayList getProcessors()	Get processors on the node. (Processor table)
ArrayList getVirtualServers	Get virtual servers on the node
VDC getAssociatedVirtualDataCenter()	Get VDC that node is associated with

Each node has one or more cards that have the following types:

- EthernetAdapter
- FibreChannelHostBusAdapter
- InternetSmallComputerSystemInterfaceHostBusAdapter
- IntegratedDriveElectronicAdapter (CDROM)
- SmallComputerSystemInterfaceHostBusAdapter

Each card has one or more ports that have the following port types.

- BondPort
- EthernetPort
- FibreChannelPort
- InternetSmallComputerSystemInterfacePort
- InternetSmallComputerSystemInterfaceSoftPort
- IntegratedDriveElectronicChannelPort
- SmallComputerSystemInterfaceChannelPort
- StorageAreaNetworkPort

Examples

To find a node by name:

```
Node node = (Node) cm.findObject(Node.class, "LabServer-101");
```

To find a node by its management interface MAC address:

```
import com.virtualiron.vce.mgmt.api.manager.ConfigurationManager;
import com.virtualiron.vce.mgmt.api.physical.EthernetPort;
import com.virtualiron.vce.mgmt.api.physical.EthernetPortAdapter;
import com.virtualiron.vce.mgmt.api.physical.Node;
...
   ConfigurationManager cm = ...

   // find node by name
   Node node = (Node) cm.findObject(Node.class, "LabServer-101");

   // find node from any port
   EthernetPort port = ...
   Node node = (Node) port.getAssociatedCard().getAssociatedChassis();

   // find node by by management ethernet MAC address
   public static Node findNodeByEthernetPort(String macAddress)
   {
     ConfigurationManager cm = ...
     ArrayList <EthernetPort> ports = (EthernetPort) cm.getObjects(EthernetPort.class);
     Iterator portIter = ports.iterator();
     while (portIter.hasNext())
     {
        EthernetPort port = (EthernetPort) portIter.next();
        if (port.getMediaAccessControlAddress() == macAddress)
        {
           return (Node) port.getAssociatedCard().getAssociatedChassis();
        }
     }
     return;
   }
```

To find a node Ethernet port based on the MAC address:

```
import com.virtualiron.vce.mgmt.api.manager.ConfigurationManager;
import com.virtualiron.vce.mgmt.api.physical.EthernetPort;
import com.virtualiron.vce.mgmt.api.physical.EthernetPortAdapter;
import com.virtualiron.vce.mgmt.api.physical.Node;

...
```

To find an Ethernet port based on MAC address:

```
// find ethernet port based on MAC address
  public static EthernetPort findEthernetPort(String macAddress)
  {
    ConfigurationManager cm = ...
    ArrayList <EthernetPort> ports = (EthernetPort) cm.getObjects(EthernetPort.class);
    Iterator portIter = ports.iterator();
    while (portIter.hasNext())
    {
      EthernetPort port = (EthernetPort) portIter.next();
      if (port.getMediaAccessControlAddress() == macAddress)
      {
        // found a port with matching MAC address
        return port;
      }
    }
    return;
  }
```

To find an Ethernet port based on a MAC address on a specific node:

```
// find ethernet port based on MAC address on a specific node
public static EthernetPort findEthernetPort(String macAddress, Node node)
{
   for (Card card : node.getCards())
      if (card instanceof EthernetPortAdapter)
         for (Port port : card.getPorts())
            if (port.getMediaAccessControlAddress() == macAddress)
               return (EthernetPort) port;
   return;
}
```

To find a node's management Ethernet port:

```
// find node's management ethernet port
public static EthernetPort findManagementEthernetPort(Node node)
{
   for (Card card : node.getCards())
      if (card instanceof EthernetPortAdapter)
         for (Port port : card.getPorts())
            if (port.getMediaAccessControlAddress() ==
               node.getManagementControlMediaAccessControlAddress())
            {
               return (EthernetPort) port;
            }
   return;
}
```

Node Operations

To move a node to a virtual data center:

If a Node is in Running state with no Running VirtualServers, you can move it between VirtualDataCenters.

```
import com.virtualiron.vce.mgmt.api.physical.Node;
import com.virtualiron.vce.mgmt.api.virtual.VirtualDataCenter;
...
   Node node = ...
   VirtualDataCenter vdc = ...

   <job.begin...>
     // move node to new VDC
     vdc.addNode(node);
   <job.commit...>

   <job.begin...>
     // move node from VDC to unassigned
     vdc.removeNode(node);
   <job.commit...>
```

To rescan for new SAN disks:

If a new LUN or disk array is available in the SAN, this command discovers new storage without rebooting the node.

```
<job.begin...>
  node.rescanFibreChannelPorts()
 <job.commit...>
```

To reboot a managed node:

```
<job.begin...>
      node.reboot()
 <job.commit...>
```

Example

The management server uses IPMI or iLO if the node supports it. Otherwise, the management server signals the node agent to reboot the node. When the job completes, the node is still in Stopped state. You must wait until node discovery completes before the node is Running and can be used again.

```
import com.virtualiron.vce.mgmt.api.manager.ConfigurationManager;
import com.virtualiron.vce.mgmt.api.physical.Node;
import com.virtualiron.vce.mgmt.api.physical.Foundry;
import com.virtualiron.vce.mgmt.api.event.NodeStatusEvent;
import com.virtualiron.vce.mgmt.api.event.NodeRunningEvent;
...
```

```
ConfigurationManager cm = ...
Node node = ...
Foundry foundry = node.getFoundryContext();

<job.begin...>
   // reboot Node
   node.reboot();
<job.commit...>

// wait up to 60 seconds for Node to return to Running state
int timeout = 60 * 1000;       // 60s in milliseconds
EventLog eventLog = foundry.getEventLog();
ArrayList <ManagedObjectEvent> events = eventLog.listenForEvents(NodeStatusEvent.class, node,
timeout);
if (events.size() == 0)
{
   // no NodeStatusEvent received in 60 seconds
}
for (ManagedObjectEvent event : events)
   if (event instanceof NodeRunningEvent)
      // node is running
```

6

NETWORK CONFIGURATION

MAPPING PHYSICAL PORTS TO LOGICAL NETWORKS

Before configuring networks, you will map all available ports to the set of logical Ethernet networks created within VI-Center. This involves the following tasks:

- Logically connecting all ports on each node in the Resource Center to the networks you want them to use.
- Defining and specifying a name or alias for each network. The name should be a recognizable name that has to do with the network's use.

Default Management Network

Virtual Iron Extended Enterprise Edition (XEE) requires that you manage system nodes over a dedicated Ethernet network instead of a public network. The dedicated management network is detected and displayed for you. By default, the name of this network is its Class C IP address. The type of this network is *Management*.

Example

```
EthernetNetwork enet = ....
if (enet.getManagementFlag())
{
   // this is the node management network
}
```

Configuring an Ethernet Network

An Ethernet network connects node Ethernet ports and VirtualServer VNICs. The VIMS automatically creates a management Ethernet network for managing nodes.

