# VACM

## Users and Programmers Manual

**San Mehat**

**Zac Sprackett**

**Dean Johnson**

**Jerry Katzung**

**Carsten Haitzler**

**VACM: Users and Programmers Manual**

by San Mehat, Zac Sprackett, Dean Johnson, Jerry Katzung, and Carsten Haitzler

First Edition

# Table of Contents

# List of Figures

# Preface

This document describes how to install, use, and customize `VACM`. As it is a live document, and always under construction, any constructive criticism is welcomed by the authors.

# Chapter 1. Introduction

VACM (pronounced Va-cuum) is an integrated suite of scalable remote management and monitoring tools for administering clusters or large install bases of servers. The primary authors of VACM are:

- San Mehat <nettwerk@valinux.com>
- Carsten Haitzler <raster@valinux.com>
- Dean Johnson <dtj@sgi.com>
- Jerry Katzung <katzung@valinux.com>
- Zac Sprackett <zacs@valinux.com>

VACM's back end is modular in architecture, allowing it to utilize a virtually endless list of different management and monitoring technologies. The front end is also modular, allowing it to be integrated into a variety of environments without impacting the environment's management and monitoring policies. While VACM was originally intended for use with Linux based scientific clusters, it has been shown that the VACM infrastructure is also suited for ISPs, ISVs, and other large server installations.

If you are interested in participating in the VACM project, either as a feedback provider, backend module writer, or a client writer, please subscribe to the VACM mailing lists, linked off of the *"VACM project page"* (http://sourceforge.net/projects/vacm)

# Chapter 2. Quickstart Guide

## 2.1. About the Quickstart Guide

This chapter is intended to help you get VACM installed and operational on your system. It is not intended to cover everything and you are encouraged to read the rest of the documentation. This section aims to be a step by step guide to get you on your way.

Users with RPM based distributions will find the section "Installing VACM with RPM" invaluable for getting the binaries installed on their systems. Users of other distributions will want to read the section "Installing VACM from source".

## 2.2. Installing VACM with RPM

VACM is available in both source and binary RPM format. The latest release RPMS are always available from the *filelist* (http://sourceforge.net/project/showfiles.php?group_id=19) page on *SourceForge* (http://sourceforge.net).

The VACM installation has been split into multiple binary RPM's. The following binary RPM's are available:

`vacm`

The main VACM distribution including the Nexxus daemon and all the VACM modules. This RPM need only be installed on the controlling node of the cluster.

`vacm-devel`

Header files for developing VACM modules. This RPM is only required to develop VACM modules.

`vacm-node`

Package to be installed on VACM managed nodes. It contains daemons for use with various VACM modules for additional monitoring functionality.

`vacm-flim`

Flim is a Graphical User Interface (GUI) Client for VACM which provides easy interaction with the nexxus through several plugin modules. Flim requires vacm-clientlib.

`vacm-hoover`

Hoover is a Graphical User Interface (GUI) Client for VACM. Hoover requires vacm-clientlib.

`vacm-vash`

VACM command line client for scripting and low level command-line access. Vash need only be installed on client machines requires vacm-clientlib.

`vacm-sercon`

Command-line serial console terminal program to remotely access consoles of nodes on a VACM cluster.

`vacm-clientlib`

VACM client library required to run VACM clients. This RPM must be installed on any machine which will run VACM clients.

`vacm-doc`

VACM documentation. This package should contain SGML, postscript and HTML versions of the docs. It has no dependancies and can be installed as appropriate.

The binary RPM’s are built against a VA Enhanced RedHat system. They should also work with a stock RedHat installation as well. Users of other RPM based distributions will want to compile the source RPM to avoid dependancy issues.

On the nodes to be managed by VACM, install the following RPM:

```
rpm -Uvh vacm-node-2.0.0-1.i386.rpm
```

Finally, the following RPMs need to be installed on any machines which will function as clients.

```
rpm -Uvh vacm-clientlib-2.0.0-1.i386.rpm vacm-vash-2.0.0-1.i386.rpm \
  vacm-sercon-2.0.0-1.i386.rpm
```

Users who wish to utilize a GUI for their management should also install the following:

```
rpm -Uvh vacm-hoover-2.0.0-1.i386.rpm rpm -Uvh vacm-flim-2.0.0-1.i386.rpm
```

## 2.3. Installing VACM from source

Step one is to get the VACM source distribution. The most current release can always be found on the *filelist* (http://sourceforge.net/project/showfiles.php?group_id=19) page on *SourceForge* (http://sourceforge.net). The source distribution is the file with the suffix of .tar.gz.

The first step is to untar the archive. For our example we'll use a version 2.0.0 tarball.

```
[zacs@denial]$ tar -xvzf vacm-2.0.0.tar.gz
```

Next, change into the VACM distribution directory.

```
[zacs@denial]$ cd vacm
```

It is now time to configure the distribution. Users with openssl installed on their systems will want to enable the use of SSL encryption in VACM. These users should add the flag --enable-ssl to the command listed below.

```
[zacs@denial]$ ./autogen.sh --prefix=/usr
```

The next step is to actually build the vacm distribution. This step will probably take a few minutes.

```
[zacs@denial]$ make
```

Once the distribution is built, it is time to install. Installation of vacm requires root priviledges. From this point on you will need to be root.

```
[root@denial]# make install
```

Most users will want VACM to start automatically at system bootup. This can be accomplished by adding the following to the end of the rc.local file. This file can usually be found in /etc/rc.d/rc.local

```
/usr/bin/nexxus &
```

If this is a new installation, and not just an upgrade from an older version of VACM, it is necessary to create a configuration file. For security reasons, it is important for this file to be readable only by root.

```
[root@denial]# touch /usr/lib/vacm/vacm_configuration
[root@denial]# chmod 600 /usr/lib/vacm/vacm_comfiguration
```

At this point, VACM has been installed, it is now time to start the nexxus daemon.

```
[root@denial]# /usr/bin/nexxus &
[VACM] 2.0.0 Nexxus daemon (Build Sep 18 2000 21:55:05)
```

## 2.4. Basic Nexxus Configuration

We are now ready to use vash, the command line interface to VACM, to connect to our Nexxus. By default Nexxus creates an administrative account with username blum and password frub. It is generally a good idea to change these defaults right away.

```
[zacs@denial]$ vash -c localhost -u blum -p frub
NEXXUS_READY
```

To rename the administrative account, send the following ipc command. For the purpose of this example, the blum account will be renamed to zsprackett.

```
vash$ ipc localhost nexxus:admin_rename:blum:zsprackett
NEXXUS:8:JOB_STARTED
NEXXUS:8:JOB_COMPLETED
```

Changing the password is accomplished as follows. The default password will be changed to ‘p4ssw0rd‘.

```
vash$ ipc localhost nexxus:admin_chg_password:frub:p4ssw0rd
NEXXUS:10:JOB_STARTED
NEXXUS:10:JOB_COMPLETED
```

Next create a group named webservers. All managed nodes must be members of a group.

```
vash$ ipc localhost nexxus:group_add:webservers
NEXXUS:12:JOB_STARTED
NEXXUS:12:JOB_COMPLETED
```

It is important to add the administrative user to the newly created group. VACM users can only manage nodes in groups to which they belong.

```
vash$ ipc localhost nexxus:group_add_admin:webservers:zsprackett
NEXXUS:14:JOB_STARTED
NEXXUS:14:JOB_COMPLETED
```

At this point it is safe to tell VACM about the nodes to be managed. In this example the managed node will be named *www*. Our example adds the newly created node to the group webservers. It is also worth mentioning that the name assigned to a managed node under VACM does not have to correspond to the nodes DNS hostname.

```
vash$ ipc localhost nexxus:node_add:www:webservers
NEXXUS:16:JOB_STARTED
NEXXUS:16:JOB_COMPLETED
```

Certain modules rely on global variables. The most important of these seems to be the *IP_ADDRESS* variable. The IP_ADDRESS variable can be set as follows.

```
vash$ ipc localhost nexxus:set_var:www:ip_address:192.168.1.1
NEXXUS:17:JOB_STARTED
NEXXUS:17:JOB_COMPLETED
```

# 2.5. Quick and Dirty Module Configuration

This section is a quick and dirty guide to configuring modules for basic operation. It is potentially useful during VACM installation. Please refer to the real module documentation for further information on configuring and testing these mdoules.

## 2.5.1. Configuring EMP

To configure node www for management via EMP, you must know the address of the device to which the EMP port for this node is physically connected. This can be a serial port, like /dev/ttyS0, or a network address and port of the form 192.168.1.1-911. The following example assumes a serial port at ttyS1. In a typical setup, such as this example, the password should be set to *NONE*. The password must correspond with the EMP access password in the Bios.

A proper configuration command such as the following will result in this output.

```
vash$ ipc localhost emp:configuration:www:/dev/ttyS1:NONE
EMP:18:JOB_STARTED
EMP:18:STATUS:ENGAGING
EMP:18:STATUS:PROTOCOL_DETECTED
EMP:18:STATUS:CONNECTION_ACCEPTED
EMP:18:STATUS:DOWNLOADING_FRU
EMP:18:STATUS:DOWNLOADING_SDR
EMP:18:STATUS:DOWNLOADING_SEL
EMP:18:JOB_COMPLETED
```

The following IPC responses indicate an authentication failure. Make sure that the password entered is consistent with the EMP password settings in the Bios. A new configuration command can be sent to update an incorrect setting.

```
EMP:19:JOB_STARTED
EMP:19:STATUS:ENGAGING
EMP:19:STATUS:PROTOCOL_DETECTED
EMP:19:STATUS:CONNECTION_DENIED
EMP:19:JOB_COMPLETED
```

Output such as the following indicates that Nexxus is trying to configure a port which is not speaking the EMP protocol. Check the cabling and ensure you are attempting to engage on the correct port and that the port is set up for EMP in the Bios.

```
EMP:20:JOB_STARTED
EMP:20:STATUS:ENGAGING
EMP:20:STATUS:PROTOCOL_UNAVAILABLE
EMP:20:JOB_COMPLETED
```

## 2.5.2. Configuring VA1000

VA 1000 nodes are clustered by joining the nodes with a dedicated management bus. The nodes determine their addresses dynamically, so a special start-up procedure is required the first time you power up the cluster. Appendix C explains the procedure required to configure and power up a cluster for the first time.

To configure VA 1000 nodes under VACM, you must know the address of each node on the cluster management bus (CMBus). These addresses are determined automatically when power is applied to each node for the first time. To determine the address of a node, use the following procedure:

- Press and hold the Power switch down for at least 5 seconds.
- Release the Power switch.

- The two rightmost LED's will flash once, indicating the start of the CMBus address transmission.
- The three rightmost LED's will flash individually to indicate the CMBus address. From left to right, the LED's represent 100's, 10's, and 1's of the address.
- Finally, the two rightmost LED's blink together once more to indicate the end of the address transmission.

A proper configuration command results in the following output:

```
vash$ ipc localhost va1000:configuration:www:26
  VA1000:3:JOB_STARTED
  VA1000:3:JOB_COMPLETED
```

## 2.5.3. Configuring RSH

Please be advised that the RSH protocol is not very secure. Using this module on a publicly accessible network such as the internet is not recommended. That being said, RSH can be very useful. Configuring a node for rsh is a two step process. To configure the RSH module for node www send the following command.

```
vash$ ipc localhost rsh:configuration:www:rsh:root
RSH:21:JOB_STARTED
RSH:21:JOB_COMPLETED
```

On the node itself, edit the file /root/.rhosts and add this line.

```
hostname.of.nexxus root
```

Set the permissions on the file to 600.

```
chmod 600 /root/.rhosts
```

## 2.5.4. Configuring SERCON

To configure a nexxus for serial console access to www, send the following ipc message. This example assumes the console of node www is attached to /dev/ttyS0 and a baud rate of 19200bps.

```
vash$ ipc localhost sercon:configuration:www:/dev/ttyS0:19200
SERCON:22:JOB_STARTED
SERCON:22:JOB_COMPLETED
```

By default, console access will only be available during the post boot sequence. Further configuration is required to support full serial console access. To redirect OS console messages on the remote node through a serial console:

```
[root@www]# cd /dev
[root@www]# rm -f console tty0
[root@www]# mknod -m 622 console c 5 1
[root@www]# mknod -m 622 tty0 c 4 0
```

In the globals section of /etc/lilo.conf add the following:

```
serial=0,19200n8
```

In the kernel section of /etc/lilo.conf add the following:

```
append="console=ttyS0,19200"
```

Run lilo to activate the new settings.

```
[root@www]# /sbin/lilo
```

To enable a getty for the serial port add the following to /etc/inittab:

```
s0:12345:respawn:/sbin/mingetty ttyS0 DT19200 vt100
```

Remove /etc/ioctl.save:

```
bash# rm -f /etc/ioctl.save
```

Add the following line to /etc/securetty to permit root logins over the serial port:

```
ttyS0
```

## 2.5.5. Configuring SYSSTAT

*vacm_sys_statd* must be installed and started on each node to be monitored. Upon starting vacm_sys_statd for the first time, you will be prompted to set a password. Remember this password as you will need to supply it when you configure the nexxus portion of sysstat.

```
vash$ ipc localhost sysstat:configuration:www1:p4ssw0rd
SYSSTAT:7:JOB_STARTED
SYSSTAT:7:JOB_COMPLETED
```

Test the sysstat module:

```
vash$ ipc localhost sysstat:WHO:www1
SYSSTAT:8:JOB_STARTED
SYSSTAT:8:WHO:zsprackett:tty1
SYSSTAT:8:WHO:zsprackett:pts/2
SYSSTAT:8:JOB_COMPLETED
```

## 2.5.6. Configuring BAYTECH

To configure a baytech power strip, first add a node:

```
vash$ ipc localhost nexxus:node_add:bt:webservers
NEXXUS:16:JOB_STARTED
NEXXUS:16:JOB_COMPLETED
```

Next, configure the BAYTECH module for the new node:

```
vash$ ipc localhost baytech:configuration:bt:/dev/ttyS3
NEXXUS:17:JOB_STARTED
NEXXUS:17:JOB_COMPLETED
```

## 2.5.7. Configuring VASENET

To configure a VA100, first add a node:

```
vash$ ipc localhost nexxus:node_add:va100:webservers
NEXXUS:16:JOB_STARTED
NEXXUS:16:JOB_COMPLETED
```

Configure the nodes IP address:

```
vash$ ipc localhost nexxus:set_var:va100:ip_address:192.168.1.2
NEXXUS:16:JOB_STARTED
NEXXUS:16:JOB_COMPLETED
```

Next, configure the VASENET module for the new node:

```
vash$ ipc localhost vasenet:configuration:va100:master:abcdefgh
NEXXUS:17:JOB_STARTED
NEXXUS:17:JOB_COMPLETED
```

Nodes must now be created for each nanoprobe based system:

```
vash$ ipc localhost nexxus:node_add:nano1:webservers
NEXXUS:16:JOB_STARTED
NEXXUS:16:JOB_COMPLETED
```

To configure the nanoprobe based systems, you need to know the name of the VA100 controller as well as the nodes vasenet address:

```
vash$ ipc localhost vasenet:configuration:nano1:va100:010203
NEXXUS:17:JOB_STARTED
```

```
NEXXUS:17:JOB_COMPLETED
```

## 2.5.8. Configuring SBT2

To configure an sbt2 based system, first add a node:

```
vash$ ipc localhost nexxus:node_add:tupelo:webservers
NEXXUS:16:JOB_STARTED
NEXXUS:16:JOB_COMPLETED
```

Next, configure the SBT2 module for the new node:

```
vash$ ipc localhost sbt2:configuration:tupelo:/dev/ttyS4
NEXXUS:17:JOB_STARTED
NEXXUS:17:JOB_COMPLETED
```

## 2.5.9. Configuring Quanta

To configure a quanta based system, first add a node:

```
vash$ ipc localhost nexxus:node_add:quantabox:webservers
NEXXUS:16:JOB_STARTED
NEXXUS:16:JOB_COMPLETED
```

Next, configure the Quanta module for the new node:

```
vash$ ipc localhost quanta:configuration:quantabox:/dev/ttyS5
NEXXUS:17:JOB_STARTED
NEXXUS:17:JOB_COMPLETED
```

# Chapter 3. Getting Started

This chapter describes in detail the process of building and installing VACM, including various compile time options and runtime options.