To create an Ethernet network:

```
<job.begin>
  EthernetNetwork enet = cm.createObject(EthernetNetwork.class, "192.168.10.x");
<job.commit...>;
```

To add a node EthernetPort to an existing VLAN group:

```
<job.begin>
  EthernetNetwork enet = ...
  EthernetPort port = ...
  enet.addLogicalInterface(port,EthernetNetworkLogicalInterface.class);
<job.commit...>
```

Example

Create an Ethernet Network and add a node ethernet port.

```
import com.virtualiron.vce.mgmt.api.manager.ConfigurationManager;
import com.virtualiron.vce.mgmt.api.physical.EthernetNetwork;
import com.virtualiron.vce.mgmt.api.physical.EthernetPort;
import com.virtualiron.vce.mgmt.api.physical.EthernetLogicalInterface;
...
  ConfigurationManager cm = ...
  EthernetPort ethernetPort = ...
  <job.begin...>
    // create ethernet network
    EthernetNetwork enet =
    (EthernetNetwork) cm.createObject(EthernetNetwork.class, "Network
    Name");
    // add node ethernet port to ethernet network
    enet.addLogicalInterface(ethernetPort, EthernetLogicalInterface.class);
<job.commit...>
```

Configuring a VLAN

Virtual Iron supports multiple virtual LANs (VLANs) on the same port. Each VLAN is essentially an independent logical network operating with other VLANs over the same physical connection.

Configuring VLANs involves creating one or more VLAN Groups, each of which can house multiple VLANs. Each VLAN is assigned a distinct VLAN identification. The VLAN ID is used by an attached VLAN switch to segregate traffic among the different VLANs operating on the same link. Once a VLAN is configured, it functions exactly like a separate physical connection.

It is important to coordinate VLAN configuration with the administrator of attached VLAN switches, so that appropriate VLAN IDs are assigned to the VLANs you configure.

To create a VLAN group:

```
<job.begin...>
  VirtualLocalAreaNetworkGroup vlanGroup =
    cm.createObject(VirtualLocalAreaNetworkGroup.class, "VLAN Group");
<job.commit...>
```

To add a node EtherPort to an existing VLAN Group:

```
<job.begin...>
  VirtualLocalAreaNetworkGroup vlanGroup = ...
  EthernetPort port = ...
  vlanGroup.addLogicalInterface(port, VirtualLocalAreaNetwrkLogicalInterface.class);
<job.commit...>
```

To create a VLAN network with VLAN ID 101 and add to an existing VLAN Group:

```
<job.begin...>
  VirtualLocalAreaNetworkGroup vlanGroup = ...
  VirtualLocalAreaNetwork vlan = cm.createObject(VirtualLocalAreaNetwork.class, "VLAN 101");
  vlan.setVlanID(101);
  vlanGroup.addVirtualLocalAreaNetworkDevice(vlan);
<job.commit...>
```

Example

Create a VLAN Group with a node Ethernet port and a VLAN.

```
import com.virtualiron.vce.mgmt.api.manager.ConfigurationManager;
import com.virtualiron.vce.mgmt.api.physical.VirtualLocalAreaNetworkGroup;
import com.virtualiron.vce.mgmt.api.physical.VirtualLocalAreaNetwork;
import com.virtualiron.vce.mgmt.api.physical.VirtualLocalAreaNetworkLogicalInterface;
import com.virtualiron.vce.mgmt.api.physical.EthernetPort;
...
  ConfigurationManager cm = ...
  EthernetPort ethernetPort = ...
  <job.begin...>
    VirtualLocalAreaNetworkGroup vlanGroup =
      (VirtualLocalAreaNetworkGroup) cm.createObject(VirtualLocalAreaNetworkGroup.class,
"Network Group Name");
  VirtualLocalAreaNetwork vlan =
  (VirtualLocalAreaNetwork)
  cm.createObject(VirtualLocalAreaNetwork.class, "VLAN Name");
  // add VLAN to VLAN Group
  vlanGroup.addVirtualLocalAreaNetworkDevice(vlan);
  // set VLAN ID
  vlan.setVLanID(101);
  // add node ethernet port to VLAN Group
```

```
    vlanGroup.addLogicalInterface(ethernetPort, VirtualLocalAreaNetworkLogicalInterface.class);
    <job.commit...>
...
import com.virtualiron.vce.mgmt.api.virtual.VirtualServer;
import com.virtualiron.vce.mgmt.api.virtual.VirtualNetworkInterfaceCard;
    /// vlan may be assigned to a VS VNIC
    VirtualServer vs = ...
    VirtualNetworkInterfaceCard vnic = vs.getVirtualNetworkInterfaceCards().get(0);
    vlan.addVirtualNetworkInterfaceCard(vnic);
```

Configuring an iSCSI Network

Internet SCSI (iSCSI) is an IP-based storage networking standard that allows connection to data storage facilities over local and wide area networks and the internet. Though it runs at lower bandwidth than Fibre Channel networks, iSCSI performs the same basic function as Fibre Channel. However, iSCSI has distinct advantages. Since it runs over Ethernet, it does not require an HBA or other dedicated FC device.

iSCSI Configuration Guidelines

Follow these guidelines when configuring switches fro iSCSI networks:

- Make sure that the iSCSI storage server is on the same subnet as the network you are connecting from.
- Do not mix iSCSI traffic with other LAN traffic. Use VLANs if necessary to isolate the iSCSI network on a switch.
- Use managed 1 GB switches.
- Set all target, initiator and inter-switch ports to 1 GB full duplex (auto-negotiate OFF).
- Set flow control to AUTO on target, initiator, and inter-switch ports.
- When interconnecting switches, make sure you have sufficient bandwidth connections between switches. Use fiber interconnections or Link Aggregated Groups to provide adequate bandwidth.
- In the case of Cisco 2970, enable Cisco's Spanning-Tree PortFast option.

Note: Virtual Iron® recommends that the iSCSI network is on a dedicated network that is not accessible to virtual servers. This assures the security of the storage traffic and better performance.

Virtual Iron® supports iSCSI connections between managed nodes and SAN resources.

To create an iSCSI network:

```
<job.begin...>
    InternetSmallComputerSystemInterfaceNetwork inet  =
        cm.createObject(InternetSmallComputerSystemInterfaceNetwork.class, "iSCSI Net");
<job.commit...>
```

To add an Ethernet port to an existing iSCSI network:

You must create an iSCSI soft port and assign an IQN. Reboot the node with the Ethernet port for the change to take effect. After reboot, iSCSI disks are discovered by the VIMS and are accessible.

```
<job.begin...>
  InternetSmallComputerSystemInterfaceNetwork inet  =
  InternetSmallComputerSystemInterfaceCard iscsiCard =
  EthernetPort port = ...
  String iqn = "iqn.2007-01..."
  InternetSmallComputerSystemInterfaceSoftPort  =
    iscsiCard.createPort(InternetSmallComputerSystemInterfaceSoftPort.class, 1);
  InternetSmallComputerSystemInterfaceEthernetLogicalInterface enetInterface =
  port.createLogical(InternetSmallComputerSystemInterfaceEthernetLogicalInterface.class);
  iscsiSoftPort.addInternetSmallComputerSystemInterfaceEthernetLogicalInterface(enetInterface);
  iscsiSoftPort.setInitiatorName(iqn);
<job.commit...>
```

To add an iSCSI portal to the iSCSI network:

The targetAddress string must include the IP address and port.

```
<job.begin...>
  InternetSmallComputerSystemInterfaceNetwork inet = ...
  String targetAddress = "192.168.10.200:3240";
  inet.addTargetAddress(targetAddress);
<job.commit...>
```

To enable jumbo frames on the iSCSI network:

Change the network MTU to 9000 bytes. Default is 1500.

Note: All nodes Ethernet switches and iSCSI portals must support jumbo frames.

```
 <job.begin...>
  InternetSmallComputerSystemInterfaceNetwork inet = ...
  int jumboMTU = 9000;
  inet.setMaximumTransmissionUnit(jumboMTU);
<job.commit...>
```

Example

Create an iSCSI network, add iSCSI targets and add node Ethernet ports.

```
import com.virtualiron.vce.mgmt.api.manager.ConfigurationManager;
import com.virtualiron.vce.mgmt.api.physical.InternetSmallComputerSystemInterfaceNetwork;
import
com.virtualiron.vce.mgmt.api.physical.InternetSmallComputerSystemInterfaceEthernetLogicalInterface;
import com.virtualiron.vce.mgmt.api.physical.InternetSmallComputerSystemInterfaceSoftPort;
import com.virtualiron.vce.mgmt.api.physical.InternetSmallComputerSystemInterfaceCard;
import com.virtualiron.vce.mgmt.api.physical.EthernetPort;
...
  ConfigurationManager cm = ...
  EthernetPort ethernetPort = ...
  InternetSmallComputerSystemInterfaceCard iscsiCard = ...
  <job.begin...>
    InternetSmallComputerSystemInterfaceNetwork iscsiNetwork =
      InternetSmallComputerSystemInterfaceNetwork)
      cm.createObject(InternetSmallComputerSystemInterfaceNetwork.class, "iSCSI Network
      Name");
    // add iSCSI Target IP Address (10.1.20.240) and Port (3240)
    iscsiNetwork.addTargetAddress("10.1.20.240:3240");
    // to add node ethernet port to iscsi network, create soft port on node
    InternetSmallComputerSystemInterfaceSoftPort iscsiSoftPort =
      iscsiCard.createPort(InternetSmallComputerSystemInterfaceSoftPort.class, 1);
    InternetSmallComputerSystemInterfaceEthernetLogicalInterface ethernetInterface =
ethernetPort.createLogical(InternetSmallComputerSystemInterfaceEthernetLogicalInterface.class);
iscsiSoftPort.addInternetSmallComputerSystemInterfaceEthernetLogicalInterface(ethernetInterface);
    // create unique iSCSI initiator IQN
    String mac = ethernetPort.getMediaAccessControlAddress();
    String iqn = "iqn.2007-01.com.virtualiron:01:" + mac;
    iscsiSoftPort.setInitiatorName(iqn);
    iscsiNetwork.addLogicalInterface(ethernetInterface);
    // by default, MTU is 1500 but to use Jumbo frames, set to 9000
    iscsiNetwork.setMaximumTransmissionUnit(9000);
    iscsiSoftPort.provision();
  <job.commit...>
```

7

CONFIGURING STORAGE

ADVANTAGES OF MANAGED STORAGE

VI-Center presents a unified framework for controlling local or SAN disks that are accessible to managed nodes and their virtual servers. In this framework, you create one or more *disk groups* (DGs), and subdivide them into one or more *logical disks*. Logical disks have additional utility in that they can be cloned (copied) and exported for use by other virtual servers. Neither of these capabilities is available in the management of raw SAN disks.

Overlaying a storage framework on available physical storage has distinct advantages. All physical LUNs can be administered from VI-Center. Following initial information exchange with the SAN administrator, the VI-Center administrator can use this framework to configure and manage logical disks on the SAN.

Virtual Servers (VSs) can be configured to access two types of disks:

- **Logical disks**—High-performance disks that can be assigned to one or more virtual servers. The size can be smaller than the underlying physical disk. Supported on Fibre Channel, iSCSI, and local storage.
- **Raw SAN disks**—High performance. One or more virtual servers use one entire physical LUN.

Note: For best performance, Virtual Iron recommends that all storage be in logical disks.

The following table summarizes the functional differences between these disk types.

Feature	Logical	Raw
Shareable between VSs	X	X
High-performance	X	X
Supported on iSCSI SAN	X	X
Supported on Fibre Channel SAN	X	X
Supported on local storage	X	---
Ability to sub divide physical disk	X	---
Cloning capability	X	---
Dynamic VHD file import	---	---
Fixed VHD file import	X	---
Export capability	X	---

DISCOVERY AND MANAGEMENT OF PHYSICAL DISKS

Virtual Iron supports three types of physical disks:

- Fibre Channel SAN disks
- iSCSI SAN disks
- SATA, SAS, or parallel SCSI node-local disks

Local IDE drives are not supported.

All physical disks are automatically discovered during the node boot process. If you reconfigure LUNs while a node is running, before adding them to the system, rescan the SAN ports to update the management server. This step assures synchronization between the management server and your environment.

To access physical SAN resources, the managed node hosting it requires a host bus adapter (HBA) or network interface card (NIC). Each HBA has a unique WWNN or iqn, which you need to provide to your SAN or iSCSI administrator. The administrator makes specific LUNs visible to managed nodes in the Virtual Iron® framework.

Once this information has been configured in your SAN infrastructure, SAN targets and LUNS become visible to the managed nodes programmed to access them.

Find Physical Storage

To find a node's locally-attached SCSI disks:

```
import com.virtualiron.vce.mgmt.api.manager.ConfigurationManager;
import com.virtualiron.vce.mgmt.api.physical.SmallComputerSystemInterfaceHardDisk;
import com.virtualiron.vce.mgmt.api.physical.SmallComputerSystemInterfaceAdapter;
import java.util.ArrayList;
...
   // return list of all local SCSI disks on a node
   public static ArrayList<SmallComputerSystemInterfaceHardDisk> getLocalDisks(Node node)
   {
      ArrayList <SmallComputerSystemInterfaceHardDisk> disks = new
      ArrayList<SmallComputerSystemInterfaceHardDisk>();
      for (Card card : node.getCards())
      {
         // if card is a SCSI Adapter, check port
         if (card instanceof SmallComputerSystemInterfaceAdapter)
         {
             for (Port port : card.getPorts())
            {
               for (SmallComputerSystemInterfaceDevice disk :
               port.getSmallComputerSystemInterfaceDevices())
                  if (disk instanceof SmallComputerSystemInterfaceHardDisk)
                  disks.add(disk);
            }
         }
      }
   return disks;
}

// return list of all free local SCSI disks on a node
public static ArrayList <SmallComputerSystemInterfaceHardDisk> getLocalDisks(Node node)
{
   ArrayList <SmallComputerSystemInterfaceHardDisk> disks = new
   ArrayList<SmallComputerSystemInterfaceHardDisk>(); for (Card card : node.getCards())
   {
      // if card is a SCSI Adapter, check port
      if (card instanceof SmallComputerSystemInterfaceAdapter)
      {
         for (Port port : card.getPorts())
         {
            for (SmallComputerSystemInterfaceDevice disk :
            port.getSmallComputerSystemInterfaceDevices())
               // if disk is not associated with a VS or diskgroup, it is free
               if (disk instanceof SmallComputerSystemInterfaceHardDisk && disk.getVirtualServers().
               size() == 0 && disk.getAssociatedLvmLocalGroup() == null)
```

```
                disks.add(disk);
            }
          }
        }
        return disks;
    }
```

To find all network storage (SAN LUN) accessible by a node:

```
import com.virtualiron.vce.mgmt.api.manager.ConfigurationManager;
import com.virtualiron.vce.mgmt.api.physical.FibreChannelHostBusAdapter;
import com.virtualiron.vce.mgmt.api.physical.InternetSmallComputerSystemInterfaceHostBusAdapter;
import com.virtualiron.vce.mgmt.api.physical.StorageAreaNetworkDisk;
import java.util.HashSet;
...
    // get all Network Disks (LUN) accessible from a node
    public static HashSet<StorageAreaNetworkDisk> getNetworkDisks(Node node)
    {
        HashSet<StorageAreaNetworkDisk> disks = new HashSet<StorageAreaNetworkDisk>();
        for (Card card : node.getCards())
        {
            // if card is a FC or iSCSI Adapter, check port
            if (card instanceof FibreChannelHostBusAdapter ||
                card instanceof InternetSmallComputerSystemInterfaceHostBusAdapter
            {
                for (Port port : card.getPorts())
                    for (Path path : port.getStorageAreaNetworkPaths())
                        for (StorageAreaNetworkDevice disk = path.getAssociatedStorageAreaNetworkDevice()
                            ff (disk instanceof StorageAreaNetworkDisk)
                                disks.add(disk);
            }
          }
        return disks;
    }
```

To find whether network storage is available:

```
StorageAreaNetworkDevice disk = ...
    // if disk isn't assigned to a VS or in a DiskGrou
    if (disk instanceof StorageAreaNetworkDisk && disk.getVirtualServers().size() == 0 &&
            disk.getAssociatedLvmNetworkGroup() == null)
    {
        //disk is available.
    }
```

CREATING DISK GROUPS

The VI-Center supports the creation and assignment of logical disks on local drives or SANs. Each disk group consists of one or more SANs or local disks. You make use of the storage contained in a disk group by subdividing it into one or more logical disks. The process for creating logical disks on a local drive is the same as for creating them on the SAN.