## 3.1. Compilation and Installation

Begin by downloading the VACM source and install it on a server which you have determined will serve as your Node Controller. Your Node Controller is the server or servers which run the 'server' portions of VACM (called Nexxus). The latest official distribution is always available at the *VACM project page* (http://sourceforge.net/project/?group_id=19) in the file releases section.

If you wish to work from the latest development code, you may obtain it by going to the *VACM project page* (http://sourceforge.net/project/?group_id=19) and linking through to the CVS Repository.

Once you have extracted the files from the distribution, run the autogen.sh script to test your configuration and build the makefiles. Here is a list of arguments and switches that can be passed into the autogen script to change the build behavior:

`--enable-ssl`

Enables encryption between clients and Nexxus (The Node Controller Server)

`--enable-pam`

Enables PAM support for modules that require user authentication

`--without-x`

Disables the building of the GUI's

`--prefix=<PREFIX_DIR>`

Sets the install directory

`--with-logdir=<LOG_DIR>`

Sets the directory that logs will be placed in

Once the script has completed running, simply type `'make'` and `'make install'` as root. VACM will install itself under `/usr/lib/vacm/` if no other path is specified to `autogen.sh`.

# 3.2. Encryption and Security Considerations for VACM

VACM can optionally utilize the OpenSSL encryption libraries to create secure communications connections in virtually all aspects of operation. The only exceptions are in modules which require direct interfaces to hardware over LAN where SSL protocol version 1 encryption may not be available on the remote hardware. To enable encryption support, you must first ensure that the OpenSSL libraries are installed on your Node Controller. If you are going to use any modules which require agent daemons, you must also ensure the OpenSSL libraries are installed on every remote system you wish to manage and monitor. Once the libraries are installed, pass the `--enable-ssl` commandline option to the `autogen.sh` configuration script, and make sure to verify in the output that OpenSSL was detected. VACM itself has a number of security features which ensure that only authorized administrators may manage or monitor systems. Each administrator must have a valid Nexxus user account. The account can only be logged into from authorized internet addresses, and once authenticated and authorized, the user may only execute commands that have been authorized for the particular user.

In order to utilize an SSL connection, you will need to generate a key and a certificate file on the nexxus machine as well as on any nodes which will be managed using sysstatd or user_admd.

To generate a cerificate and key on the nexxus machine:

```
openssl req -x509 -newkey rsa:1024 -keyout /usr/lib/vacm/vacm.key -
```

```
out \
/usr/lib/vacm/vacm.cert
```

After filling in all the information you will have a valid SSL certificate file and key file available to VACM. The problem now arises that you will have to enter a passphrase every time you wish to start nexxus. This is not always the desirable effect in a cluster environment. You can unwrap the passphrase from the key with the following commands, but you should bear in mind that there are serious security implications in doing this. Please ensure that this is truly necessary in your environment.

```
openssl rsa -in /usr/lib/vacm/vacm.key -out /usr/lib/vacm/vacm.key.unwrapped
mv /usr/lib/vacm/vacm.key /usr/lib/vacm/vacm.key.wrapped
mv /usr/lib/vacm/vacm.key.unwrapped /usr/lib/vacm/vacm.key
```

Similarly, on the node side:

```
openssl req -x509 -newkey rsa:1024 -keyout /etc/vacm.key -out \
/etc/vacm.cert
```

As with the nexxus key file, you may want to unwrap the password on this key. The following should do it.

```
openssl rsa -in /etc/vacm.key -out /etc/vacm.key.unwrapped
mv /etc/vacm.key /etc/vacm.key.wrapped
mv /etc/vacm.key.unwrapped /etc/vacm.key
```

## 3.3. Running Nexxus for the First Time

Test out the VACM installation by typing `'/usr/bin/nexxus -l'`. You should see something like this:

```
VACM 2.0.0 (Beta 2 Candidate) Nexxus daemon (Build Jun 1 2000 07:28:49)
```

```
[Nexxus] Standard Out Logging Enabled
[Nexxus] No groups. Creating 'default' group
[Nexxus] No users. Creating dflt user 'blum' w/ pass 'frub'
[Nexxus] User will be added to group 'default'
[Nexxus] Identified 9 modules
[Nexxus] [Useradm][0.1][USER_ADM][San Mehat (nettwerk@valinux.com)]
[Nexxus] [VA1000][1.1][VA1000][Jerry Katzung (katzung@valinux.com)]
[Nexxus] [ICMP ECHO][1.0][ICMP_ECHO][San Mehat (nettwerk@valinux.com)]
[Nexxus] [msc][2.0][MSC][Dean Johnson (dtj@sgi.com)]
[Nexxus] [rsh][2.0][RSH][Dean Johnson (dtj@sgi.com) & Zac Sprack-
ett (zacs@valinux.com)]
[Nexxus] [Sysstat][0.1][SYSSTAT][San Mehat (nettwerk@valinux.com)]
[Nexxus] [SERCON][2.0][SERCON][San Mehat (nettwerk@valinux.com)]
[Nexxus] [EMP][2.0][EMP][San Mehat (nettwerk@valinux.com)]
[Nexxus] [BayTech][2.0][BAYTECH][Zac Sprackett (zacs@valinux.com)]
```

This indicates that Nexxus has successfully started, created the default user and groups, and loaded in a few VACM modules. You can further test the installation by typing `'vash -c localhost -u blum -p frub -x "ipc localhost nexxus:node_list"'` in a separate shell. You should see the following returned:

```
NEXXUS:2:JOB_STARTED
NEXXUS:2:JOB_COMPLETED
vash$
```

The number '2' itself is unimportant. What is important is that Nexxus has responded. The 'VASH' tool is discussed in 'Using VACM with vash'. If these tests have been successful then congratulations; you are now ready to begin configuring and using VACM. At this point if you have not already subscribed, it is probably a good idea to subscribe to the VACM mailing lists on the *VACM project page* (http://sourceforge.net/project/?group_id=19).

## 3.4. Nexxus Command Line Options

Nexxus accepts the following command line options:

```
-d   -- enable debugging
-b   -- enable continuous configuration backups
-l   -- enable logging to standard out
```

# Chapter 4. VACM Architecture Overview

The following section runs through the various VACM components and architecture.

## 4.1. VACM Components

VACM consists of three main software components; some of which must run constantly, and some of which need only be run when you wish to have VACM perform a given operation. The components are described in detail below.

### 4.1.1. Nexxus (Node Controller)

Nexxus can be considered the "heart" of VACM. It is responsible for receiving user or automated script requests, and dispatching them to the proper low level handler for execution. It is also responsible for maintaining the concept of the "nodelist". Nexxus should be run on a piece of dedicated hardware referred to as the "Node Controller". Depending on the types of monitoring you wish to do, the Node Controller may or may not require special hardware. For example, if you wanted to have a Node Controller monitor and manage a bank of UPS’s that used a serial cable for control, you may have to install serial port extenders into the Node Controller so it has the necessary ports to communicate. This is not to say that a machine that has been designated a Node Controller, must run Nexxus exclusively.

Depending on the number of machines you wish to monitor, along with the types of monitoring you wish to do, giving a Node Controller other processing responsibilities is just fine. One thing to keep in mind however, is that if your Node Controller is one of the nodes you are monitoring, exercise extreme care when managing this node. For example, if your server acts both as a Node Controller *and* a processing node, and decides to power down the entire cluster, the Node Controller itself will be powered down. To avoid ’painting yourself into a corner’, VA recommends allocating a Node Controller as a dedicated node.

### 4.1.2. Modules

Modules are the "workhorses" of VACM. They receive requests from Nexxus and act upon them in a specific way. For example, the `EMP` module takes requests from Nexxus and performs operations specific to the Emergency Management Port on many server motherboards. The `BAYTECH` module takes requests and performs operations related to the Baytech serial port addressable power strips.

While some modules like `EMP` and `VA1000` are responsible for communicating with nodes using specific management communication protocols, other modules provide higher-level services for cluster management. Many of these services are implemented totally in software and allow monitoring and control of the software running on the nodes in the cluster. For example, the `SYSSTAT` module allows users to inspect various details of a node’s OS and hardware configuration.

### 4.1.3. Clients

Clients are the user interface to VACM. They handle requests from the user and submit the appropriate IPC command to the Nexxus. They receive ipc responses from the Nexxus and can display them to the user in one form or another. Clients are available for the command line (such as vash), as well for X11 (such as Hoover and Flim).

### 4.1.4. VACM IPC Messaging

VACM commands are in the form of IPC (Inter Process Communication) message strings. These are ASCII NULL terminated strings that are field separated by colons (:). All messages have the same basic format:

```
MODULE_IPC_TAG:COMMAND:NODE_ID:ARGS:...
```

The MODULE_IPC_TAG instructs which module the message is to be routed to. The COMMAND is the descriptor for the operation which is to be performed. NODE_ID is the target node which the operation is to be performed on (if applicable). In most

applicable cases, a shell style GLOB string can be used to select a number of nodes. ARGS is a colon delimited list of arguments which the command requires to complete its operation. A few example messages are shown:

```
EMP:POWER_OFF:sanbox  -- Instruct the EMP module to
                         power off node 'sanbox'
ICMP_ECHO:PING:sanbox -- Instruct the ICMP_ECHO module to
                         ping node 'sanbox'
NEXXUS:MODULES        -- Instruct Nexxus to return a list
                         of all loaded modules
```

When a message is sent to Nexxus, a `job id` is assigned to the task associated with the message and sent back to the originator of the message. This is done because operations being performed by VACM for a node may run concurrently, and the originator needs to know which return messages are for which operation it has issued. A job id is an unsigned 32 bit integer value, with the value of 0 being reserved for return messages that are not associated with a requested operation. These special return messages are known as unsolicited messages. Here is an example IPC transaction using VASH:

```
[root@lysithea nexxus]# vash  (We execute vash from the
                               command line)
vash$ connect lc              (Instruct vash to
                               connect to Nexxus)
lc login: blum                (Enter in username
                               for Nexxus auth)
Password: ****                (Enter in password
                               for user)
NEXXUS_READY                  (Nexxus informs us it is ready)
vash$ ipc lc nexxus:node_list (Get a list of nodes)
NEXXUS:2:JOB_STARTED          (Job started and job id
                               notification)
NEXXUS:2:NODELIST:box1        (List of nodes)
NEXXUS:2:NODELIST:box2
NEXXUS:2:JOB_COMPLETED        (Job completion notification)
```

VASH is described in detail in 'Using VACM with vash'.

### 4.1.5. Node Global Variables

Sometimes there may be information for a node that pertains to all modules. An IP address for example, is in most cases global for a node, and many modules who wish to communicate with a node over the network, will need to know it. For this reason a node can have what are known as "global variables" associated with it. These are variables that can be read or written by the user, and that are sent to all modules when they are loaded, or when a variable is modified. The process of setting and getting global variables is discussed in ’Getting and Setting Node Global Variables’ in the ’Nexxus Loopback Module’ section.

# Chapter 5. VACM Modules

## 5.1. Nexxus Loop back Module

The Nexxus Loopback Module is actually a module internal to Nexxus. It allows the user to perform internal Nexxus functions. This module is the only module in which a NODE_ID argument is optional in IPC messages.

### 5.1.1. Module Features

Nexxus maintains the system nodelist, user lists, logs, access control lists, and manages command dispatching. The module interface is designed for use by administrators primarily, although depending on your particular topology you may want to allow users to access certain functions.

### 5.1.2. Authenticating to Nexxus

Once a client connection is established to Nexxus, it is necessary for the client to authenticate as a user. Note that the client library does this authentication for you, so it is not necessary to send the authentication message when using clients that have linked with the client library.

```
FORMAT:
NEXXUS:AUTH:<USER_NAME>:<USER_PASSWORD>
RESPONSES:
JOB_STARTED         - Authentication Commencing
JOB_COMPLETED       - Authentication Complete
JOB_ERROR:BAD_AUTH - Authentication Failed
```

## 5.1.3. Identifying VACM version

The version of VACM running can be identified by the following command:

```
FORMAT:
NEXXUS:VERSION
RESPONSES:
JOB_STARTED
VERSION:<STRING>:<IPC_VER> - Version string and
                             module IPC version compliance
JOB_COMPLETED
```

## 5.1.4. Listing Modules Currently Loaded

A list of all modules currently loaded by Nexxus and available for use can be retrieved by the following command:

```
FORMAT:
NEXXUS:NEXXUS_MODULES
RESPONSES:
JOB_STARTED
MODULE:SHORT NAME<:LONG NAME>:<AUTHOR>:<DESCRIPTION>:<VERSION>:<STATE>
JOB_COMPLETED
FIELDS:
<IPC_TAG> - The identifier for the module.
            This tag should be used to
            send messages to the module
<STATE>   - The internal Nexxus state of
            the module. This value should
            not be relied upon for anything.
```

## 5.1.5. Adding Nodes

Nodes are added to Nexxus with the NODE_ADD command. This command takes a unique NODE_ID as it's first argument. The second argument is a GROUP to which the node is to belong. If you don't have a particular group in mind, specify the 'DEFAULT' group. Note that the current user must be a member of the group the node is to be added to. If GROUP does not exist, it will be implicitly created, and the current user added.

```
FORMAT:
NEXXUS:NODE_ADD:<NODE_ID>:<GROUP>
RESPONSES:
JOB_STARTED
JOB_ERROR:NODE_EXISTS - The node_id is not unique to the system.
                        (It may exist in another group)
JOB_ERROR:BAD_GROUP   - The current user is not a member of
                        the group specified
JOB_COMPLETED
```

## 5.1.6. Removing Nodes

Nodes can be removed with the NODE_DEL command:

```
FORMAT:
NEXXUS:NODE_DEL:<NODE_ID>
RESPONSES:
JOB_STARTED
JOB_ERROR:NODE_NOT_FOUND
JOB_COMPLETED
```

## 5.1.7. Renaming Nodes

Nodes can be renamed using the NODE_RENAME command:

*FORMAT:*
NEXXUS:NODE_RENAME:<OLD_NAME>:<NEW_NAME>
*RESPONSES:*
JOB_STARTED
JOB_ERROR:NODE_EXISTS
JOB_ERROR:NODE_NOT_FOUND
JOB_COMPLETED

## 5.1.8. Listing Nodes

A nodelist is obtained with the NODE_LIST command:

*FORMAT:*
NEXXUS:NODE_LIST
*RESPONSES:*
JOB_STARTED
NODELIST:<GROUP>:<NAME>
JOB_COMPLETED

## 5.1.9. Adding Users

Users are added to Nexxus's userlist with the ADMIN_ADD command. The new account is not allowed to connect from anywhere, and cannot execute any IPC commands. To add IPC and access privileges, you must use the ADMIN_ADD_ADDR_ACL and ADMIN_ADD_MOD_ACL commands. The default

ACL policies are set to DENY. To change this, use the ADMIN_CHG_DEFAULT command.

```
FORMAT:
NEXXUS:<ADMIN_ADD:USER_NAME>:<PASSWORD>:<GROUP>
RESPONSES:
JOB_STARTED
JOB_ERROR:USER_EXISTS
JOB_ERROR:PASSWORD_TOO_LONG - Password is > 40 characters
JOB_ERROR:BAD_PASSWORD      - Password contains illegal
                              characters
JOB_ERROR:BAD_GROUP         - Current user is not a member
                              of the group the new user
                              is to belong to
```

## 5.1.10. Removing Users

Users are removed from Nexxus's userlist with the ADMIN_DEL command:

```
FORMAT:
NEXXUS:ADMIN_DEL:<USER_NAME>
RESPONSES:
JOB_STARTED
JOB_ERROR:USER_NOT_FOUND
JOB_COMPLETED
```

## 5.1.11. Renaming Users

Users may be renamed with the ADMIN_RENAME command:

```
FORMAT:
```

```
NEXXUS:ADMIN_RENAME:<OLD_NAME>:<NEW_NAME>
RESPONSES:
JOB_STARTED
JOB_ERROR:USER_NOT_FOUND
JOB_ERROR:USER_EXISTS
JOB_COMPLETED
```

## 5.1.12. Listing Users

A list of users can be obtained from Nexxus with the ADMIN_LIST command:

```
FORMAT:
NEXXUS:ADMIN_LIST
RESPONSES:
JOB_STARTED
ADMIN_LIST:<USER>
JOB_COMPLETED
```

## 5.1.13. Changing a User's Password

The ADMIN_CHG_PASSWORD command is used to change the current user's password. It requires the old as well as the new password. The old password is verified before the password is changed.

```
FORMAT:
NEXXUS:ADMIN_CHG_PASSWORD:<OLD PASSWORD>:<NEW PASSWORD>
RESPONSES:
JOB_STARTED
JOB_ERROR:BAD_AUTH - Specified OLD password was not
                     correct for current user
```

```
JOB_COMPLETED
```

## 5.1.14. Setting the User's Password

The ADMIN_SET_PASSWORD command is used by administrators to set or reset a user's password. This command should be restricted to all but the main administrator as it allows the password of *any* user to be changed *without* any authentication checks.