You can copy or clone logical disks for use by other virtual servers, and they can also be be administratively exported or imported from a directory within VI-Center. Once you create logical disks, they are visible to all virtual servers hosted on the virtual data center.

Creating Disk Groups

Create a local or network DiskGroup; find local and network disk groups:

```
import com.virtualiron.vce.mgmt.api.manager.ConfigurationManager;
import com.virtualiron.vce.mgmt.api.physical.Foundry;
import com.virtualiron.vce.mgmt.api.physical.Node;
import com.virtualiron.vce.mgmt.api.physical.LwmGroup;
import com.virtualiron.vce.mgmt.api.physical.LogicalDisk;
import com.virtualiron.vce.mgmt.api.virtual.VirtualDataCenter;
import java.util.ArrayList;
...
```

To create a local disk disk with one or more local SCSI disks:

```
 <job.begin...>
      // create local disk group with one or more
      // local node SCSI disks
      ArrayList <SmallComputerSystemInterfaceHardDisk> diskList = ...
      LvmLocalGroup diskGroup = node.createLvmLocalGroup(diskList);
  <job.commit...>
```

To create a DiskGroup with one or more network SAN LUNs:

```
<job.begin...
  VirtualDataCenter vdc = ...
  ArrayList <StorageAreaNetworkDevice> DiskList = ...
  LvmNetworkGroup diskGroup = foundry.createLvmNetworkGroup(DiskList, vdc);
<job.commit...>
```

To delete a network disk group:

```
<job.begin...>
  // delete network disk group
  foundry.deleteLvmNetworkGroup(diskGroup);
<job.commit...>
```

To delete a local disk group:

```
<job.begin...>
  // delete local disk group
  node.deleteLvmLocalGroup(diskGroup);
<job.commit...>
```

To find the local disk groups:

```
ArrayList<LvmLocalGroup> Node.getLvmLocalGroups()
```

To find the network disk groups accessible by a VDC:

```
ArrayList <LvmNetworkGroup> VirtualDataCenter.getLvmNetworkGroups()
```

Creating Logical Disks

To create a logical disk in a local or network disk group:

```
// create a logical disk in bytes, size is in GB
LvmLocalGroupDiskGroup = ...
long bytes = Units.StorageUnit.convertToBytes(size, Units.StorageLogicalDisk logicalDisk)
diskGroup.createLogicalDisk(bytes, Units.StorageUnit.BYTE);
```

To delete a logical disk:

```
<job.begin...>
  // delete logical disk from diskGroup
  LvmDiskGroup diskGroup = logicalDisk.getParentLvmDiskGroup();
  diskGroup.deleteLogicalDisk(logicalDisk);
<job.commit...>
```

To find all logical disks in a disk group:

```
LvmLogicalDiskGroup ldg = diskGroup.getLvmLogicalDiskGroup();
ArrayList <LogicalDisk> LDisks = ldg.getLogicalDisks();
```

CLONING DISKS

Virtual Iron® allows you to clone (copy) logical disks and their associated content to any other disk group under management. Note that logical disks can only be cloned when they are not in use by a virtual server.

Virtual disks may be cloned to logical disks or other virtual disks.

From:	To:
Logical Disk—Fixed	Logical Disk—Fixed
Virtual Disk—Dynamic	Logical Disk—Fixed or Virtual Disk—Dynamic
Virtual Disk—Fixed	Logical Disk—Fixed

Cloning a Logical Disk

The source disk can not be part of a running virtual server. During the clone operation, the node from which the clone is made is locked.

To clone a logical disk to a new DiskGroup:

```
LogicalDisk sourcedisk = ...
LvmNetworkGroup diskGroup = ...
LogicalDisk cloneDisk = sourcedisk.clone(diskGroup, Lvm.LvmType.LogicalDisk);
```

EXPORTING AND IMPORTING A LOGICAL DISK

Use the export and import functions to move VHD files on the VI-Center from and to the disk groups in your data center. This is also useful if you want to move a logical disk to a new disk group in a different VDC.

There are two types of VHD files, each of which is stored differently in the data center:

- **Dynamic**—As data is written to the dynamic disk, the file grows as large as the maximum size that was specified when it was created. Unused space is not included in the image, reducing the size of the dynamic disk file. If you import a dynamic VHD, the file is imported to a virtual disk.
- **Fixed**—A fixed-size hard drive is one in which space is allocated when the VHD is created. The size of the disk does not change when data is added or deleted. If you import a fixed VHD, the files are imported to a logical disk.

Commonly used VHD files on the VI-Center may be made available to multiple users. Use the import and export functions to move VHD files to the VI-Center for use in virtual servers.

During the import or export operation, the node is locked.

Example: Exporting a Logical Disk

Export LogicalDisk to VIMS VirtualizationManager/vdisks directory as a fixed vhd file.

```
import com.virtualiron.vce.mgmt.api.manager.ConfigurationManager;
import com.virtualiron.vce.mgmt.api.manager.VdiManager;
import com.virtualiron.vce.mgmt.api.physical.Foundry;
import com.virtualiron.vce.mgmt.api.physical.LogicalDisk;
import com.virtualiron.vce.mgmt.api.physical.VirtualDiskImage;

...
   ConfigurationManager cm = ...
   LogicalDisk logicalDisk = ...
   Foundry foundry = cm.getFoundryContext();
   VdiManager vdimgr = foundry.getVdiManager();
   String name = "exportDisk.vhd";

   <job.begin...>
      // find vhd file, name, in VirtualIron/vdisks directory
      VirtualDiskImage vdi = vdimgr.findAssociatedVirtualDiskImage(name);
      if (null == vdi)
      {
         VirtualDiskImage vdi = vdimgr.createVirtualDiskImage(name);
      }

      // export LogicalDisk as a fixed vhd file to VirtualIron/vdisks directory
      logicalDisk.exportVirtualDiskImage(vdi);
   <job.commit...>
```

Example: Importing Logical Disk

To import a logical disk from the VIMS directory:

Import Dynamic vhd Disk as VirtualDisk. Import Fixed vhd disk as LogicalDisk.

```
import com.virtualiron.vce.mgmt.api.manager.ConfigurationManager;
import com.virtualiron.vce.mgmt.api.manager.VdiManager;
import com.virtualiron.vce.mgmt.api.physical.Foundry;
import com.virtualiron.vce.mgmt.api.physical.Lwm;
import com.virtualiron.vce.mgmt.api.physical.LwmGroup;
import com.virtualiron.vce.mgmt.api.physical.VirtualDiskImage;
import com.virtualiron.vce.mgmt.api.physical.VirtualDisk;
import com.virtualiron.vce.mgmt.api.physical.LogicalDisk;

...

   ConfigurationManager cm = ...
   LwmGroup diskGroup = (LvmGroup) cm.findObject(LvmGroup.class, ...);
```

```
// find vhd file, name, in VirtualIron/vdisks directory
Foundry foundry = cm.getFoundryContext();
VdiManager vdimgr = foundry.getVdiManager();
String name = "importDisk.vhd";

VirtualDiskImage vdi = vdimgr.findAssociatedVirtualDiskImage(name);

<job.begin...>
   // if vhd file is Dynamic, import as VirtualDisk
   if (vdi.getFileFormatType() == VirtualDiskImage.VhdType.Dynamic)
   {
     VirtualDisk virtualDisk = (VirtualDisk)
          diskGroup.importVirtualDiskImage(vdi, Lvm.LvmType.VirtualDisk);
   }

   // if vhd file is Fixed, import as LogicalDisk
   if (vdi.getFileFormatType() == VirtualDiskImage.VhdType.Fixed)
   {
     LogicalDisk logicalDisk = (LogicalDisk)
          diskGroup.importVirtualDiskImage(vdi, Lvm.LvmType.LogicalDisk);
   }
<job.commit...>
```

8

VIRTUAL ENVIRONMENT

CREATING VIRTUAL DATA CENTERS

A virtual data center (VDC) is an administrative entity that consists of one or more managed nodes. Each VDC functions as a true data center—a group of nodes that have been segregated to meet specific business needs.

The hardware resources in each VDC are only available to the virtual servers hosted by them. In the same way, other VDCs function as a separate set of physical resources. The many VDCs into which you can separate all the nodes under management can be likened to the partitions into

To create a virtual data center:

```
<job.begin...>
cm = VirtualizationManager.getConfigurationManager();
Foundry foundry = cm.getFoundryContext();
VirtualDataCenter vdc = configManager.createObject(VirtualDataCenter.class, "New VDC");
foundry.addVirtualDataCenter(vdc);
<job.commit>
```

To delete a virtual data center

To delete a virtual data center make a managed object delete call. All nodes must be removed from a VDC before it can be deleted.

```
<job.begin...>
cm.deleteObject(vdc);
<job.commit>
```

To move a Node to a VDC:

Note that you can only move a node if it has no running virtual servers.

```
<job.begin...>
Node node = ...
vdc.addNode(node);
<job.commit>
```

To remove a Node from a VDC

```
<job.begin...>
vdc.removeNode(node);
<job.commit>
```

To get all Nodes in a VDC

```
<job.begin...>
ArrayList <Node> Nodes = vdc.getNodes();
<job.commit>
```

CONFIGURING A VIRTUAL SERVER

Make sure you have completed the following before you create a virtual server:

- Prepared nodes to be managed.
- Cabled all nodes properly.
- Installed the Virtual Iron® software.
- Established a connection to the Virtual Iron management server.
- Defined connections between system nodes and physical networks.
- Created a Virtual Data Center (VDC).
- Assigned a node to the VDC.

Once you complete these tasks, you have created the basis for virtualizing the resources of one or more nodes. All that remains is to create virtual servers.

Virtual Server Configuration

To create a virtual server, you must first create the object:

```
VirtualServer vs = (VirtualServer)cm.createObject(VirtualServer.class), "New VS");
```

Method	Description
addBootDeviceToFront(BootDevice,boolean)	Set BootDevice. If true, set BootType.
addStorageDevice(StorageDevice)	Add storage device.
addVirtualNetworkInterfaceCard(VNIC)	Add VNIC, from license file.
changeActiveCdrom(StorageDevice)	
setAutoRecoveryFlag(boolean)	If true, auto recovery policy.
setMemory(long)	Set memory size in MB.
setMaxCpuUtilization(int)	Set Max allowed CPU utilization. Default:100
setNumberOfProcessors(int)	Set number of processors.
setOperatingSystemType(String)	Set operating system type. See Virtual Server, Operating Systems.
setPriority(int)	Set priority for scheduling. Default: 100.
setUseVsToolsFlag(boolean)	If true, accelerated drivers will be used.

Next, enable or disable VS Tools. It is important to do this first, as rules may have an effect on other VS parameters, such as the number of storage devices or the number of processors.

Then, set VirtualServer memory, number of processors, and so on.

```
vs.setMemory(512)
```

To start a virtual server, you must specify a boot device. Add the boot device to the storage device list. If setBootType is enabled, the boot type is set.

To add a boot device to a VirtualServer:

```
<job.begin...>
  vs.addBootDeviceToFront(BootDevice, boolean);
<job.commit...>
```

To set the VS Operating System

```
<job.begin...>
  vs.setOperatingSystemType(VirtualServer.OperatingSystems.VS_WIN_SERVER_2003);
<job.commit...>
```

Refer to the following table:

com.virtualiron.vce.mgmt.api.virtual.VirtualServer.OperatingSystems		
public static final String	VS_NONE	"None"
public static final String	VS_OTHER_LINUX	"Other Linux"
public static final String	VS_OTHER_WIN	"Other Windows"
public static final String	VS_RH_3	"Red Hat Enterprise Linux 3"
public static final String	VS_RH_4	"Red Hat Enterprise Linux 4"
public static final String	VS_RH_5	"Red Hat Enterprise Linux 5"
public static final String	VS_SUSE_10	"Suse Linux Enterprise Server 10"
public static final String	VS_SUSE_9	"Suse Linux Enterprise Server 9"
public static final String	VS_WIN_2000	"Microsoft Windows 2000"
public static final String	VS_WIN_SERVER_2003	"Microsoft Windows Server 2003"
public static final String	VS_WIN_VISTA	"Microsoft Windows Vista"
public static final String	VS_WIN_XP	"Microsoft Windows XP"

The following is a list of valid BootDevices:

- IntegratedDriveElectronicOpticalDisk
- SmallComputerSystemInterfaceOpticalDisk
- NetworkBlockDeviceCdromImage
- StorageAreaNetworkDisk
- LogicalDisk
- VirtualDisk
- VirtualNetworkInterfaceCard (pxeboot only)

To add an additional storage device to a VirtualServer:

```
<job.begin...>
  vs.addStorageDevice(StorageDevice);
job.commit...>
```

The following is a list of valid StorageDevices:

- IntegratedDriveElectronicOpticalDisk
- SmallComputerSystemInterfaceOpticalDisk
- NetworkBlockDeviceCdromImage
- NetworkBlockDeviceFloppyImage
- StorageAreaNetworkDisk
- LogicalDisk
- VirtualDisk

To add a VNIC, an available VNIC is taken from the license file:

```
Foundry foundry =
LicenseManager
VNIC vnic =
vs.add VNIC(vnic);
// get an available VNIC from license file
Foundry foundry = cm.getFoundryContext();
LicenseManager licenseManager = foundry.getLicenseManager();
VirtualNetworkInterfaceCard vnic = licenseManager.getAvailableVirtualNetworkInterfaceCard();
```

To get associated VDC:

```
VirtualDataCenter vdc = vs.getAssociatedVirtualDataCenter();
```

To get associated node:

If return value is null, virtual server is unassigned.

```
Node node = vs.getAssociatedNode();
```