*FORMAT:*
NEXXUS:ADMIN_SET_PASSWORD:<USER NAME>:<NEW PASSWORD>
*RESPONSES:*
JOB_STARTED
JOB_ERROR:USER_NOT_FOUND
JOB_COMPLETED

## 5.1.15. Listing Groups a User is a Member of

The ADMIN_LIST_GROUPS command is used to return a list of groups that the specified user is a member of.

*FORMAT:*
NEXXUS:ADMIN_LIST_GROUPS:<USER>
*RESPONSES:*
JOB_STARTED
JOB_ERROR:USER_NOT_FOUND
ADMIN_LIST_GROUPS:<GROUP>
JOB_COMPLETED

## 5.1.16. Adding Module IPC Access Control Rules For a User

Module IPC rules are used to control the IPC commands a user is permitted to send. For example, an administrator may want to allow user ’Bob’ to have the ability to send messages to the SYSSTAT module to obtain system status information. But, the administrator may want to restrict Bob so that he can’t send ’POWER_OFF’ messages to the EMP module. A module IPC rule can be added with the ADMIN_ADD_MOD_ACL_RULE command:

```
FORMAT:
NEXXUS:ADMIN_ADD_MOD_ACL_RULE:<USER_NAME>:<POLICY>:<MODULE>:<GLOB>
RESPONSES:
JOB_STARTED
JOB_ERROR:USER_NOT_FOUND
JOB_ERROR:DUPLICATE_RULE
JOB_COMPLETED
FIELDS:
<POLICY> - Can be either ALLOW or DENY
<MODULE> - The module IPC tag this rule is for
<GLOB>   - A standard shell style GLOB wildcard
           pattern that will be applied to the
           command string to determine if this
           rule is to be used.
EXAMPLES:
NEXXUS:ADMIN_ADD_MOD_ACL_RULE:SAN:DENY:EMP:POWER_*
NEXXUS:ADMIN_ADD_MOD_ACL_RULE:SAN:ALLOW:EMP:*
```

## 5.1.17. Removing Module IPC Access Control Rules For a User

An IPC rule can be deleted for a user with the ADMIN_DEL_MOD_ACL_RULE.

Field descriptions are identical to that of ADMIN_ADD_MOD_ACL_RULE.

*FORMAT:*
NEXXUS:ADMIN_DEL_MOD_ACL_RULE:<USER_NAME>:<POLICY>:<MODULE>:<GLOB>
*RESPONSES:*
JOB_STARTED
JOB_ERROR:USER_NOT_FOUND
JOB_ERROR:RULE_NOT_FOUND
JOB_COMPLETED

## 5.1.18. Listing Module IPC Access Control Rules For a User

Obtain a list of module ACL rules with the ADMIN_LIST_MOD_ACL_RULES command:

*FORMAT:*
NEXXUS:ADMIN_LIST_MOD_ACL_RULES:<USER NAME>
*RESPONSES:*
JOB_STARTED
MOD_ACL_RULE:<MODULE>:<POLICY>:<GLOB>
JOB_COMPLETED

## 5.1.19. Adding Address Rules For a User

Address rules allow Nexxus to control the TCP/IP address a user may connect from. The ADMIN_ADDR_ACL_RULE command is used for this purpose. The argument <POLICY> is either ALLOW, or DENY. The argument <ADDR> specifies the network

address pattern for the rule. <SIGBITS> is the number of significant bits in the address to match against.

*FORMAT:*
```
NEXXUS:ADMIN_ADD_ADDR_ACL_RULE:<USER NAME>:<POLICY>:<ADDRESS>:<SIGBITS>
```
*RESPONSES:*
```
JOB_STARTED
JOB_ERROR:USER_NOT_FOUND
JOB_ERROR:DUPLICATE_RULE
JOB_ERROR:BAD_SIGBITS    - sigbits 0 or > 32
JOB_ERROR:BAD_ADDRESS    - Invalid inet address specified
JOB_COMPLETED
```
*EXAMPLES:*
```
NEXXUS:ADMIN_ADD_ADDR_ACL_RULE:san:ALLOW:127.0.0.7:32
NEXXUS:ADMIN_ADD_ADDR_ACL_RULE:san:DENY:10.1.0.0:16
```

## 5.1.20. Removing Address Rules For a User

An address rule can be deleted for a user with the ADMIN_DEL_ADDR_ACL_RULE. Field descriptions are identical to that of ADMIN_ADD_ADDR_ACL_RULE.

*FORMAT:*
```
NEXXUS:ADMIN_DEL_MOD_ADDR_RULE:<USER NAME>:<POLICY>:<ADDRESS>:<SIGBITS>
```
*RESPONSES:*
```
JOB_STARTED
JOB_ERROR:USER_NOT_FOUND
JOB_ERROR:RULE_NOT_FOUND
JOB_ERROR:BAD_SIGBITS    - sigbits  0 or > 32
JOB_ERROR:BAD_ADDR       - Invalid inet address specified
JOB_COMPLETED
```

## 5.1.21. Listing Address Rules For a User

A list of Address rules for a user can be obtained with the ADMIN_LIST_ADDR_ACL_RULES command:

```
FORMAT:
NEXXUS:ADMIN_LIST_ADDR_ACL_RULES:<USER NAME>
RESPONSES:
JOB_STARTED
ADDR_ACL_RULE:<POLICY>:<ADDRESS>:<SIGBITS>
JOB_COMPLETED
```

## 5.1.22. Listing the Default ACL Policies For a User

The default ACL policy for a user is used when there is no specific matching ACL rule found in the user's configuration. There is a separate default policy specified for both module IPC and address access control. The ADMIN_LIST_DEFAULT_ACL_POLICY command returns the current default policy behavior for the acl and user specified. The <ACL> field must be either 'MODULE' or 'ADDRESS'

```
FORMAT:
NEXXUS:ADMIN_LIST_DEFAULT_ACL_POLICY:<USER NAME>:<ACL>
RESPONSES:
JOB_STARTED
JOB_ERROR:USER_NOT_FOUND
JOB_ERROR:BAD_ACL       - ACL was not one
                          of 'ADDRESS' or 'MODULE'
DEFAULT_POLICY:<POLICY> - The default policy.
                          Either 'ALLOW' or 'DENY'
JOB_COMPLETED
```

## 5.1.23. Changing the Default ACL policies For a User

To change the default ACL policy for a user, the ADMIN_CHG_DEFAULT_ACL_POLICY command is used. The specified ACL default policy type is changed for the specified user to the specified policy.

*FORMAT:*
`NEXXUS:ADMIN_CHG:DEFAULT_ACL_POLICY:<USER NAME>:<ACL>:<POLICY>`
*RESPONSES:*
`JOB_STARTED`
`JOB_ERROR:USER_NOT_FOUND`
`JOB_ERROR:BAD_ACL`
`JOB_ERROR:BAD_POLICY`
`JOB_COMPLETED`

## 5.1.24. Adding a Group

In most installations, an administrator arranges nodes in groups. Groups allow nodes to be separated in such a way that only users that belong to the group are allowed to see the nodes or perform operations on them. For example, an installation may have a 100 node webfarm and a 5 node database cluster. The webfarm administrators may not want the database administrators to be able to power down their machines (by accident or deliberately); so the webfarm nodes are added into a separate group to which only the webfarm administrators are members of. A group is created with the GROUP_ADD command.

*FORMAT:*
`NEXXUS:GROUP_ADD:<GROUP NAME>`
*RESPONSES:*
`JOB_STARTED`
`JOB_ERROR:GROUP_EXISTS`
`JOB_COMPLETED`

## 5.1.25. Removing a Group

A group can be removed with the GROUP_DEL command. A group can only be deleted if it contains no nodes or members. The ’DEFAULT’ group cannot be deleted.

*FORMAT:*
NEXXUS:GROUP_DEL:<GROUP NAME>
*RESPONSES:*
JOB_STARTED
JOB_ERROR:GROUP_NOT_FOUND
JOB_ERROR:BAD_GROUP
JOB_ERROR:GROUP_HAS_NODES
JOB_ERROR:GROUP_HAS_MEMBERS
JOB_COMPLETED

## 5.1.26. Renaming a Group

Rename a group with the GROUP_RENAME command:

*FORMAT:*
NEXXUS:GROUP_RENAME:<OLD_NAME>:<NEW_NAME>
*RESPONSES:*
JOB_STARTED
JOB_ERROR:GROUP_NOT_FOUND
JOB_ERROR:GROUP_EXISTS

## 5.1.27. Moving a Node To a Group

A node may be moved into or assigned to a group with the NODE_SET_GROUP command:

*FORMAT:*
NEXXUS:NODE_SET_GROUP:<NODE_ID>:<GROUP>
*RESPONSES:*
JOB_STARTED
JOB_ERROR:GROUP_NOT_FOUND
JOB_ERROR:NODE_NOT_FOUND
JOB_ERROR:ALREADY_IN_GROUP

## 5.1.28. Listing Groups

A list of all existing groups may be retrieved with the GROUP_LIST command:

*FORMAT:*
NEXXUS:GROUP_LIST
*RESPONSES:*
JOB_STARTED
GROUP_LIST:<GROUP NAME>
JOB_COMPLETED

## 5.1.29. Adding a User To a Group

A group may be added to the list of groups a user is in by using the GROUP_ADD_ADMIN command:

*FORMAT:*
NEXXUS:GROUP_ADD_ADMIN:<GROUP NAME>:<USER NAME>

*RESPONSES:*
```
JOB_STARTED
JOB_ERROR:GROUP_NOT_FOUND
JOB_ERROR:USER_NOT_FOUND
JOB_ERROR:ALREADY_IN_GROUP
JOB_COMPLETED
```

## 5.1.30. Removing a User From a Group

A group may be removed from the list of groups a user is in by using the GROUP_DEL_ADMIN command:

*FORMAT:*
```
NEXXUS:GROUP_DEL_ADMIN:<GROUP NAME>:<USER NAME>
```
*RESPONSES:*
```
JOB_STARTED
JOB_ERROR:GROUP_NOT_FOUND
JOB_ERROR:USER_NOT_FOUND
JOB_ERROR:NOT_IN_GROUP
JOB_COMPLETED
```

## 5.1.31. Getting and Setting Node Global Variables

Getting and Setting of node global variables can be done using the GET_VAR and SET_VAR commands:

*FORMAT:*
```
NEXXUS:GET_VAR:<NODE_ID>:<VARIABLE_NAME>
```
*FORMAT:*
```
NEXXUS_SET_VAR:<NODE_ID>:<VARIABLE_NAME>:<VALUE>
```

*RESPONSES:*
JOB_STARTED
JOB_ERROR:NODE_NOT_FOUND
JOB_ERROR:VAR_NOT_SET
JOB_COMPLETED

## 5.1.32. Getting a Node Global Variable From All Nodes

Sometimes client may want to retrieve a particular variable from all nodes. The NODE_VAR_LIST allows you to do this. If the variable is not set on a node, the <VALUE> field will return empty.

*FORMAT:*
NEXXUS:NODE_VAR_LIST:<VARIABLE_NAME>
*RESPONSES:*
JOB_STARTED
NODEVARLIST:<JOB_ID>:<VARIABLE_NAME>:<VALUE>
JOB_COMPLETED

## 5.1.33. Toggling Nexxus Debug Mode

To enable or disable debug mode on a running Nexxus (which is the same as launching Nexxus with the -d switch on the commandline):

*FORMAT:*
NEXXUS:NEXXUS_DEBUG:ON
*FORMAT:*
NEXXUS:NEXXUS_DEBUG:OFF
*RESPONSES:*
JOB_STARTED

```
JOB_ERROR:INVALID_ARGUMENT
JOB_COMPLETED
```

## 5.1.34. Listing Users Currently Online

To list the users currently connected to a Nexxus, use the ’LIST_CLIENTS’ command:

*FORMAT:*
```
NEXXUS:LIST_CLIENTS
```
*RESPONSES:*
```
JOB_STARTED
NEXXUS:<JOB_ID>:LIST_CLIENTS:<FD>:<USERNAME>:<IDLE SECONDS>
JOB_COMPLETED
```

## 5.1.35. Sending a Message to All Online Users

To send an online message string to all online users, use the ’WALL’ command:

*FORMAT:*
```
NEXXUS:WALL:<MSG>
```
*RESPONSES:*
```
JOB_STARTED
NEXXUS:<JOB_ID>:WALL:<USER>:<IP_ADDRESS>:<MSG>
JOB_COMPLETED
```

### 5.1.36. Unsolicited Messages

The only unsolicited messages supported by nexxus is the ’WALL’ message indicating a message has been sent from an online user:

```
NEXXUS:0:WALL:<USER>:<IP_ADDRESS>:<MSG>
```

## 5.2. EMP Module

The Emergency Management Port module (EMP) is designed to communicate with an IPMI compliant Baseboard Management Controller (BMC) over either the serial port, or a serial port connected to a network terminal server. On Intel(tm) based motherboards, the EMP port is hardwired to COM2 (ttyS1 in Linux). Through the BIOS, this port can be enabled and configured. In this section, we will go through the EMP module features, how to setup your management topology for using EMP to manage multiple machines, how to setup EMP on an Intel based machine, and how to perform EMP operations using the VACM EMP module.

### 5.2.1. Module Features

The EMP module provides the end user or client with the ability to perform many low level hardware monitoring and management functions. Many of these functions are available ’Out of Band’, meaning that they can be performed regardless of hardware power state or condition. Some of these functions include:

- Chassis power on, power off, hard reset, power cycle
- Retrieval of hardware inventory information such as Board Manufacturer, Product Name, Serial Numbers

- Ability to set an asset tag (or other string) that is maintained in the hardware NVRAM and available to the local operating system
- Board level hardware failure and warning log management
- Realtime chassis hardware status queries
- Realtime sensor data retrieval supporting onboard temperature, fan speed, and voltage sensors
- Front panel power light flash control
- Sensor value threshold retrieval

## 5.2.2. Setting Up a Management Topology for EMP

For best results, spend some time planning how the nodes connect to the Nexxus Node Controller. EMP is a peer to peer implimentation; you need one serial port for every node you wish to connect to. If you are using console mode, it is recommended you use COM1 for console redirection, requiring two serial ports for every node you wish to monitor and manage. There are many ways such a setup can be accomplished.Multiport serial cards such as the Comtrol Rocketport(tm) work well since they have low latency, however, each card takes up a PCI slot on your Node Controller. Rocketports are currently available at a density of 32 ports per card, giving you the capability of servicing 16 nodes per card if you are using console mode on the recommended second serial port. High end terminal servers such as Cisco(tm) chassis with ASYNC cards work extremely well over a network, the only requirement being that the physical network that carries the EMP data between the terminal server and the Node Controller should be isolated as to ensure that only EMP traffic is travelling on the wire. If you don’t isolate EMP traffic, network traffic spikes could cause latency problems with the EMP hardware. Cisco hardware is also generally a lot more expensive than conventional multiport serial cards. Shown below are two diagrams illustrating the aforementioned EMP wiring strategies: [INSERT ROCKETPORT PICTURE HERE] [INSERT CISCO PICTURE HERE]

## 5.2.3. Configuring EMP On a Remote System

The EMP hardware can be configured on most systems through the system BIOS. Currently supported platforms are the Intel Nightshade, Nightlight, and Lancewood motherboards. To enable and configure the EMP properly for VACM use with these platforms, enter the BIOS configuration area and go to the ’Server Management’ section. You should configure the EMP parameters to look like this:

**Figure 5-1. Bios Setup for EMP**

If you are going to enable console redirection mode so you can remotely enter the BIOS and effect changes, enter the ’Console Redirection’ level and set the parameters as:

**Figure 5-2. Bios Setup for Serial Console**

This configuration shows the console to be connected to COM1. This is the *recommended* behaviour. You *can* setup the console on COM2, but if your system has a problem booting or otherwise locks up while console mode is engaged, then you will *not* be able to reset it remotely via EMP. You should also enable the boot time diagnostics screen. If the diagnostics screen is disabled, console mode will not function properly.