Example: Configure a VirtualServer

Create a VirtualServer that boots from a logical disk and assign its VNIC to an Ethernet-Network. VirtualServer is put in VDC as Unassigned.

```
import com.virtualiron.vce.mgmt.api.manager.ConfigurationManager;
import com.virtualiron.vce.mgmt.api.manager.LicenseManager;
import com.virtualiron.vce.mgmt.api.physical.EthernetNetwork;
import com.virtualiron.vce.mgmt.api.physical.LogicalDisk;
import com.virtualiron.vce.mgmt.api.virtual.VirtualServer;
import com.virtualiron.vce.mgmt.api.virtual.VirtualNetworkInterfaceCard;


...
   ConfigurationManager cm = ...
   boolean setBootType = true;

   // find logical disk as boot disk
   LogicalDisk bootdisk = cm.findObject(LogicalDisk.class, ...);

   // find ethernet network to assign to VS VNIC
   EthernetNetwork enet = cm.findObject(EthernetNetwork.class, ...);

   // find VDC to put Unassigned VS
   VirtualDataCenter vdc = (VirtualDataCenter) cm.findObject(VirtualDataCenter.class, "My VDC");

   // may check if vs already exists before creating it
   VirtualServer vs = (VirtualServer) cm.findObject(VirtualServer.class, "New VirtualServer");

   try {
      Job job = ...
      job.begin();

      VirtualServer vs = cm.createObject(VirtualServer.class, "New VirtualServer");
      job.addOperationDescription("Create Virtual Server", vs, vs, vs);

      // disable VsTools
      vs.setUseVsToolsFlag(0);
      vs.setNumberOfProcessors(2);

      // memory in MB
      vs.setMemory(512);

      // add logical disk as boot disk
      vs.addBootDeviceToFront(bootdisk, setBootType);
      vs.setOperatingSystemType(VirtualServer.OperatingSystems.VS_WIN_SERVER_2003);
```

```
    // add VNIC to VS and Ethernet network
     vs.addVirtualNetworkInterfaceCard(vnic);
     enet.addVirtualNetworkInterfaceCard(vnic);

     // add VS to VDC as Unassigned
     vdc.addVirtualServer(vs);

     job.commit();
}
catch (Exception e)
{
     job.abort();
}
```

VirtualServer Information

For an existing virtual server, you can collect information about the configuration:

Method	Description
BootDevice getBootDevice()	Return BootDevice.
ArrayList getStorageDevices()	ArrayList of StorageDevices.
ArrayList getVirtualNetworkInterfaceCards()	ArrayList of VNICs.
boolean getAutoRecoveryFlag()	If true, AutoRecovery policy is enabled.
long getMemory()	Get memory size in MB,
int getMaxCpuUtilization()	Get max allowed CPU utilization.
int getNumberOfProcessors()	Get number of processors.
String getOperatingSystemType()	Get operating system type.
int getPriority()	Get priority for scheduling. Default: 100
boolean getUseVsToolsFlag()	If true, accelerated drivers are used.

VirtualServer Operations

To start a virtual server

If a VirtualServer (VS) is stopped an associated with a running node, a VirtualServer may be started. If VS Tools is disabled, the virtual server will be in Running state after the job completes. However, the virtual server is only powered on.

If VS Tools is enabled, the virtual server will be in Starting state after the job completes. The virtual server will go to Running state after the guest boots and VIMS receives the first successful statistic inquiry.

```
<job.begin...>
  vs.start();
<job.commit...>
```

Example

```
import com.virtualiron.vce.mgmt.api.manager.ConfigurationManager;
import com.virtualiron.vce.mgmt.api.virtual.VirtualServer;
...

  ConfigurationManager cm = ...
  VirtualServer vs = (VirtualServer) cm.findObject(VirtualServer.class, "...");

  <job.begin...>
    // start VirtualServer operation
    vs.start();
  <job.commit...>

  // if Accelerated Drivers are enabled, VirtualServer will be in Starting state
  //   after vs.start() job is completed
  if (vs.getVsToolsFlag())
  {
    // listen for VirtualServerRunningEvent
  }
  else
  {
    // if Accelerated Drivers are disabled, VirtualServer will be in Running state
  }
```

To shutdown a virtual server:

If a virtual server is running with VS Tools enabled and is not in error, it may be issued a *shut down* or *power off* command to gracefully shut down.

```
<job.begin...>
  vs.shutdown();
<job.commit...>
```

To stop a virtual server:

If a virtual server has VS Tools disabled or the virtual server is in error, it may be stopped immediately. The guest is not shut down gracefully.

```
<job.begin...>
  vs.stop();
<job.commit...>
```

Example

If VSTools is disabled, hard-reset the VirtualServer. If the virtual server is in error, hard-reset the virtual server.

```
import com.virtualiron.vce.mgmt.api.manager.ConfigurationManager;
import com.virtualiron.vce.mgmt.api.virtual.VirtualServer;
...
   ConfigurationManager cm = ...
   VirtualServer vs = (VirtualServer) cm.findObject(VirtualServer.class, "VirtualServer Name");

   <job.begin...>
      // if VS is Running with Accelerated Drivers, shutdown gracefully
      if (vs.getVsToolsFlag())
      {
         vs.shutdown();
      }
      else
      {
         // otherwise, do a hard reset
         vs.stop();
      }
   <job.commit...>

  // if Accelerated Drivers are enabled, VirtualServer will be in Stopping state
  //   and still shutting down when job completes
   if (vs.getVsToolsFlag())
   {
      // wait for VS to shutdown, listen for VirtualServerStoppedEvent
   }
   else
   {
      // VS is Stopped
   }
```

To restart running virtual with VS Tools enabled:

If a virtual server is running with VS Tools enabled and is not in error, it may be restarted. Restart does a graceful shutdown and then Start.

```
<job.begin...>
   vs.restart();
<job.commit...>
```

To move a stopped virtual server to a new node:

The node must have adequate resources—network and storage access. Otherwise, the move will fail.

```
Node node =
  <job.begin...>
    node.addVirtualServer(vs);
  <job.commit...>
```

To move a running virtual server with VS Tools enabled to a new node:

```
Node node =
  <job.begin...>
    vs.migrate(node);
  <job.commit...>
```

Snapshots and Overbooking

When you do a snapshot of your logical disks, the system optimizes the space needed to maintain the snapshot data. However, the system also maintains expansion space for the snapshot to grow. Use the Overbooking value to specify the amount of space to use, and to allow the system to go over, or overbook, the specified expansion space.

Here's an example: You have created a 100 GB disk group and then created the following disks for your virtual servers to use:

Logical Disk Size	Snapshot Size	Snapshot Expansion Size
20 GB	2 GB	18 GB
30 GB	3 GB	27 GB

At this point, your overbooking is **Enabled** and set to **100%**, the defaults for all disk groups. You can not create any more logical disks since you are currently using the 100 GB space: 20 + 30+ 2 + 3 +18 + 27 GB.

Assuming that your snapshot disks will most likely not grow to consume their entire expansion size, you can adjust the overbooking value to take advantage of this. By changing the Overbook Limit to 150%, you can create another logical disk as shown:

The 150% Overbook Limit allocates an additional 50 GB: 25 + 2.5 + 22.5 GB.

To get a snapshot a VS:

You can get a snapshot of stopped or running virtual server with a LogicalDisk, BootDevice, or StorageDevice.

```
<job.begin...>
   VirtualServer vs = vs.createSnapshot();
<job.commit...>
```

Example

Snapshot a VirtualServer creates a cloned VirtualServer with snapshots of all its logical disks. Overbooking limits may be specified when you take the snapshot.

```
import com.virtualiron.vce.mgmt.api.manager.ConfigurationManager;
import com.virtualiron.vce.mgmt.api.virtual.VirtualServer;
...
   ConfigurationManager cm = ...
   VirtualServer vs = (VirtualServer) cm.findObject(VirtualServer.class, "...");

   <job.begin...>
      // clone VS and snapshot its logicaldisks, use default values
      snapvs = (VirtualServer) vs.createSnapshot();
   <job.commit...>

   <job.begin...>
      // same snapshot but specify overbooking parameters for snapshot disks
      int sizePercent      = 10;    // initial size of snapshot as percent of parent disk
      int thresholdPercent = 50;    // when snapshot utilizes this percent, snapshot is resized
      int targetPercent    = 30;    // when resizing snapshot, target utilization is less than this
      VirtualServer snapvs = vs.createSnapshot(sizePercent, thresholdPercent, targetPercent);
   <job.commit...>
```

```
// to get snapshot parent
VirtualServer parentvs = snapvs.getSnapshotParent();

// to get snapshot children
ArrayList <VirtualServer> childrenvs = snapvs.getSnapshotChildren();

<job.begin...>
   // to delete a snapshot
   parentvs.deleteSnapshot(snapvs);
<job.commit...>

import com.virtualiron.vce.mgmt.api.physical.LvmGroup;
import com.virtualiron.vce.mgmt.api.physical.LogicalDisk;

// assuming VS storage device is a LogicalDisk
LogicalDisk snapdisk = (LogicalDisk) snapvs.getStorageDevices().get(0);
LvmGroup diskgroup = snapdisk.getParentLvmDiskGroup();
```