The EMP module has a number of extended features that are not normally available via EMP but are related to the BMC. These extended features are accessed via a TCP/IP network, communicating with the EMP_EXTENTD daemon that must be running on all target machines. Since a daemon is used on the remote machine, these features are only available ’in-band’; when the operating system is up and running. The emp_extentd daemon allows the EMP module to perform realtime sensor queries, and blink the front panel power indicator. The extensions daemon also provides support for the system hardware watchdog built into the BMC. The daemon can be configured to automatically power cycle the system, if the operating system locks up due to software

or hardware failure. The extensions daemon communicates with the BMC via the IPMI KCS interface. This interface is a local ISA bus interface that the local system uses to perform low level BMC operations. The IPMI_KCS driver must be installed in the node's running kernel for the extensions daemon to communicate with the BMC. The IPMI_KCS driver patches are available in the `nexxus/nexxus_modules/emp/support_utilities` directory, and apply to the 2.2.14 kernel (the patch should apply to newer 2.2 kernels just fine). To install the patch, log into the remote node and transfer the patch over. Then apply the patch to your running kernel. The following example assumes the patch has been transferred to the `/usr/src/linux directory/`.

```
[root@H101 /]# cd /usr/src/linux
[root@H101 linux]# patch -p 1  ../ipmi_kcs_version.linux-2.2.14.patch
patching file ‘arch/i386/kernel/traps.c’
patching file ‘drivers/char/Config.in’
patching file ‘drivers/char/Config.in.orig’
patching file ‘drivers/char/Makefile’
patching file ‘drivers/char/Makefile.orig’
patching file ‘drivers/char/ipmi_kcs.c’
patching file ‘drivers/char/ipmi_kcs.h’
patching file ‘drivers/char/misc.c’
patching file ‘include/linux/ipmi_kcs_ioctls.h’
patching file ‘include/linux/miscdevice.h’
[root@H101 linux]#
```

The file `ipmi_kcs.c`, `ipmi_kcs.h`, `ipmi_kcs_ioctls.h`, `miscdevice.h`, and `misc.c` are the actual source files for the driver. The `traps.c` file contains a modified NMI handler which will cause the front panel power indicator to begin flashing should the daemon be configured in "non-power-cycle- mode" and the watchdog times out.

Once the patch is installed, it is necessary to reconfigure the kernel to use the driver. Using the kernel configuration tool of your choice, enter the ’character devices’ menu and enable the ’IPMI KCS Interface’. Once enabled, save your kernel configure, rebuild your kernel and install.

The extensions daemon itself takes two command line arguments. The first argument determines whether or not the daemon opens a network socket and waits for sensor

requests from the EMP Module. This is for situations where a user may wish to take advantage of the watchdog functions of the daemon, but not the sensor functions. This argument is either ’ON’ or ’OFF’. The second argument is the watchdog firing action which can either be set to ’REBOOT’ or ’NOREBOOT’. In REBOOT mode, the hardware watchdog will power-cycle the machine if the watchdog fires. In NOREBOOT mode, the hardware generates an NMI (non maskable interrupt) that causes the kernel to *attempt* to blink the front panel power indicator (depending on the cause of the software or hardware failure, the kernel may not be in a state to execute code).

Here is an example of running the daemon in the foreground with sensor extensions enabled, in reboot mode:

```
[root@H101 /root]# ./emp_extentd on reboot
VACM EMP Extensions Daemon (c) 1999,2000 San Mehat (nettwerk@valinux.com)
--Sensor Extensions enabled.
--Watchdog Action is REBOOT.
```

The following example shows running the daemon in the foreground with sensor extensions enabled, and in non-reboot mode:

```
[root@H101 /root]# ./emp_extentd on noreboot
VACM EMP Extensions Daemon (c) 1999,2000 San Mehat (nettwerk@valinux.com)
--Sensor Extensions enabled.
--Watchdog Action is NMI ONLY.
```

The IPMI_KCS driver provides a /proc/ipmi interface that is used to see watchdog status, and view hardware inventory records stored in the NVRAM. The interface is shown below:

```
Driver Version  : 1.1
BMC Version     : 1.14
IPMI Version    : 0.9

WD Timer Status         : STARTED
WD Timeout Action       : NONE
WD Pre-Timeout IRQ      : NMI
```

```
WD Last Pet             : 964752365 (28 seconds ago)
WD countdown            : 27

Board Area Records:
Board Manuf.    : Intel
Board Prod Name : L440GX+
Board Serial    : IMLW94304081
Board Part      : 721242-008

Product Area Records:
Product Manuf.  : VA Linux
Product Name    : FullOn 2x2
Product Part    : VAR101679
Product Version : Not Available
Product Serial  : Not Available
Product Asset   : San’s FullON(26 byte maximum size)
```

The first section displays the driver version, BMC firmware version, and IPMI version compliance numbers. The second section displays information on the watchdog. WD Timer Status indicates weather the watchdog is currently running or not. WD Timeout Action indicates what action is taken when the watchdog timer expires. WD Pre-Timeout IRQ indicates what action is taken just before the watchdog timer expires. The WD Last Pet field gives the system tick timer timestamp of the last watchdog ping. WD countdown indicates the current hardware countdown value of the watchdog in 1/10’s of a second. The third section contains information about to the motherboard manufacturer, name, serial number, and part number. The third field contains information about the overall Product manufacturer name, part, version and serial number. The last field contains the currently configured asset tag. The asset tag can be programmed via the EMP module remotely and can be used for inventory control, or storing small bits of configuration data such as an IP address, or ethernet hardware address.

## 5.2.4. Configuring the EMP Module To Manage a Node

Before a node can be managed by the EMP module, the module needs to be told what serial port device or what TCP/IP address and port to connect to. Also, if the target node's BIOS has been configured to require an EMP password, the module will need to be notified. The CONFIGURATION command is used to either configure the device and password parameters for a node, or to disable VACM EMP management for a node altogether. If no password is configured, the PASSWORD field should be 'NONE'. To disable VACM EMP management for a node, the DEVICE/ADDRESS field should be 'NONE'

```
FORMAT:
EMP:CONFIGURATION:<NODE_ID>:<DEVICE/ADDRESS>:<PASSWORD>
RESPONSES:
JOB_STARTED
STATUS:<STATUS>
JOB_ERROR:(errno string)
JOB_COMPLETED
FIELDS:
<DEVICE/ADDRESS> - either a /dev/device entry, or a network
                   address and port to connect to. Network
                   addresses should be specified as
                   xxx.xxx.xxx-yyy, where x is the network
                   address portion, and y is the target
                   port portion. If <DEVICE/ADDRESS> is
                   set to 'NONE', then EMP management
                   and monitoring is disabled for the
                   specified node.
<STATUS> - As the EMP module negotiates with the target
           node hardware, the
           following may be returned to
           signify the negotiation progress:
'ENGAGING'              - Thread is engaging to port
'ENGAGE_FAILED'         - Unable to engage to port
'PROTOCOL_DETECTED'     - EMP protocol detected
'PROTOCOL_UNAVAILABLE' - EMP protocol not detected on port
'CONNECTION_ACCEPTED'   - Connection accepted by hardware
```

```
'CONNECTION_DENIED'    - Bad password during connection
                         authentication with h/w
'INTERFACE_TEMP_LOCKED'- Too many bad passwords; interface
                         locked for 30 seconds
'DOWNLOADING_FRU'      - FRU download in progress
'DOWNLOADING_SDR'      - SDR download in progress
'DOWNLOADING_SEL'      - SEL download in progress
```

*NOTES:*

If you receive a PROTOCOL_UNAVAILABLE message, the node *may* be in console mode (which is part of the reason you shouldn't do console on COM2). If the node goes into console mode at any point during the negotiation, a 'Protocol driver not attached' JOB_ERROR message will be sent.

## 5.2.5. Retrieving a Remote Node's BMC Information

The current version of BMC firmware and IPMI version compliance is obtained with the 'BMC_INFO' command:

*FORMAT:*

```
EMP:BMC_INFO:<NODE_ID>
```

*RESPONSES:*

```
JOB_STARTED
BMCINFO:<BMC FIRMWARE VERSION>:<BMC MANUFACTURER>:
        <IPMI_VERSION>:<HARDWARE_REVISION>
JOB_ERROR:(errno string)
JOB_COMPLETED
```

## 5.2.6. Retrieving a Configured Node's Module Configuration

Retrieve a node's configuration from the EMP module with the 'INQUIRY' command:

```
FORMAT:
EMP:INQUIRY:<NODE_ID>
RESPONSES:
JOB_STARTED
INQUIRY:<NODE_ID>:<SERIAL NUMBER>:
        <DEVICE ADDRESS>:<STATE>:<PASSWORD>
JOB_ERROR:(errno string)
JOB_COMPLETED
NOTES:
IP address information can be retrieved with the nexxus 'GET_VAR' com-
mand.
```

## 5.2.7. Retrieving a Node's Current Connection Status

Determine the protocol state of the node with the 'NODE_STATUS' command:

```
FORMAT:
EMP:NODE_STATUS:<NODE_ID>
RESPONSES:
JOB_STARTED
NODESTATUS:<DEVICE ADDRESS>:<STATE>
JOB_ERROR:(errno string)
JOB_COMPLETED
FIELDS:
<STATE> can be one of:
'NEGOTIATING' - Negotiation in progress
'CONSOLE'     - Node is in console mode
'DEAD'        - Pending internal removal
'INITIALIZING'- Thread is starting
```

```
'DETECTED'    - Protocol detected (Everything OK)
'NOT_DETECTED'- Protocol not detected. (Waiting for protocol)
```

## 5.2.8. Retrieving a Node's Inventory Information

Obtain inventory information from the FRU with the 'NODE_INV_INFO' cmd. This information is static and kept in the target nodes NVRAM

*FORMAT:*

```
EMP:NODE_INV_INFO:<NODE_ID>
```

*RESPONSES:*

```
JOB_STARTED
NODEINVINFO:<B_MAN>:<B_NAME>:<B_PART>:<B_SER>:
            <B_ETH>:<P_MAN>:<P_NAME>:<P_PART>:
            <P_VER>:<P_SER>:<ASSET>
JOB_ERROR:(errno string)
JOB_COMPLETED
```

*FIELDS:*

```
<B_MAN> - Board Manufacturer
<B_NAME>- Board Name
<B_PAR> - Board Part Number
<B_SER> - Board Serial Number
<B_ETH> - Board Ethernet Address (NOT USED)
<P_MAN> - Product Manufacturer
<P_NAME>- Product Name
<P_PART>- Product Part Number
<P_VER> - Product Version
<P_SER> - Product Serial Number
<ASSET> - Asset Tag
```

## 5.2.9. Setting the Asset Tag on a Node

The asset tag for a node is stored in the hardware NVRAM. This asset tag can be used to store any type of string data from inventory control information to configuration data. The asset tag can be read remotely through the module using the NODE_INV_INFO command, or locally using the IPMI_KCS driver's `/proc/ipmi` interface. The size of the asset tag is restricted to the field space allocated in the FRU. By default on some systems this may be 0. To ensure enough space, reload your FRU and SDRs using your vendor supplied FRU and SDR updater and specify an asset tag with as many placeholder bytes as you wish to program. VA Linux customers may have the field length already expanded. If not, contact technical support and request the BMC/SDR/FRU updater floppy images for your particular server. If you attempt to write a tag larger than the served space in the FRU, you will receive an error. To set the asset tag, use the SET_ASSET_TAG command.

*FORMAT:*
EMP_SET_ASSET_TAG:<NODE_ID>:<TAG>
*RESPONSES:*
JOB_STARTED
JOB_ERROR:(errno string)
JOB_COMPLETED

## 5.2.10. Retrieving a Node's Current Chassis Status

The current chassis status of a node's chassis can be obtained with the CHASSIS_STATUS command. This command also returns the last power event (if any).

*FORMAT:*
EMP:CHASSIS_STATUS:<NODE_ID>
*RESPONSES:*
JOB_STARTED
CSTATUS:<PS_PWRON>:<PS_POVER>:<PS_INTLCK>:<PS_CTLFLT>:

```
        <PS_PWRRSTPOL>:<LPE_ACFAIL>:<LPE_POVER>:
        <LPE_INTLCK>:<LPE_PFLT>:<LPE_CMD>:<M_INT>:
        <M_SEC>:<M_DFLT>:<M_FAN>
JOB_ERROR:(errno string)
JOB_COMPLETED
FIELDS:
<PS_PWRON>     - Power State - Power On?
<PS_POWER>     - Power State - Power Overload?
<PS_INTLCK>    - Power State - Interlock Active?
<PS_PFLT>      - Power State - Power Fault?
<PS_CTLFLT>    - Power State - Power Control Fault?
<PS_PWRRSTPOL> - Power State - Power Restore Policy?
<LPE_ACFAIL>   - Last Power Event - AC Failed?
<LPE_POVER>    - Last Power Event - Power Overload?
<LPE_INTLCK>   - Last Power Event - Interlock Power Down?
<LPE_PFLT>     - Last Power Event - Power Fault?
<LPE_CMD>      - Last Power Event - Commanded ON/OFF?
<M_INT>        - Misc Chassis State - Intrusion Active?
<M_SEC>        - Misc Chassis State - Secure Mode Active?
<M_DFLT>       - Misc Chassis State - Drive Fault?
<M_FAN>        - Misc Chassis State - Cooling Fan Fault?
```

## 5.2.11. Retrieving a List of Chassis Capabilities

A brief list of chassis capabilities can be retrieved with this command. These chassis capabilities can be used to determine if some of the response fields of a CHASSIS_STATUS are valid. Chassis capabilities are retrieved with the CHASSIS_CAPABILITIES cmd.

```
FORMAT:
EMP:CHASSIS_CAPABILITIES:<NODE_ID>
RESPONSES:
JOB_STARTED
CCAPABLE:<PROVIDES_INTRUSION?>:<PROVIDES_SECUREMODE?>:
```

```
          <PROVIDES_FPNMI?>:<PROVIDES_POWERINTERLOCK?>
JOB_ERROR:(errno string)
JOB_COMPLETED
```

## 5.2.12. Powering Down the Chassis

The node can be powered down with the POWER_DOWN command. Depending on the state of the remote hardware, there may be a short delay before the hardware actually powers down. This command is a restricted command. If restricted mode is enabled in the remote nodes BIOS, an error will be returned.

*FORMAT:*
EMP:POWER_OFF:<NODE_ID>
*RESPONSES:*
JOB_STARTED
JOB_ERROR:(errno string)
JOB_COMPLETED

## 5.2.13. Powering Up the Chassis

The node can be powered up with the POWER_ON command. This command is a restricted command. If restricted mode is enabled in the remote nodes BIOS, an error will be returned.

*FORMAT:*
EMP:POWER_ON:<NODE_ID>
*RESPONSES:*
JOB_STARTED
JOB_ERROR:(errno string)
JOB_COMPLETED

## 5.2.14. Hard Resetting the Chassis

The node can be hard reset with the HARD_RESET command. This command is a restricted command. If restricted mode is enabled in the remote nodes BIOS, an error will be returned.

*FORMAT:*
EMP:HARD_RESET:<NODE_ID>
*RESPONSES:*
JOB_STARTED
JOB_ERROR:(errno string)
JOB_COMPLETED

## 5.2.15. Power Cycling the Chassis

The node can be power cycled with the POWER_CYCLE command. When issued, the target node firmware will power off the node, wait a few seconds, then power it up again. This command is a restricted command. If restricted mode is enabled in the remote nodes BIOS, an error will be returned.

*FORMAT:*
EMP:POWER_CYCLE:<NODE_ID>
*RESPONSES:*
JOB_STARTED
JOB_ERROR:(errno string)
JOB_COMPLETED

## 5.2.16. Sending a Chassis Front Panel NMI

If the target node supports Front Panel NMI (See CHASSIS_CAPABILITIES cmd), a front panel NMI can be strobed with the PULSE_FP_NMI command. The function of this command is vendor specific. This command is a restricted command. If restricted mode is enabled in the remote nodes BIOS, an error will be returned.