To disable overbooking:

```
<job.begin...>
  // if overbooking is enabled on a diskgroup, disable it
  if (diskgroup.getEnforceMaxOverbookingPercent())
  {
     diskgroup.setEnforceMaxOverbookingPercent(0);
  }
<job.commit...>
```

To change overbooking percentage:

```
<job.begin...>
  // overbooking percent may be set to higher values
  //   overbooking of 100% or less doesn't allow overbooking
  if (diskgroup.getMaxOverbookingPercent() < 120)
  {
    diskgroup.setMaxOverbookingPercent(150);
  }
<job.commit...>
```

9

STATISTICS AND PERFORMANCE

The VIMS collects performance statistics every 20 seconds from every node and virtual server and puts the data in a TimeSeries. Each TimeSeries is a circular queue that contains the last 90 minutes of samples taken at 20-second intervals, or 270 samples. If there is no data to collect, the VIMS still makes an entry but marks it as invalid.

Time Series

90:00	89:40	...	1:20	1:00	:40s	:20s	latest

Note that all statistics data is collected from the nodes except for virtual server processor and memory data, which are still collected from the virtual server via VS Tools.

NodeData

Each node has a TimeSeries associated with it that contains a NodeData entry. The NodeData has node-specific information plus links to Process, Memory, Network, and Disk data.

Method	Description
long getLocalTime()	Get node local time
List getVirtualServers()	Return ArrayList of node's VirtualServers
ProcessorData getProcessorData()	
MemoryData getMemoryData()	
NetworkData getNetworkData()	
FibreChannelData getFibreChannelData()	
DiskThroughputData getDiskThroughputData()	

VirtualServerData

Each virtual server has a TimeSeries that contains a Virtual Server Data entry. The Virtual Server data has VS-specific information plus links to Processor, Memory, Network, and Disk data.

- If a VS has VSTools enabled, Processor, Memory, Network, and Disk data are collected.
- If a VS does not have VS Tools enabled, only Network and Disk data are collected.
- If a VS has never been assigned to a node, the TimeSeries is empty.
- If a VS is migrated between nodes, there are a few invalid entries until a baseline is created on the new node.

Method	Description
String getArch()	get guest processor architecture (x86, x86_64)
String getDriver()	get VSTools
String getDriverBuildNumber()	get VSTools version
String getDriverBuildNumber()	get VSTools version
String getKernelRelease()	get guest OS kernel (Linux 2.6.9)
String getOperatingSystem()	get guest OS type (RHAS4, Win2k3)
String getRpm()	get VSTools Linux Rpm (if available)
ManagedObject getNode()	return node VS is running on
ProcessorData getProcessorData()	
MemoryData getMemoryData()	
NetworkData getNetworkData()	
VirtualDiskData getVirtualDiskData()	

Processor Data

ProcessorData is the processor utilization data collected for both nodes and virtual servers.

Method	Description
boolean isValid()	Is data valid
long getTimestamp()	
int getClockRate()	number of clock ticks per second (100)
double[] getDomainLoadAverage()	arraylist of domain 1, 5, 15 minute load averages
double getDomainLoadAverage(int)	1, 5, 15 minute load averages for specific domain
double getDomainIdlePercentTotal()	total percentage of Idle time (VirtualServer only)
double getDomainNicePercentTotal()	total percentage of Nice time (VirtualServer only)
double getDomainSystemPercentTotal()	total percentage of System time (VirtualServer only)
double getDomainUserPercentTotal()	total percentage of User time (VirtualServer only)
double getDomainZeroPercent()	percentage of domain 0 time
int getNumberOfProcessors()	number of processors
double getProcessorSpeed()	processor speed in GHz
double getTickFraction()	fraction of total clock ticks in time interval given to domain
double getTotalProcessorCapacity()	node's total processor capacity
double getUsedProcessorCapacity()	processor capacity used by a domain
double getAvailableProcessorCapacity()	node's available processor capacity (Node only)
double getVirtualServerPercent()	

Processor Utilization

To compute processor utilization (but not per processor utilization) from a node's point of view:

```
NodeData nd = ...
ProcessorData pd = nd.getProcessorData();
// Percent of total processor capacity used by domain zero (i.e. overhead)
double domZeroPct = pd.getDomainZeroPercent();
// Percent of processor capacity used by ALL virtual servers on the node
double vsPct = pd.getVirtualServerPercent();
```

To compute percentage of a node's total processor capacity that a particular virtual server is using.

```
VirtualServerData vsd = ...
ProcessorData pd = vsd.getProcessorData();
double vsPct = pd.getTickFraction() * 100.0;
```

To get a virtual server's point of view:

```
VirtualServerData vsd = ...
ProcessorData pd = vsd.getProcessorData();
double user = pd.getDomainUserPercentTotal();
double system = pd.getDomainSystemPercentTotal();
double nice = pd.getDomainNicePercentTotal();
double vsPct = user + system + nice;
```

To get the processor capacity used by domain zero:

```
NodeData nd = ...
ProcessorData pd = nd.getProcessorData();
double usedCapacity = pd.getUsedProcessorCapacity();
```

To compute used processor capacity for the entire node:

```
NodeData nd = ...
ProcessorData pd = nd.getProcessorData();
double usedCapacity = pd.getTotalProcessorCapacity() - pd.getAvailableProcessorCapacity();
NodeData nd = ...
ProcessorData pd = nd.getProcessorData():
double usedCapacity = pd.etTotalProcessorCapacity() - pd.getAvailableProcessorCapacity();
```

Memory Data

Memory data collected for nodes and virtual servers.

Method	Description
boolean isValid()	If true, data is valid
long getTimestamp()	Return timestamp for data
long getFree()	Return free memory in bytes
long getTotal()	Return total memory in bytes
long getUsed()	Return used memory in bytes
long getTotalSwap()	Return total swap in bytes
long getFreeSwap()	Return free swap in bytes
long getAvailableInMB()	
long getShared()	

Network Data

Network data collected for nodes and virtual servers.

Method	Description
boolean isValid()	If true, data is valid
long getTimestamp()	Timestamp
String getDeviceId(String)	deviceID (e.g., eth0, vif1.0)
int getDomainId(String)	Get Xen domain id of virtualserver for this interface
ArrayListgetInterfaceNames()	Array list of all network interfaces
BigInteger getRxBytes(String)	Received bytes for interface name
BigInteger getRxPackets(String)	Received packets for interface name
BigInteger getRxTotalBytes()	Total received bytes for virtual server
BigInteger getRxThroughput()	Number of KiB received since last sample
BigInteger getTxBytes(String)	Transmitted bytes for interface name
BigInteger getTxPackets(String)	Transmitted packets for interface name
BigInteger getTxTotalBytes()	Total transmitted bytes for virtual server
BigInteger getTxThroughput()	Number of KiB transmitted since last sample
BigInteger getThroughput()	Number of KiB received and transmitted since last sample
ArrayList getIpAddresses()	Array list of interface IP addresses
BigInteger getIpAddress(String)	IP address of interface name

VirtualDiskData

Virtual disk data collected for virtual servers only.

Method	Description
boolean isValid()	Is data valid?
long getTimestamp()	Timestamp
HashSet getDeviceIDs()	Array list of device IDs
BigInteger getReadSectors(String deviceId)	Number of 512-byte sectors read for device ID
BigInteger getWriteSectors(String deviceId)	Number of 512-byte sectors written for device ID
long VirtualDiskData.getIscsiThroughput()	
BigInteger VirtualDiskData.getIscsiTotalSectors()	
BigInteger getTxTotalSectors()	Total number of 512-byte sectors written
BigInteger getRxTotalSectors()	Total number of sectors read
ArrayList getVirtualDisks()	Array list of virtual server virtual disks
BigInteger getReadBytes(String)	Number of bytes read for device ID
BigInteger getWriteBytes(String)	Number of bytes written for device ID
BigInteger getTxTotalBytes()	Total number of bytes written
BigInteger getRxTotalBytes()	Total number of bytes read

DiskThroughputData

DiskThroughputData is collected for nodes only, and includes local disk and Fibre Channel statistics only. iSCSI statistics are included in the node NetworkData. The throughput calculation uses two successive disk statistics samples.

Method	Description
boolean isValid ()	If true, data is valid
long getTimestamp ()	Get timestamp
boolean isThroughputValid ()	If true, throughput is valid
BigInteger getThroughput ()	Get total read and write disk throughput

Examples: Get Available Node Capacity

Example: Get Latest Virtual Server Load Average

```
long lastTimestamp = 0;
ConfigurationManager cm = VirtualizationManager.getConfigurationManager();
// find the latest VS time series
VirtualServer vs = (VirtualServer) cm.findObject(VirtualServer.class, "...");
TimeSeries ts = vs.getStatisticEvent().getLinkedTimeSeries().getTimeSeries(0);
// Save its handle somewhere...
ManagedObjectHandle moh = vs.getHandle();
...
// No more often than every 20 seconds
// First check that the virtual server still exists
// Next line assumes that a static connection exists...
VirtualServer vs = (VirtualServer) cm.findObject(moh);
if (null == vs)
{
   // vs was deleted
}
else
{
   VirtualServerData vsd = (VirtualServerData) ts.tail();
   long timestamp = vsd.getTimeStamp();
   if (timestamp > lastTimeStamp)
   {
      lastTimeStamp = timestamp;
      ProcessorData pd = vsd.getProcessorData();
      if (pd.isValid())
      {
         double[] loadAvg = pd.getDomainLoadAverage();
         ...
      }
      else
      {
         // Processor data is not valid
      }
   }
   else
   {
      // No newer data
      // vs could be stopped or migrating
   }
}
```

Example: Get Virtual Server IP Addresses:

```
ConfigurationManager cm = ...
VirtualServer vs = (VirtualServer) cm.findObject(VirtualServer.class, "...");
// find most recent time series
TimeSeries ts = vs.getStatisticEvent().getLinkedTimeSeries().getTimeSeries(0);
VirtualServerData vsd = (VirtualServerData) ts.tail();
NetworkData networkData = vsd.getNetworkData()
if (nd.isValid())
{
    ipAddress[] = networkData.getIpAddresses();
}
else
{
    // latest network data isn't valid
}
```

10

System Events

The Event subsystem provides a flexible method to query the status of any managed object. A VirtualServer may be queried as to its power state (Running, Stopping, Stopped) as well as its error state.

Event Object Model Hierarchy

The Event object model is a hierarchy of event types that make use of multiple inheritance.

Example

VirtualServerRunningEvent extends InformationalEvent, VirtualServerStatusEvent, RunningEvent

RunningEvent extends UpEvent, PowerEvent

VirtualServerStatusEvent extends StatusEvent, VirtualServerEvent

VirtualServerErrorEvent extends ErrorEvent, VirtualServerStatusEvent

To query an object's status:

Method	Description
StatusEvent ManagedObject.getStatusEvent()	Return latest status event
StatusEvent ManagedObject.getStatusEvent(Class, Severity)	Return latest status event of Class with severity or lower.
StatusEvent ManagedObject.getStatusEventIgnoring(Class, Severity)	Return latest status event of Class with severity or higher
StatusEvent ManagedObject.getStatusEventTriage()	Return most recent critical event for managed object and its children

Event Severity

Each event is assigned a severity by default. ErrorEvents are assigned CRITICAL, WarningEvents are MAJOR and InformationalEvents are INFORMATIONAL. The user or the system may acknowledge critical or warning events, which reduces their severity.

Severity	Color	Description
StatusEvent.Severity.UNKNOWN	Red	Requires corrective action, such as rebooting a node or shutting down the virtual server.
StatusEvent.Severity.CRITICAL	Red	
StatusEvent.Severity.MAJOR	Yellow	An alert warning of a potential problem that may require corrective action.
StatusEvent.Severity.MINOR	Yellow	
StatusEvent.Severity.NORMAL	Clear	Informational messages.
StatusEvent.Severity.INFORMATIONAL	Clear	

Example

Given the following sequence of status events for a VirtualServer:

Time Stamp	Event
01:00:00	VirtualServerCreateEvent
01:01:00	VirtualServerStoppedEvent
01:02:00	VirtualServerStartingEvent
01:03:00	VirtualServerErrorEvent (Critical)
01:04:00	VirtualServerRunningEvent (Info)

For each method call below, you get the following results:

Method	Returns
VirtualServer.getStatusEvent()	VirtualServerRunningEvent (INFO)
VirtualServer.getStatusEvent(PowerEvent, INFO)	VirtualServerRunningEvent (INFO)
VirtualServer.getStatusEventIgnoring(ErrorEvent, MINOR)	VirtualServerErrorEvent (CRITICAL)
VirtualServer.getStatusEventTriage()	VirtualServerErrorEvent (CRITICAL)

Node Status Events

NodeRunningEvent extends NodeStatusEvent, RunningEvent, InformationalEvent.
NodeErrorEvent extends NodeStatusEvent, ErrorEvent.
NodeOfflineEvent extends NodeErrorEvent, OfflineEvent.

Event	Type	Description
NodeRunningEvent	InfoEvent	Node is connected and has successfully completed discovery.
NodeStoppingEvent	InfoEvent	Node is powering down.
NodeStoppedEvent	InfoEvent	Node is powered down.
NodeOfflineEvent	ErrorEvent	Node is powered down for more than 1 minute.
NodeVersionMismatchErrorEvent	ErrorEvent	Node is running older version.
NodeVersionLvmRefreshWarningEvent	WarningEvent	Node Lvm refresh failed.

VirtualServer Status Events

Event	Description
VirtualServerStoppedEvent	VS is powered off.
VirtualServerStartingEvent	VS is powering on.
VirtualServerRunningEvent	vstools disabled: VS is powered on vstools enabled: VS is powered on and management server has received VS statistics.
VirtualServerStoppingEvent	VS is powering off.
VirtualServerErrorEvent	VS is in error.
VirtualServerMigrationErrorEvent	VS migration failed.
VirtualServerOfflineEvent	When node goes offline, so do all its VSs.

Examples

```
// check if VS is Running (may still be in Error)
if (vs.getStatusEvent(PowerEvent.class, StatusEvent.Severity.INFORMATIONAL) instanceof
VirtualServerRunningEvent)
{
   // VS is running, do something
}
// check if VS is in Error
if (vs.getStatusEventIgnoring(ErrorEvent.class, StatusEvent.Severity.MINOR) instanceof
VirtualServerErrorEvent)
{
   // VS is in error, do something
}
```