*FORMAT:*
```
EMP:PULSE_FP_NMI:<NODE_ID>
```
*RESPONSES:*
```
JOB_STARTED
JOB_ERROR:(errno string)
JOB_COMPLETED
```

## 5.2.17. Downloading the System Event Log

The EMP hardware system event log can be retrieved with the ’DOWNLOAD_LOG’ command. The log is downloaded from the module’s disk based node SEL file. The module queries the hardware every 5 seconds, and downloads any new entries. New entries are then deleted from the hardwarei’s event buffer to make room for new events. Since the module attempts to modify the remote SEL buffer whenever it queries it, restricted mode must be disabled for SEL functions to work.

*FORMAT:*
```
EMP:DOWNLOAD_LOG:<NODE_ID>
```
*RESPONSES:*
```
JOB_STARTED
SEL:<NODE_SERIAL>:<RECORD_ID>:<RECORD_TYPE>:<RECORD_TIMESTAMP>:
    <RECORD_GENERATOR>:<RECORD_FORMAT>:<SENSOR_TYPE>:<SENSOR_NAME>:
    <DESCRIPTION>
JOB_ERROR:(errno string)
JOB_COMPLETED
```

## 5.2.18. Clearing the System Event Log

The CLEAR_LOG command is used to remove all entries from the disk based system event log that the module keeps for a specific node.

*FORMAT:*
```
EMP:CLEAR_LOG:<NODE_ID>
```
*RESPONSES:*
```
JOB_STARTED
JOB_ERROR:(errno string)
JOB_COMPLETED
```

## 5.2.19. Receiving System Event Logs Asynchronously

Sometimes a client may want to receive system event log entries as they come in, and not have to constantly poll for them. The LOG_TAIL command is used to enable and disable asynchronous event log entry notification.

*FORMAT:*
```
EMP:LOG_TAIL:<NODE_ID>:<ON/OFF>
```
*RESPONSES:*
```
JOB_STARTED
JOB_ERROR:(errno string)
JOB_COMPLETED
```

## 5.2.20. Retrieving a List of Sensors on a Node

A list of available sensors on each node and their types can be retrieved with the GET_SENSOR_LIST command.

```
FORMAT:
EMP:GET_SENSOR_LIST:<NODE_ID
RESPONSES:
JOB_STARTED
SENSOR:SENSOR_NUMBER:<NAME>:<TYPE>
JOB_ERROR:(errno string)
JOB_COMPLETED
FIELDS:
<TYPE> - Either 'ANALOG', 'DIGITAL', or 'UNSUPPORTED'.
```

## 5.2.21. Retrieving Sensor Thresholds for a Sensor

Threshold values for a sensor can be retrieved with the GET_SENSOR_THRESHOLDS command.

```
FORMAT:
EMP:GET_SENSOR_THRESHOLDS:<NODE_ID>:<SENSOR_NUMBER>
RESPONSES:
JOB_STARTED
THRESHOLD:<SENSOR_NUMBER>:<MIN_VAL>:<LWR_NONCRIT>:<LWR_NONREC>:
          <LWR_CRIT>:<UPR_NONCRIT>:<UPR_NONREC>:<UPR_CRIT>:<MAX_VAL>
JOB_ERROR:(errno string)
JOB_COMPLETED
FIELDS:
<MIN_VAL>     - Minimum Sensor Value
<LWR_NONCRIT> - Lower Non-Critical Threshold
<LWR_NONREC>  - Lower Non-Recoverable Threshold
<LWR_CRIT>    - Lower Critical Threshold
<UPR_NONCRIT> - Upper Non-Critical Threshold
```

```
<UPR_NONREC>  - Upper Non-Recoverable Threshold
<UPR_CRIT>    - Upper Critical Threshold
<MAX_VAL>     - Maximum Sensor Value
NOTES:
Fields which have a return value of 0.0 indicate no threshold set.
```

## 5.2.22. Reading a Sensor

A realtime sensor reading can be obtained from a node with the READ_SENSOR command. This command will function only if the EMP Extensions Daemon is running on the remote node.

```
FORMAT:
EMP:READ_SENSOR:<NODE_ID>:<SENSOR_NUMBER>
RESPONSES:
JOB_STARTED
SENSOR:<SENSOR_NUMBER>:<VAL>:<UNIT>
SENSOR:<SENSOR_NUMBER>:<CONDITION>
JOB_ERROR:(errno string)
JOB_COMPLETED
FIELDS:
<VAL>       - Numerical value (Returned for ANALOG sensors only)
<UNIT>      - String unit name which VAL refers to
              (Returned for ANALOG sensors only)
<CONDITION> - Condition string for the sensor
              (Returned for DIGITAL sensors only)
```

## 5.2.23. Setting the Flash State of a Node's Front Panel Power Indicator

The IDENTIFY command is used to change the state of the front panel power indicator on a node. When identify is turned ON, the front panel power indicator will flash. When identify is turned OFF, the front panel power indicator will return to it's normal state. This command wil function only if the EMP Extensions Daemon is running on the remote node, and is currently only available on Intel LW440GX (Lancewood) class motherboards.

*FORMAT:*
```
EMP:IDENTIFY:<NODE_ID>:<ON/OFF>
```
*RESPONSES:*
```
JOB_STARTED
JOB_ERROR:(errno string)
JOB_COMPLETED
```

## 5.2.24. Resetting the EMP Module for a Node

The REFRESH command is used to reset the module thread for a node. The thread will disconnect from the EMP port, reconnect and renegotiate. See the CONFIGURATION command for a description of response messages.

*FORMAT:*
```
EMP:REFRESH:<NODE_ID>
```
*RESPONSES:*
```
(see section 4.2.6)
```

## 5.2.25. Unsolicited Messages

The following unsolicited messages may be sent by the EMP module at any time:
SEL:<NODE_SERIAL>:<RECORD_ID>:<RECORD_TYPE>:<RECORD_TIMESTAMP>:
<RECORD_GENERATOR>:<RECORD_FORMAT>:<SENSOR_TYPE>:<SENSOR_NAME>:
<DESCRIPTION>

## 5.2.26. EMP Module Node Global Variable Requirements

The EMP module requires that the *IP_ADDRESS* variable be set to the internet address of a node only if sensor requests are to be performed.

## 5.2.27. EMP Module Supported Hardware List

The following hardware has been qualified for remote management with the EMP module:

- Intel Nightshade / Nightlite Server Boards
- Intel Lancewood Server Boards rev A -> G with BMC firmware 1.14
- Intel Koa/D'Iberville Server Boards

The following hardware has been qualified for Node Controller interface with remote EMP hardware:

- Comtrol Rocketport cards & Rocketport breakout boxes
- Cisco Systems 36xx chassis with 16/32 port async cards
- Standard 16550 internal serial ports

The following Linux kernels have been qualified for use in Node Controllers:

- Linux 2.2.13 and greater (SMP/UP)

# 5.3. VASENET Module

The VASENET module provides management capabilities for VA Linux 1120 and 1220 servers through the use of a VA100.

## 5.3.1. Module Features

The VASENET module provides the end user or client with the following capabilities:

- Reset a node
- Power on a node
- Identify a node
- Node status

## 5.3.2. Configuring a VA100

```
FORMAT:
VASENET:CONFIGURATION:<NODE_ID>:MASTER:<PASSWORD>
RESPONSES:
JOB_STARTED
JOB_ERROR:(errno string)
JOB_COMPLETED
FIELDS:
<PASSWORD> - The password to use when connecting to a VA100
```

## 5.3.3. Configuring a Managed Node

*FORMAT:*
VASENET:CONFIGURATION:<NODE_ID>:SLAVE:<MASTER_NAME>:<VASENET_ADDRESS>
*RESPONSES:*
JOB_STARTED
JOB_ERROR:(errno string)
JOB_COMPLETED
*FIELDS:*
<MASTER_NAME> - The name of the VA100 that controls this node.
<VASENET_ADDRESS> - The six digit hardware address of this node.

## 5.3.4. Querying a Nodes Configuration

*FORMAT:*
VASENET:INQUIRY:<NODE_ID>
*RESPONSES:*
JOB_STARTED
INQUIRY:MASTER:<PASSWORD>
INQUIRY:SLAVE:<MASTER_NAME>:<VASENET_ADDRESS>
JOB_COMPLETED

## 5.3.5. Retrieving a Nodes Software Revision

*FORMAT:*

```
VASENET:VASENET_VERSION:<NODE_ID>
RESPONSES:
JOB_STARTED
VASENET_VERSION:<VASENET_VERSION_STRING>
JOB_COMPLETED
```

## 5.3.6. Listing a VA100's Managed Nodes

```
FORMAT:
VASENET:NODE_LIST:<NODE_ID>
RESPONSES:
JOB_STARTED
NODE_LIST:<VASENET ADDRESS>
JOB_COMPLETED
```

## 5.3.7. Refreshing a VA100's Connection

```
FORMAT:
VASENET:REFRESH:<NODE_ID>
RESPONSES:
JOB_STARTED
JOB_COMPLETED
```

## 5.3.8. Rescanning a VA100’s List of Managed Nodes

*FORMAT:*
VASENET:RESCAN:<NODE_ID>
*RESPONSES:*
JOB_STARTED
JOB_COMPLETED

## 5.3.9. Resetting a VA100

*FORMAT:*
VASENET:RESET:<NODE_ID>
*RESPONSES:*
JOB_STARTED
JOB_COMPLETED

## 5.3.10. Retrieving a VA100’s Status

*FORMAT:*
VASENET:STATUS:<NODE_ID>
*RESPONSES:*
JOB_STARTED
STATUS:<NUMBER OF NODES>:<NUMBER OF FAN ALARMS>
JOB_COMPLETED

## 5.3.11. Listing a VA100’s Managed Nodes

*FORMAT:*
VASENET:NODE_LIST:<NODE_ID>
*RESPONSES:*
JOB_STARTED
NODE_LIST:<VASENET_ADDRESS>
JOB_COMPLETED

## 5.3.12. Powering on a Node

*FORMAT:*
VASENET:POWER_ON:<NODE_ID>
*RESPONSES:*
JOB_STARTED
JOB_COMPLETED

## 5.3.13. Powering off a Node

*FORMAT:*
VASENET:PORT_OFF:<NODE_ID>
*RESPONSES:*
JOB_STARTED
JOB_COMPLETED

## 5.3.14. Rebooting a Node

*FORMAT:*
VASENET:HARD_RESET:<NODE_ID>
*RESPONSES:*
JOB_STARTED
JOB_COMPLETED

## 5.3.15. Identifying a Node

*FORMAT:*
VASENET:IDENTIFY:<NODE_ID>
*RESPONSES:*
JOB_STARTED
JOB_COMPLETED

## 5.3.16. Querying a Nodes Status

*FORMAT:*
VASENET:CHASSIS_STATUS:<NODE_ID>
*RESPONSES:*
JOB_STARTED
CHASSIS_STATUS:<POWER>:<ALARMS>:<NIC>:<HD>
JOB_COMPLETED
*FIELDS:*
<POWER> - Upper case indicates asserted.
<ALARMS> - Upper case indicates asserted.
<NIC> - Upper case indicates asserted.

```
<HD> - Upper case indicates asserted.
```

## 5.4. VA1000 Module

The `VA1000` module supports the management of VA 1000 server clusters. The VA 1000 has a management microcontroller and a dedicated Cluster Management Bus (CMBus) to provide remote management.

Each node in a VA 1000 cluster has a unique address that is determined dynamically when power is first applied. To insure that the addresses are properly allocated, follow the instructions in Appendix C - Clustering VA 1000 Nodes.

To get a node's CMBus address, press and hold the Power button down for 5 seconds, and release. The two rightmost LED's will flash, indicating the start of the CMBus address information. Next, the three rightmost LED's will flash, indicating the 100's, 10's, and 1's or the node's CMBus address. Finally, the two rightmost LED's will flash again, indicating the end of the CMBus address information. See Appendix C for more information.

For the following commands, the following error responses are possible:

- `JOB_COMPLETED` - Successful completion.
- `MALFORMATTED_MESSAGE` - Incorrectly formatted message.
- `UNSUPPORTED_MESSAGE` - Command not available/supported.
- `MODULE_NOT_AVAILABLE` - VA 1000 hardware not present or available.
- `NODE_NOT_CONFIGURED` - Node's CMBus address unknown.
- `NODE_NOT_FOUND` - Node name is invalid/unknown.
- `NODE_NOT_RESPONDING` - Node's CMBus controller did not respond.
- `JOB_ERROR` - Unexpected error response.

## 5.4.1. Configuring the VA1000 Module to Manage a Node

In order to manage a VA 1000 node, the VA1000 module needs to know the cluster management bus (CMBus) address of the node. The CONFIGURATION command provides the CMBus address of a node to the VA1000 module.

```
FORMAT:
VA1000:CONFIGURATION:<name>:<cmbus_address>
RESPONSES:
JOB_STARTED
JOB_COMPLETED
  or
JOB_STARTED
JOB_ERROR:<error message>
FIELDS:
<cmbus_address> - The CMBus address is typically given in decimal.
```

## 5.4.2. Retrieving a Configured Node's Module Configuration

To retrieve the configuration information for a node from the VA1000 module, use the INQUIRY command.

```
FORMAT:
VA1000:INQUIRY:<name>
RESPONSES:
JOB_STARTED
INQUIRY:<name>:<cmbus_address>
JOB_COMPLETED
  or
```

```
JOB_STARTED
JOB_ERROR:<error message>
```

## 5.4.3. Powering Down the Chassis

A node can be powered down with the POWER_OFF command. A power-down request to the cluster controller node is ignored.

*FORMAT:*
```
VA1000:POWER_OFF:<name>
```
*RESPONSES:*
```
JOB_STARTED
JOB_COMPLETED
  or
JOB_STARTED
JOB_ERROR:<error message>
```

## 5.4.4. Powering Up the Chassis

A node can be powered up with the POWER_ON command. If the node is already powered, nothing happens.

*FORMAT:*
```
VA1000:POWER_ON:<name>
```
*RESPONSES:*
```
JOB_STARTED
JOB_COMPLETED
  or
JOB_STARTED
JOB_ERROR:<error message>
```

## 5.4.5. Power Cycling the Chassis

A node can be power cycled with the POWER_CYCLE command. This is equivalent to issuing a POWER_OFF command followed by a POWER_ON command.

```
FORMAT:
VA1000:POWER_CYCLE:<name>
RESPONSES:
JOB_STARTED
JOB_COMPLETED
  or
JOB_STARTED
JOB_ERROR:<error message>
```

## 5.4.6. Hard Resetting the Chassis

```
FORMAT:
VA1000:HARD_RESET:<name>
RESPONSES:
JOB_STARTED
JOB_COMPLETED
  or
JOB_STARTED
JOB_ERROR:<error message>
```

## 5.4.7. Retrieving a Node’s Current Chassis Status

The current state of a node’s chassis can be obtained with the CHASSIS_STATUS command. The available information is a subset of the chassis status information available on an EMP node.

```
FORMAT:
VA1000:CHASSIS_STATUS:<name>
RESPONSES:
JOB_STARTED
CSTATUS:<power_on>:<selected>:<reserved>:<reserved>:
        <reserved>:<identifying>:<reserved>:<reserved>
JOB_COMPLETED
  or
JOB_STARTED
JOB_ERROR:<errno string>
FIELDS:
<power_on>    - Power On?
<selected>    - Using the monitor/keyboard/mouse?
<identifying> - Blinking the rightmost two LED’s?
<reserved>    - Reserved for future use - returns "NO"
```

## 5.4.8. Selecting a Node to Use the Console

The VA 1000’s cluster management bus includes the monitor, keyboard, and mouse signals necessary to drive a console from any node in the cluster. Pressing the Power switch on a node "selects" that node to use the monitor, keyboard, and mouse as a console. The SELECT command provides the capability to select a node from software.

Only one node can be selected. Issuing a SELECT command will deselect the currently selected node, and select the new node to drive the console.

```
FORMAT:
VA1000:SELECT:<name>
```

```
RESPONSES:
JOB_STARTED
JOB_COMPLETED
  or
JOB_STARTED
JOB_ERROR:<error message>
NOTES:
If the node is off, SELECT will power it up.
If the node is identifying, SELECT will cause identifying to stop.
```

## 5.4.9. Identifying a Node in a Cluster

Issuing an IDENTIFY command makes it possible to locate a particular node in a cluster. A node identifies itself by blinking the two rightmost LED's on the front panel. Sending an IDENTIFY:..:OFF command or a SELECT command will cause the node to stop identifying. A node can identify itself even if the node is powered down.

```
FORMAT:
VA1000:IDENTIFY:<name>:<ON|OFF>
RESPONSES:
JOB_STARTED
JOB_COMPLETED
  or
JOB_STARTED
JOB_ERROR:<error message>
```

## 5.4.10. Reading EEPROM

Each VA 1000 node contains a small amount of EEPROM that is managed by the cluster management microcontrollers. The EEPROM_READ command reads one byte

from the EEPROM.

*FORMAT:*
```
VA1000:EEPROM_READ:<name>:<address>
```
*RESPONSES:*
```
JOB_STARTED
EEPROM_READ:<value (in hex)>
JOB_COMPLETED
  or
JOB_STARTED
JOB_ERROR:<error message>
```

## 5.4.11. Writing EEPROM

The EEPROM_READ command writes one byte to the EEPROM managed by the cluster management microcontrollers.

*FORMAT:*
```
VA1000:EEPROM_WRITE:<name>:<address>:<value>
```
*RESPONSES:*
```
JOB_STARTED
JOB_COMPLETED
  or
JOB_STARTED
JOB_ERROR:<error message>
```

## 5.4.12. Displaying VA1000 Module Global State

The DEBUG command allows inspection of module globals to aid in debugging problems. The DUMP subcommand dumps all the globals, while the GET

subcommand dumps one global by name. The SET command, if enabled, allows global values to be altered. The typical use of SET would be to raise the debug level to generate more verbose logging output. The ability to set globals is disabled by default.

*FORMAT:*

```
VA1000:DEBUG:<name>:<DUMP|GET|SET>[:<variable_name>[:<value>]]
```

*RESPONSES:*

```
JOB_STARTED
DEBUG:g_local_node:10
DEBUG:g_debug_level:1
DEBUG:g_smb_fd:8
DEBUG:g_cmd_ctr:7
DEBUG:g_timeout:5
DEBUG:lw-207:10:10:i:s:d
DEBUG:lw-208:2:2:i:s:d
DEBUG:m:123:123:i:s:d
DEBUG:missing::-1:i:s:d
JOB_COMPLETED
  or
JOB_STARTED
JOB_ERROR:<errno string>
```

## 5.4.13. Issuing Cluster Management Commands Directly

If enabled, the RAW command makes it possible to issue a cluster management command directly to the cluster management microcontroller. This command is disabled by default. Result string depends on the command that was issued.

*FORMAT:*

```
VA1000:RAW:<name>:<addr cmd result len d1 d2 d3>
```

*RESPONSES:*

```
JOB_STARTED
RAW:<response bytes>
JOB_COMPLETED
```

```
  or
JOB_STARTED
JOB_ERROR:<errno string>
```

# 5.5. SERCON Module

The SERCON module allows a user to communicate with a remote node's serial console. The remote node must support some form of serial based console. In this section, we will cover the SERCON module features, how to setup serial console on a variety of different nodes, and how to use the SERCON module.

## 5.5.1. Module Features

The SERCON module provides more than just the ability to manage a remote node via it's console. Multiple administrators may be connected to the same console simultaneously, although only one is able to write. An alert mechanism is also supported to trap critical events that may wish to be logged. By specifying an alert string, along with the number of lines to capture, critical messages or events can be saved in the modules alert log for later review.

## 5.5.2. Setting Up Serial Console on Remote Systems

## 5.5.3. Configuring the SERCON Module To Manage a Node

The CONFIGURATION command is used to configure the device/address, and

baudrate parameters for a remote console. To disable VACM SERCON management for a node, set the DEVICE/ADDRESS to ’NONE’

*FORMAT:*
SERCON:CONFIGURATION:<NODE_ID>:<DEVICE/ADDRESS>:<BPS>
*RESPONSES:*
JOB_STARTED
JOB_ERROR:(errno string)
JOB_COMPLETED
*FIELDS:*
<DEVICE/ADDRESS> - either a /dev/device entry, or a network address and port to connect to. Network addresses should be specified as xxx.xxx.xxx-yyy, where x is the network address portion, and y is the target port portion. If <DEVICE/ADDRESS> is set to ’NONE’, then SERCON management and monitoring is disabled for the specified node.

## 5.5.4. Reading Back a Nodes SERCON Configuration

The device and baudrate settings for a node are obtained with the ’INQUIRY’ command.

*FORMAT:*
SERCON:INQUIRY:<NODE_ID>
*RESPONSES:*
JOB_STARTED
INQUIRY:<NODE_ID>:<DEVICE_ADDRESS>:<BAUDRATE>
JOB_ERROR:(errno string)
JOB_COMPLETED

## 5.5.5. Connecting to a Remote Console

The ’CONNECT’ command is used to request a console connection between the client and the node. If a RO (read only) request is made, it will always be granted as long as the node is configured with a proper device/address. If a RW (read/write) request is made, the request will only be granted if no other clients have requested a RW connection. The ’LIST_CONNECTIONS’ and ’FORCE_DISCONNECT’ commands may be used to terminate an existing RW connection to allow a new writer. If a connection is granted, the module will return a TCP/IP address, port number, and ASCII one time use password for the client to connect to for console access. The one time password is used to ensure authorized access to the console. This connection will remain open unless the following events occur:

- The client -> TCP console connection is closed
- The requesting client issues a ’DISCONNECT’ command
- Any client issues a ’FORCE_DISCONNECT’ command
- The requesting client -> Nexxus connection is closed

Once the ’CONNECT’ job has been completed, connect to the console by opening a TCP connection to the specified IP address and port. The *telnet* tool can be used for this purpose, however the *sercon_terminal* tool is better suited for connecting to remote consoles with the SERCON module.

```
FORMAT:
SERCON:CONNECT:<NODE_ID>:<RO/RW>
RESPONSES:
JOB_STARTED
CONNECT:<FD>:<IP_ADDRESS>:<PORT>:<PASSWORD>
JOB_ERROR:(errno string)
JOB_COMPLETED
```

## 5.5.6. Disconnecting from a Remote Console

A client may disconnect it's console connection cleanly by issuing the 'DISCONNECT' command. This command instructs the SERCON module to close it's TCP connection to the client. The 'DISCONNECT' command can only be used to disconnect sessions that were 'CONNECT'ed from the same client. To disconnect an arbitrary session, use the 'FORCE_DISCONNECT' command.

*FORMAT:*
SERCON:DISCONNECT:<NODE_ID>:<FD>
*RESPONSES:*
JOB_STARTED
JOB_ERROR:(errno string)
JOB_COMPLETED

## 5.5.7. Listing Current Connections on a Remote Console

A list of connections to a current node's console is obtained with the 'LIST_CONNECTIONS' command. This command returns the client FD handle, IP address, connection time, and access mode for each client connected. The FD handle can be used for future 'FORCE_DISCONNECTION' command. The IP_ADDRESS field may be 0.0.0.0 if the connection has not yet been established with the client. The connection time is specified in HH MM SS.

*FORMAT:*
SERCON:LIST_CONNECTIONS:<NODE_ID>
*RESPONSES:*
JOB_STARTED
CONNECTION:<FD>:<IP_ADDRESS>:<CONNECT TIME>:<RO/RW>
JOB_ERROR:(errno string)
JOB_COMPLETED

## 5.5.8. Forcing Disconnection of a Console Connection

A connection can be force disconnected with the ’FORCE_DISCONNECT’ command.

*FORMAT:*
SERCON:FORCE_DISCONNECT:<NODE_ID>:<FD>
*RESPONSES:*
JOB_STARTED
JOB_ERROR:(errno string)
JOB_COMPLETED

## 5.5.9. Stealing Write Mode from Another Idle Console

Write mode may be ’stolen’ from another console in read/write mode if the target console has been idle for more than five minutes.

*FORMAT:*
SERCON:STEAL_CONNECTION:<NODE_ID>:<FD_RO>:<FD_RW>
*RESPONSES:*
JOB_STARTED
JOB_ERROR:(errno string)
JOB_COMPLETED

## 5.5.10. Adding a Console Alert for a Node

Console alerts are background monitors which the sercon module uses to identify events which the user wishes to keep track of. To add a console alert, use the

'ALERT_ADD' command, passing in the number of lines to capture and a trigger string. The trigger string may contain any printable character.

*FORMAT:*
```
SERCON:ALERT_ADD:<NODE_ID>:<NUM_LINES_TO_CAPTURE>:<TRIGGER_STRING>
```
*RESPONSES:*
```
JOB_STARTED
JOB_ERROR:(errno string)
JOB_COMPLETED
```

## 5.5.11. Deleting a Console Alert for a Node

To delete a previously configured console aert, use the 'ALERT_DEL' command. This command requires the *alert ID*, which can be obtained with the 'ALERT_LIST' command.

*FORMAT:*
```
SERCON:ALERT_DEL:<NODE_ID>:<ALERT_ID>
```
*RESPONSES:*
```
JOB_STARTED
JOB_ERROR:(errno string)
JOB_COMPLETED
```

## 5.5.12. Listing Console Alerts for a Node

A list of all currently configured console alerts for a node is obtained with the 'ALERT_LIST' command.

*FORMAT:*
```
SERCON:ALERT_LIST:<NODE_ID>
```

*RESPONSES:*
JOB_STARTED
ALERT:<ALERT_ID>:<NUM_LINES_TO_CAPTURE>:<TRIGGER_STRING>
JOB_ERROR:(errno string)
JOB_COMPLETED

## 5.5.13. Reading Console Alert Logs for a Node

Console alert logs containing any triggered alerts are downloaded from the module with the 'ALERT_DUMP' command

*FORMAT:*
SERCON:ALERT_DUMP:<NODE_ID>
*RESPONSES:*
JOB_STARTED
ALERT:<TRIGGER_STRING>:<CAPTURE_LINE>:<MAX_LINES>:<CAPTURED_LINE>
JOB_ERROR:(errno string)
JOB_COMPLETED

## 5.5.14. Clearing Console Alert Logs for a Node

The console alert log is cleared for a node with the 'ALERT_CLEAR' command.

*FORMAT:*
SERCON:ALERT_CLEAR:<NODE_ID>
*RESPONSES:*
JOB_STARTED
JOB_ERROR:(errno string)
JOB_COMPLETED

### 5.5.15. SERCON Module Node Global Variable Requirements

The SERCON module has no node global variable requirements.

## 5.6. SYSSTAT Module

The SYSSTAT module is used to obtain run-time information about the remote operating system running on a node. The module communicates with a remote agent daemon called *vacm_sys_statd* to obtain the remote information.

### 5.6.1. Module Features

The SYSSTAT module provides the end user or client with the following information:

- Memory statistics and information
- CPU load
- System uptime
- Mounted filesystem list and usage information
- Online users
- Process listing
- Kernel version
- APM status
- Interrupt allocation list
- I/O port allocation list
- DMA channel allocation list

- Swap file/partition information

The module also provides the ability to obtain information via the vacm_sys_stat_proxy in the event that the remote system is behind a firewall. Traffic between the module and the agent daemon can be encrypted using OpenSSL is VACM was compiled with OpenSSL support.

## 5.6.2. Installing and Setting Up the SYSSTAT Agent Daemon

The agent daemon is part of the *vacm_node_exports* package and must be installed on any system that is to be monitored via SYSSTAT. If secure communication between the module and the agent is desired, VACM should be compiled with OpenSSL support, and OpenSSL should be installed on each remote system to be monitored. (See 'Encryption and Security Considerations for VACM'.) The SSL certificate 'pem' file should be installed as `/etc/vacm_sys_statd.pem`.

To install the agent, simply transfer and install the *vacm_node_exports* package on every remote system you wish to monitor. Then add the agent daemon into your system startup scripts. Once installed, a Nexxus authentication password must be assigned to the daemon to ensure that only authorized Node Controllers may connect to the daemon. To initialize the Nexxus authentication password, simply run the daemon manually. You will be prompted for the password which will be used to authenticate remote connections. This password is stored in `/etc/vacm_sys_statd.passwd` and may be copied to the remaining nodes you wish to monitor if you wish to have the same Nexxus authentication password for every node.

Once the password is set, you can restart the daemon to begin normal operation.

## 5.6.3. Configuring the vacm_sys_stat_proxy for a node

If a remote node is behind a firewall or non-routable network, the module may not be able to communicate with the remote agent daemon. In this case the

`vacm_sys_stat_proxy` is used to proxy connections from the module to the agent. The proxy server is located in the *vacm_node_exports* package. If secure communication is being used, then OpenSSL must be installed on the firewall or proxy machine. The SSL certificate 'pem' file should be installed as `/etc/vacm_sys_stat_proxy.pem`. Add the proxy server into your startup scripts, and run it to set the Nexxus proxy authentication password. Once set, simply re-run the proxy server for normal operation.

## 5.6.4. Configuring the SYSSTAT Module To Manage a Node

Before a node can be monitored with SYSSTAT, the module needs to know the Nexxus authentication password for a node. This password is used to ensure that only authorized Nexxus connections are accepted by a remote agent daemon. The address of the remote machine is pulled out from the 'IP_ADDRESS' node global variable. The maximum password length is 40 characters and must contain only printable characters.

```
FORMAT:
SYSSTAT:CONFIGURATION:<NODE_ID>:<PASSWORD>
RESPONSES:
JOB_STARTED
JOB_ERROR:PASSWORD_TOO_LONG   -- Password exceeds 40 characters
JOB_ERROR:BAD_PASSWORD        -- Password contains non printable
                                 characters
JOB_COMPLETED
```

## 5.6.5. Configuring a node to be monitored via SYSTAT proxy

A remote proxy server is specified for a node with the 'SET_PROXY' command. The

proxy is unset by specifying 'NONE' for <PROXY_ADDR>.

```
FORMAT:
SYSSTAT:SET_PROXY:<NODE_ID>:<PROXY_ADDR>:<PROXY_PASSWORD>
RESPONSES:
JOB_STARTED
JOB_ERROR:PASSWORD_TOO_LONG   -- Password exceeds 40 characters
JOB_ERROR:BAD_PASSWORD        -- Password contains non printable
                                 characters
JOB_ERROR:BAD_ADDRESS         -- Invalid address specified
JOB_COMPLETED
```

## 5.6.6. Obtaining Node Memory Statistics

To return realtime memory statistics on a remote node, use the 'MEM' command:

```
FORMAT:
SYSSTAT:MEM:<NODE_ID>
RESPONSES:
JOB_STARTED
MEM:<T_MEM>:<F_MEM>:<S_MEM>:<BUF>:<CACHE>:<T_SWAP>:<F_SWAP>
JOB_COMPLETED
FIELDS:
<T_MEM>  - Total System Memory
<F_MEM>  - Free System Memory
<S_MEM>  - Shared System Memory
<BUF>    - Buffers
<CACHE>  - Cache
<T_SWAP> - Total Swap
<F_SWAP> - Free Swap
```

## 5.6.7. Obtaining Node CPU Load

Current CPU load statistics are obtained with the ’CPU_LOAD’ command:

*FORMAT:*
SYSSTAT:CPU_LOAD:<NODE_ID>
*RESPONSES:*
JOB_STARTED
CPU_LOAD:<1_MIN_AVG>:<5_MIN_AVG>:<15_MIN_AVG>
JOB_COMPLETED

## 5.6.8. Obtaining Node Uptime

The ’UPTIME’ command returns the number of seconds the remote system has been up:

*FORMAT:*
SYSSTAT:UPTIME:<NODE_ID>
*RESPONSES:*
JOB_STARTED
UPTIME:<SECONDS>
JOB_COMPLETED

## 5.6.9. Obtaining Node Mounted Filesystem Information

The ’FS’ command returns a list of mounted filesystems on a remote node and their disk space usage:

*FORMAT:*
SYSSTAT:FS:<NODE_ID>
*RESPONSES:*

```
JOB_STARTED
FS:<DEVICE_NAME>:<TOTAL_BLK>:<USED_BLK>:<FREE_BLK>:<PERCENT_USED>:<MNT_PT>
JOB_COMPLETED
```

## 5.6.10. Obtaining a List of Users Online a Node

The ’WHO’ command returns a list of users currently logged into a node:

```
FORMAT:
SYSSTAT:WHO:<NODE_ID>
RESPONSES:
JOB_STARTED
FS:<USER_NAME>:<TTY>
JOB_COMPLETED
```

## 5.6.11. Obtaining a List of Processes Running On a Node

Process listings are retrieved from a remote node with the ’PS’ command:

```
FORMAT:
SYSSTAT:PS:<NODE_ID>
RESPONSES:
JOB_STARTED
PS:<PID>:<USERNAME>:<CPU %>:<MEM %>:<RSS>:<STATE>:<TTY>:<COMMAND>
JOB_COMPLETED
FIELDS:
<PID> - Process ID
<USERNAME> - Username running process
<CPU %>    - Percentage of CPU currently in use
<MEM %>    - Percentage of memory currently in use
<RSS>      - Resident set size of process
```

```
<STATE>    - Current state of process
<TTY>      - TTY associated with process
<COMMAND>  - Command name
```

## 5.6.12. Obtaining a Nodes Kernel Version

The current kernel version of a remote node is obtained with the ’VERSION’ command:

```
FORMAT:
SYSSTAT:VERSION:<NODE_ID>
RESPONSES:
JOB_STARTED
VERSION:<KERNEL_VERSION>
JOB_COMPLETED
```

## 5.6.13. Obtaining a Nodes APM Status

The current advanced power management status of a node is retrieved with the ’APM’ command.

```
FORMAT:
SYSSTAT:APM:<NODE_ID>
RESPONSES:
JOB_STARTED
APM:<LINE_STATUS>:<BATT_STATUS>:<PERC_CAPACITY>:<UNITS_REMAINING>:
    <MEASURE>
JOB_COMPLETED
FIELDS:
<LINE_STATUS>     - AC line status
```

```
<BATT_STATUS>     - Battery status
<PERC_CAPACITY>   - Percentage battery remaining
<UNITS_REMAINING> - Number of 'units' of battery
                    capacity remaining
<MEASURE>         - Measure of units'
```

## 5.6.14. Obtaining an Interrupt Allocation List for a Node

A hardware interrupt map of a remote node is obtained with the 'INTERRUPTS' command:

```
FORMAT:
SYSSTAT:INTERRUPTS:<NODE_ID>
RESPONSES:
JOB_STARTED
INT:<INT_NO>:<INT_SRC>:<INT_NAME>:<CNT_CPU0>:<CNT_CPU1>:<CNT_CPU2>:
    <CNT_CPU3>
JOB_COMPLETED
FIELDS:
<INT_NO>   - Interrupt number
<INT_SRC>  - Interrupt source
<INT_NAME> - Interrupt name
<CNT_CPU0> - Interrupt count for CPU 0
<CNT_CPU1> - Interrupt count for CPU 1 (may be 'NONE')
<CNT_CPU2> - Interrupt count for CPU 2 (may be 'NONE')
<CNT_CPU3> - Interrupt count for CPU 3 (may be 'NONE')
```

## 5.6.15. Obtaining an I/O Port Allocation List for a Node

An I/O port map for a node is retrieved with the ’IOPORTS’ command:

*FORMAT:*
SYSSTAT:IOPORTS:<NODE_ID>
*RESPONSES:*
JOB_STARTED
IO:<ADDRESS_RANGE>:<FUNCTION>
JOB_COMPLETED

## 5.6.16. Obtaining a DMA Channel Allocation List for a Node

The DMA channel allocation list is retrieved with the ’DMA’ command:

*FORMAT:*
SYSSTAT:DMA:<NODE_ID>
*RESPONSES:*
JOB_STARTED
DMA:<CHANNEL>:<FUNCTION>
JOB_COMPLETED

## 5.6.17. Obtaining Swapfile Statistics for a Node

Detailed swapfile statistics for a node are obtained withthe ’SWAP’ command:

*FORMAT:*
SYSSTAT:SWAP:<NODE_ID>
*RESPONSES:*

```
JOB_STARTED
SWAP:<SWAP_FILE>:<SWAP_TYPE>:<SWAP_SIZE>:<SWAP_USED>:<SWAP_PRI>
JOB_COMPLETED
FIELDS:
<SWAP_FILE> - Swap filename
<SWAP_TYPE> - Swap type (either 'partition' or 'file')
<SWAP_SIZE> - Swap file/partition size
<SWAP_USED> - Swap used
<SWAP_PRI>  - Swap priority
```

### 5.6.18. SYSSTAT Module Node Global Variable Requirements

The SYSSTAT module requires that the *IP_ADDRESS* variable be set to the internet address of a node. This address is used to connect to the remote `vacm_sys_statd`.

## 5.7. USER_ADM Module

The USER_ADM module allows secure remote user administration on a single or group of nodes. The module communicates with a remote agent daemon called *vacm_user_admd* to perform the remote operations. This module *requires* that OpenSSL be built into your VACM installation, otherwise the module will not be installed. This is purely for security reasons.

### 5.7.1. Module Features

The USER_ADM module provides the end user or client with the following

administrative capabilities:

- User account creation and removal
- User account group manipulation
- Group creation and manipulation
- User home directory modification
- User group, shell, uid modification
- User account expiry and inactive day count modification
- User comment and password modification
- User account locking and unlocking

## 5.7.2. Installing and Setting Up the USER_ADM Agent Daemon

The agent daemon is part of the *vacm_node_exports* package and must be installed on any system which is to be user account managed via USER_ADM. In order for USER_ADM to be used, OpenSSL must be installed on the node controller, as well as each system the agent is to run on (See 'Encryption and Security Considerations for VACM'.) The SSL certificate 'pem' file should be installed as `/etc/vacm_user_admd.pem`.

To install the agent, simply transfer and install the *vacm_node_exports* package on every remote system you wish to monitor. Then add the agent daemon into your system startup scripts. Once installed, a Nexxus authentication password must be assigned to the daemon to ensure that only authorized Node Controllers may connect to the daemon. To initialize the Nexxus authentication password, simply run the daemon manually. You will be prompted for the password which will be used to authenticate remote connections. This password is stored in `/etc/vacm_user_admd.passwd` and

may be copied to the remaining nodes you wish to monitor if you wish to have the same Nexxus authentication password for every node.

Once the password is set, you can restart the daemon to begin normal operation.

## 5.7.3. PAM Considerations with USERADM

If your remote systems use PAM authentication, you will have to specify the configuration option `--enable-pam` to the `autogen.sh` script when building VACM.

## 5.7.4. Configuring the USERADM Module To Manage a Node

Before a node can be administered with USER_ADM, the module needs to know the Nexxus authentication password for a node. This password is used to ensure that only authorized Nexxus connections are accepted by a remote agent daemon. The address of the remote machine is pulled out from the 'IP_ADDRESS' node global variable. The maximum password length is 40 characters and must contain only printable characters.

```
FORMAT:
USER_ADM:CONFIGURATION:<NODE_ID>:<PASSWORD>
RESPONSES:
JOB_STARTED
JOB_ERROR:PASSWORD_TOO_LONG   -- Password exceeds 40 characters
JOB_ERROR:BAD_PASSWORD        -- Password contains non printable
                                 characters
JOB_COMPLETED
```

## 5.7.5. Adding a User to a Remote Node

To add a user to a remote node, use the ’USER_ADD’ command:

```
FORMAT:
USER_ADM:USER_ADD:<NODE_ID>:<NAME>:<UID>:<GROUP>:<HOME>:<SHELL>:<PASSWD>:<EX
RESPONSES:
JOB_STARTED
JOB_ERROR:UID_IN_USE       -- User number (UID) is already in use
JOB_ERROR:NAME_IN_USE      -- User name is already in use
JOB_ERROR:BAD_HDIR_PATH    -- Unable to create home directory
JOB_ERROR:BAD_SHELL_PATH   -- Shell does not exist
JOB_ERROR:COMMENT_TOO_LONG -- Comment length too long
JOB_ERROR:BAD_COMMENT      -- Comment contains bad characters
JOB_COMPLETED
FIELDS:
<NAME>     -- User name to add
<UID>      -- UID to assign
              (may be ’DEFAULT’ to have one assigned)
<GROUP>    -- Primary group number / name (may be ’DEFAULT’)
<HOME>     -- Home directory (may be ’DEFAULT’)
<SHELL>    -- Default shell (may be ’DEFAULT’)
<PASSWD>   -- Password (may be ’NONE’)
<EXPIRE>   -- Expire Date (YYYY-MM-DD, ’NONE’ for no expiry)
<INACTIVE> -- Inactive days (’NONE’ to disable, 0 for immediate)
<COMMENT>  -- Comment / Gecos field
              (may be ’NONE’, max 40 chars)
NOTES:
If the primary group is ’DEFAULT’, then a new primary group with the name of
ated.
```

## 5.7.6. Removing a User From a Remote Node

To remove a user account from a remote system, use the ’USER_REMOVE’ command. The users home directory is deleted if the ’DELETE_DIR_FLAG’ argument is set to ’YES’.

```
FORMAT:
USER_ADM:USER_REMOVE:<NODE_ID>:<USER_NAME>:<DELETE_DIR_FLAG>
RESPONSES:
JOB_STARTED
JOB_ERROR:USER_NOT_FOUND   -- User does not exist on system
JOB_ERROR:PERMISSION_DENID -- ROOT user delete prohibited
JOB_COMPLETED
NOTES:
You cannot delete the ROOT user on a node.
```

## 5.7.7. Adding a User to a Group

A group is added to the list of groups to which a user is a member of with the ’USER_GRP_ADD’ command:

```
FORMAT:
USER_ADM:USER_GRP_ADD:<NODE_ID>:<USER_NAME>:<GROUP_NAME>
RESPONSES:
JOB_STARTED
JOB_ERROR:USER_NOT_FOUND -- User does not exist on system
JOB_ERROR:ALREADY_MEMBER -- User is already a member of
                            the specified group
JOB_COMPLETED
```

## 5.7.8. Removing a User From a Group

A group is removed from a group with the ’USER_GRP_REM’ command:

```
FORMAT:
USER_ADM:USER_GRP_REM:<NODE_ID>:<USER_NAME>:<GROUP_NAME>
RESPONSES:
JOB_STARTED
JOB_ERROR:USER_NOT_FOUND -- User does not exist on system
JOB_ERROR:NOT_MEMBER     -- User is not a member of the
                            specified group
JOB_ERROR:PRIMARY_GROUP  -- Specified group is the users
                            primary group
JOB_COMPLETED
NOTES:
You cannot remove a user from it’s primary group. To modify a users pri-
mary group, use the ’USER_CHG_GROUP’ command.
```

## 5.7.9. Listing Groups Which a User is a Member of

A list of the groups which a user is a member of is obtained with the ’USER_GRP_LIST’ command:

```
FORMAT:
USER_ADM:USER_GRP_LIST:<NODE_ID>:<USER_NAME>
RESPONSES:
JOB_STARTED
USER_GRP_LIST:<GROUP_NAME>:<GROUP_ID>
JOB_ERROR:USER_NOT_FOUND -- User does not exist on system
JOB_COMPLETED
```

## 5.7.10. Changing a Users Primary Group

The primary group to which a user belongs is modified with the ’USER_CHG_GRP’ command:

```
FORMAT:
USER_ADM:USER_CHG_GRP:<NODE_ID>:<USER_NAME>:<GROUP_NAME>
RESPONSES:
JOB_STARTED
JOB_ERROR:USER_NOT_FOUND  -- User does not exist on system
JOB_ERROR:ALREADY_PRIMARY -- Group specified is already the
                             users primary group
JOB_COMPLETED
```

## 5.7.11. Changing a Users Home Directory

The default home directory for a user is modified with the ’USER_CHG_DIR’ command:

```
FORMAT:
USER_ADM:USER_CHG_DIR:<NODE_ID>:<USER_NAME>:<HOME_DIR>
RESPONSES:
JOB_STARTED
JOB_ERROR:USER_NOT_FOUND    -- User does not exist on system
JOB_ERROR:BAD_HDIR_PATH     -- Bad pathname
JOB_ERROR:DIR_ERROR:<REASON>-- Unable to create directory
JOB_COMPLETED
```

## 5.7.12. Changing a Users Default Shell

The users default shell is modified with the 'USER_CHG_SHELL' command:

```
FORMAT:
USER_ADM:USER_CHG_SHELL:<NODE_ID>:<USER_NAME>:<SHELL_PATH>
RESPONSES:
JOB_STARTED
JOB_ERROR:USER_NOT_FOUND  -- User does not exist on system
JOB_ERROR:BAD_SHELL_PATH  -- Bad pathname
JOB_COMPLETED
```

## 5.7.13. Changing a Users UID

A users unique UID is modified with the 'USER_CHG_UID' command. Once issued, all files in the users home directory are changed to reflect the new UID

```
FORMAT:
USER_ADM:USER_CHG_UID:<NODE_ID>:<USER_NAME>:<UID>
RESPONSES:
JOB_STARTED
JOB_ERROR:USER_NOT_FOUND         -- User does not exist on system
JOB_ERROR:PERMISSION_DENIED      -- Invalid UID request
JOB_ERROR:UID_IN_USE:<USER_NAME> -- UID in use by <USER_NAME>
USER_CHG_UID:IN_PROGRESS         -- File ownership being updated
USER_CHG_UID:PROCESS_FAILURE     -- Child process failure
JOB_COMPLETED
```

## 5.7.14. Changing a Users Account Expiry

The users expiry date is modified with the ’USER_CHG_EXP’ command:

*FORMAT:*
```
USER_ADM:USER_CHG_EXP:<NODE_ID>:<USER_NAME>:<YYY-MM-DD>
```
*RESPONSES:*
```
JOB_STARTED
JOB_ERROR:USER_NOT_FOUND  -- User does not exist on system
JOB_COMPLETED
```

## 5.7.15. Changing a Users Inactive Account Timer

A users inactive account timer is modified with the ’USER_CHG_EXP’ command. The inactive account timer determines how many days after a password expiry the account will remain available until it is permanently disabled (until admin intervention).

*FORMAT:*
```
USER_ADM:USER_CHG_INA:<NODE_ID>:<USER_NAME>:<INACTIVE_DAYS>
```
*RESPONSES:*
```
JOB_STARTED
JOB_ERROR:USER_NOT_FOUND  -- User does not exist on system
JOB_ERROR:BAD_VALUE       -- Bad value specified
JOB_COMPLETED
```

## 5.7.16. Changing a Users Comment

The users comment / Gecos field is modified with the ’USER_CHG_COMMENT’ command:

*FORMAT:*

```
USER_ADM:USER_CHG_COMMENT:<NODE_ID>:<USER_NAME>:<COMMENT>
RESPONSES:
JOB_STARTED
JOB_ERROR:USER_NOT_FOUND   -- User does not exist on system
JOB_ERROR:COMMENT_TOO_LONG -- Comment exceeds 40 characters
JOB_COMPLETED
```

## 5.7.17. Changing a Users Password

If PAM has been compiled into USER_ADM, it is possible to change a users password with the ’USER_CHG_PASSWD’ command:

```
FORMAT:
USER_ADM:USER_CHG_PASSWD:<NODE_ID>:<USER_NAME>:<OLD>:<NEW>
RESPONSES:
JOB_STARTED
JOB_ERROR:USER_NOT_FOUND   -- User does not exist on system
JOB_ERROR:BAD_PASSWORD     -- Bad <NEW> password specified
JOB_ERROR:PERMISSION_DENIED-- Bad <OLD> password specified
JOB_ERROR:NO_PAM_SUPPORT   -- No PAM support on node
JOB_ERROR:PAM_ERR:<REASON> -- PAM specific error
JOB_COMPLETED
```

## 5.7.18. Listing All Users on a Node

A list of all user accounts on a node is retrieved with the ’USER_LIST’ command:

```
FORMAT:
USER_ADM:USER_LIST:<NODE_ID>
RESPONSES:
JOB_STARTED
```

```
USER_LIST:<NAME>:<UID>:<GID>/<GROUP_NAME>:<HDIR>:<SHELL>:<EXP>:
          <INACTIVE>:<COMMENT>
JOB_COMPLETED
```

## 5.7.19. Listing All Groups on a Node

A list of all groups existing a node is obtained with the ’GROUP_LIST’ command:

*FORMAT:*
```
USER_ADM:GROUP_LIST:<NODE_ID>
```
*RESPONSES:*
```
JOB_STARTED
GROUP_LIST:<GROUP_NAME>:<GROUP_ID>
JOB_COMPLETED
```

## 5.7.20. Locking a User Account on a Node

Locking a user account with the ’LOCK_USER’ command temporarily disables the user account so that the user cannot log in.

*FORMAT:*
```
USER_ADM:LOCK_USER:<NODE_ID>:<USER_NAME>
```
*RESPONSES:*
```
JOB_STARTED
JOB_ERROR:USER_NOT_FOUND -- User does not exist on system
JOB_ERROR:ALREADY_LOCKED -- User account is already locked
JOB_COMPLETED
```

### 5.7.21. Unlocking a User Account on a Node

Unlocking a user account with the ’UNLOCK_USER’ command, restores a user account which has been previously locked with ’LOCK_USER.

*FORMAT:*
`USER_ADM:UNLOCK_USER:<NODE_ID>:<USER_NAME>`
*RESPONSES:*

```
JOB_STARTED
JOB_ERROR:USER_NOT_FOUND   -- User does not exist on system
JOB_ERROR:ALREADY_UNLOCKED -- User account is already unlocked
JOB_COMPLETED
```

### 5.7.22. USER_ADM Module Node Global Variable Requirements

The USER_ADM module requires that the *IP_ADDRESS* variable be set to the internet address of a node. This address is used to connect to the remote `vacm_user_admd`.

## 5.8. RSH Module

The RSH module is used to execute commands on a remote host. The module implements a set of available commands. It does not allow the user to execute arbitrary commands on a remote host.

## 5.8.1. Module Features

The RSH module provides the end user or client with the following capabilities:

- Node Inventory
- CPU load average
- Online users
- Memory usage
- Process Listing
- Retrieve remote syslog
- Remote OS level shutdown and Reboot

## 5.8.2. Configuring the RSH Module To Manage a Node

The CONFIGURATION command is used to configure the agent, and remote user for RSH to use during connect.

```
FORMAT:
SERCON:CONFIGURATION:<NODE_ID>:<RSH_AGENT>:<RSH_USER>
RESPONSES:
JOB_STARTED
JOB_ERROR:(errno string)
JOB_COMPLETED
FIELDS:
<RSH_AGENT> - The RSH agent is the transport agent used for the RSH
module to execute commands on a remote host.  Currently the only transport
supported is RSH.
<RSH_USER> - The RSH user is the username to use to connect to the
remote node.  This should usually be root.
```

RSH takes the address of the remote node from the global variable IP_ADDRESS.

## 5.8.3. Obtaining Node Inventory

*FORMAT:*
RSH:NODE_INV_INFO:<NODE_ID>
*RESPONSES:*
JOB_STARTED
NODE_INV_INFO:<OS_NAME>:<KERNEL_VERSION>
JOB_COMPLETED

## 5.8.4. Obtaining Load Average

*FORMAT:*
RSH:LOAD_AVG:<NODE_ID>
*RESPONSES:*
JOB_STARTED
LOAD_AVG:<LOAD_ONE>:<LOAD_FIVE>:<LOAD_FIFTEEN>
JOB_COMPLETED

## 5.8.5. Obtaining Online User Listing

*FORMAT:*
RSH:WHO:<NODE_ID>
*RESPONSES:*
JOB_STARTED
WHO:<USERNAME>:<TTY>

```
JOB_COMPLETED
```

## 5.8.6. Obtaining Memory Usage

```
FORMAT:
RSH:MEM:<NODE_ID>
RESPONSES:
JOB_STARTED
MEM:<T_MEM>:<F_MEM>:<S_MEM>:<BUFF>:<CACHE>:<T_SWAP>:<F_SWAP>
JOB_COMPLETED
FIELDS:
<T_MEM>  - Total System Memory
<F_MEM>  - Free System Memory
<S_MEM>  - Shared System Memory
<BUFF>   - Buffers
<CACHE>  - Cache
<T_SWAP> - Total Swap
<F_SWAP> - Free Swap
```

## 5.8.7. Obtaining Process Listing

```
FORMAT:
RSH:PS:<NODE_ID>
RESPONSES:
JOB_STARTED
PS:<PID>:<USER>:<CPU>:<MEM>:<RSS>:<STATE>:<TTY>:<CMD>
JOB_COMPLETED
FIELDS:
<PID>  - Process ID
```

```
<USER>  - Username running process
<CPU>  - Percentage of CPU currently in use
<MEM>   - Percentage of memory currently in use
<RSS>  - Resident set size of process
<STATE> - Current state of process
<TTY> - TTY associated with process
<CMD> - Command name
```

## 5.8.8. Retrieving Remote Syslog

*FORMAT:*
RSH:DOWNLOAD_LOG:<NODE_ID>
*RESPONSES:*
JOB_STARTED
DOWNLOAD_LOG:<LOG_ENTRY>
JOB_COMPLETED

## 5.8.9. Shutdown a Node

*FORMAT:*
RSH:SHUTDOWN:<NODE_ID>
*RESPONSES:*
JOB_STARTED
JOB_COMPLETED

### 5.8.10. Restart a Node

*FORMAT:*
RSH:RESTART:<NODE_ID>
*RESPONSES:*
JOB_STARTED
JOB_COMPLETED

## 5.9. BAYTECH Module

The BAYTECH module provides serial access to a Baytech Power Strip.

### 5.9.1. Module Features

The BAYTECH module provides the end user or client with the following capabilities:

- Reset the Baytech Unit
- Power on a Port
- Power off a Port
- Reboot a Port

## 5.9.2. Configuring the Baytech Module To Manage a Powerstrip

The CONFIGURATION command is used to set the physical device address of the Baytech Power Strip.

*FORMAT:*
BAYTECH:CONFIGURATION:<NODE_ID>:<DEVICE_ADDRESS>
*RESPONSES:*
JOB_STARTED
JOB_ERROR:(errno string)
JOB_COMPLETED
*FIELDS:*
<DEVICE_ADDRESS> - Either a serial port or the address of a port on a terminal server in the form of hostname-port

## 5.9.3. Resetting a Baytech Unit

*FORMAT:*
BAYTECH:RESET:<NODE_ID>
*RESPONSES:*
JOB_STARTED
JOB_COMPLETED

## 5.9.4. Powering on a Port

*FORMAT:*
BAYTECH:PORT_ON:<NODE_ID>:<PORT_ID>

*RESPONSES:*
JOB_STARTED
JOB_COMPLETED

### 5.9.5. Powering off a Port

*FORMAT:*
BAYTECH:PORT_OFF:<NODE_ID>:<PORT_ID>
*RESPONSES:*
JOB_STARTED
JOB_COMPLETED

### 5.9.6. Rebooting a Port

*FORMAT:*
BAYTECH:PORT_REBOOT:<NODE_ID>:<PORT_ID>
*RESPONSES:*
JOB_STARTED
JOB_COMPLETED

## 5.10. SBT2 Module

The SBT2 module provides hardware control for Intel SBT2 based hardware.

## 5.10.1. Module Features

The SBT2 module provides the end user or client with the following capabilities:

- Reset the Unit
- Power on or Off the Unit
- Chassis Status

## 5.10.2. Configuring the SBT2 Module To Manage a Node

The CONFIGURATION command is used to set the physical device address of the managed node.

```
FORMAT:
SBT2:CONFIGURATION:<NODE_ID>:<DEVICE_ADDRESS>
RESPONSES:
JOB_STARTED
JOB_ERROR:(errno string)
JOB_COMPLETED
FIELDS:
<DEVICE_ADDRESS> - Either a serial port or the address of a port on
a terminal server in the form of hostname-port
```

## 5.10.3. Resetting a Unit

```
FORMAT:
SBT2:HARD_RESET:<NODE_ID>
RESPONSES:
```

```
JOB_STARTED
JOB_ERROR:(errno string)
JOB_COMPLETED
```

## 5.10.4. Powering on a Unit

*FORMAT:*
SBT2:POWER_ON:<NODE_ID>
*RESPONSES:*
JOB_STARTED
JOB_ERROR:(errno string)
JOB_COMPLETED

## 5.10.5. Powering off a Unit

*FORMAT:*
SBT2:POWER_OFF:<NODE_ID>
*RESPONSES:*
JOB_STARTED
JOB_ERROR:(errno string)
JOB_COMPLETED

## 5.10.6. Refreshing the connection to a Unit

*FORMAT:*

```
SBT2:REFRESH:<NODE_ID>
RESPONSES:
JOB_STARTED
JOB_ERROR:(errno string)
JOB_COMPLETED
```

### 5.10.7. Retrieving the chassis status of a Unit

```
FORMAT:
SBT2:CHASSIS_STATUS:<NODE_ID>
RESPONSES:
JOB_STARTED
JOB_ERROR:(errno string)
CHASSIS_STATUS:<POWER_STATE>
JOB_COMPLETED
FIELDS:
<POWER_STATE> - This field can be one of the following:
WORKING|SUSPEND_TO_RAM|SUSPEND_TO_DISK|
SOFT_OFF|MECHANICAL_OFF|SLEEPING|LEGACY_OFF
```

## 5.11. QUANTA Module

The Quanta module provides hardware control for Quanta based hardware.

## 5.11.1. Module Features

The Quanta module provides the end user or client with the following capabilities:

- Reset the Unit
- Power on or Off the Unit
- Chassis Status

## 5.11.2. Configuring the Quanta Module To Manage a Node

The CONFIGURATION command is used to set the physical device address of the managed node.

```
FORMAT:
QUANTA:CONFIGURATION:<NODE_ID>:<DEVICE_ADDRESS>
RESPONSES:
JOB_STARTED
JOB_ERROR:(errno string)
JOB_COMPLETED
FIELDS:
<DEVICE_ADDRESS> - Either a serial port or the address of a port on
a terminal server in the form of hostname-port
```

## 5.11.3. Resetting a Unit

```
FORMAT:
QUANTA:HARD_RESET:<NODE_ID>
```

*RESPONSES:*
JOB_STARTED
JOB_ERROR:(errno string)
JOB_COMPLETED

## 5.11.4. Powering on a Unit

*FORMAT:*
QUANTA:POWER_ON:<NODE_ID>
*RESPONSES:*
JOB_STARTED
JOB_ERROR:(errno string)
JOB_COMPLETED

## 5.11.5. Powering off a Unit

*FORMAT:*
QUANTA:POWER_OFF:<NODE_ID>
*RESPONSES:*
JOB_STARTED
JOB_ERROR:(errno string)
JOB_COMPLETED

## 5.11.6. Refreshing the connection to a Unit

*FORMAT:*
QUANTA:REFRESH:<NODE_ID>
*RESPONSES:*
JOB_STARTED
JOB_ERROR:(errno string)
JOB_COMPLETED

## 5.11.7. Retrieving the BMC version of a Unit

*FORMAT:*
QUANTA:BMC_INFO:<NODE_ID>
*RESPONSES:*
JOB_STARTED
JOB_ERROR:(errno string)
CHASSIS_STATUS:<BMC_VERSION_STRING>
JOB_COMPLETED

# Chapter 6. VACM Clients

## 6.1. Using VACM with vash

### 6.1.1. Introduction to vash

vash is an excellent tool for simple shell scripts to communicate with VACM or to perform VACM operations directly from the command line. It has both an interactive and non interactive mode of operation. vash takes a number of optional arguments:

The power of vash comes from the ipc command. It allows you to send any ipc message to the module of your choice directly from the command line. vash does not care about the semantics of individual modules and as such requires the user to understand what IPC commands a module accepts.

### 6.1.2. Commandline Options

```
-c <nexxus>   -- Connect to Nexxus
-u <username> -- Username
-p <password> -- Password
-x <command>  -- Execute command in Non Interactive Mode
```

### 6.1.3. vash Internal Commands

vash currently implements the following commands:

**alias <alias>=<command>**

Alias user supplied command "alias" to command "command".

**connect <hostname>**

Connect to a Nexxus at "hostname".

**disconnect <hostname>**

Disconnect from the Nexxus at "hostname".

**exit**

Exit vash.

**ipc <nexxus> <message>**

Send ipc message "message" to Nexxus "nexxus.

**nexxus_modules <nexxus>**

List loaded modules on Nexxus "nexxus".

**source <filename>**

Source and execute commands from "filename".

**unalias <alias>**

Remove the alias for "alias".

# 6.2. Using VACM with Flim

## 6.2.1. Introduction to Flim

Flim is a small-scale cluster management graphical front-end authored by Carsten Haitzler <raster@valinux.com> of VA Linux Systems, Inc. It has a modular interface and plugin design to allow plugins for the toolbar and monitor plugin section

to be easily written and slotted in for management and monitoring of small to medium sized clusters (up to about 200 nodes).

## 6.2.2. Using Flim

**Figure 6-1. Flim Screenshot**

To start flim up execute the ‘flim’ command on the command line (or add it to any menu or toolbar on your desktop as the command to execute for that menu entry or button). You should now have a basic interface up with no nexxus or nodes configured.

To find out more information about the currently installed version of Flim that you are now running select about from the help menu on the upper-right corner of the flim window.

**Figure 6-2. About Menu**

This will display a window with information about the version of Flim running.

**Figure 6-3. About Dialog Window**

The first thing you will want to do is to add a nexxus to Flim so it can query what nodes are attached to the nexxus (if any) and display information about them, or let you add and delete nodes. To do this select "New" from the "Nexxus" menu at the top of the flim window.

**Figure 6-4. The Nexxus Menu**

Once you have selected this, a new nexxus will be added and a dialog will appear for you to set the appropriate information for the new nexxus you added.

**Figure 6-5. Nexxus Settings Dialog Window**

The Name field is a convenience name for the administrator to use. Using a recognizable name such as "Top Floor", "Front Machine Room", "Sourceforge" etc. would be a good idea here. The address is the internet address which will be used to contact the nexxus from the client. This may be any machine name or IP address that validly points to that nexxus. The next 2 fields are the login and password to use for that nexxus to gain access. Edit these as appropriate. Remember that henceforth all Flim plugins and flim itself will access this nexxus as that user. If that user has restricted access Flim might not be able to display information or perform management of that node in some areas. When you press "OK" the settings will be applied and all plugins will be informed of the change of settings for that nexxus.

If your nexxus is already set up all the nodes it is talking to will appear in the tree view. you can select the nexxus and all nodes connected to that nexxus by selecting the nexxus in the tree view. clicking on a nexxus or a node will toggle its selection. You can use the "Edit" menu to also select and deselect nodes.

**Figure 6-6. Edit Menu**

If no nodes are attached and configured for the nexxus, or you need to add more, or delete nodes, you can use the "Node" menu. When you select delete it will delete all selected nodes in the tree view.

**Figure 6-7. Node Menu**

Be wary of what you have selected before you delete nodes. If you select "Settings", the settings on selected nodes can be changed, and if you select Add a new node will be added, allowing you to set its default settings.

When configuring a new node, or changing the settings on an existing node, the following entries are significant.

**Figure 6-8. Node Settings Dialog**

The first entry is the name of the node. This is the same name the nexxus uses to identify the node. The network address would be the IP address of the node in question for VACM to address it for other reasons such as for the sensor daemon module and systat module. Leave this blank if it is not relevant. The device entry next is the device by which vacm will contact the node for emp. This is normally a serial device of some sort. The password is the password to be used to control access to that node’s vacm settings. Leave it blank or "NONE" for no password. The asset tag can be up to 14 characters to uniquely identify private information about the machine.

Flim operates on a plugin mechanism. The toolbar on the left and pane on the right get populated with plugins that are separate processes spawned by Flim. You can configure the plugins you want to run by bringing up Flim’s preferences dialog.

**Figure 6-9. Flim Preferences Dialog**

**Figure 6-10. Flim Preferences Dialog**

Click on the Add button to add a plugin to the list, select and deselect it in the list to enable and disable that plugin.

Flim allows you to group your nodes in virtual groups - for example "Rack1", "Rack2", "Customer A", "Levis", "501s" etc. You can create a new group just by selecting a set of nodes and then selecting "New group from selected nodes" which will bring up a dialog to name the group. You can delete a group by selecting it and selecting "Delete" from the "Group" menu, and you can rename a group by selecting "Settings..." from the "Group" menu.

**Figure 6-11. Flim Groups Menu**

**Figure 6-12. Flim Groups Dialog**

# 6.3. Using VACM with Hoover

## 6.3.1. Using Hoover

Hoover is an graphical user interface under development, authored by Dean Johnson <dtj@sgi.com>. Its interface allows the user to aggregate cluster information, for one or more clusters, in a very compact space and attempts to have only one window if at all possible. The primary display (referred to as a ’panel’) is a tree view of the clusters and their nodes. From that panel, the user can then find information, such as the system event logs, or perform various administrative actions on the nodes (or whole clusters), such as resetting them or powering them down. When information has been requested,

typically a new panel is created in the lower tabbed panel. Panels can be dragged and dropped inside or outside of hoover for flexibility.

**Figure 6-13. Hoover Screenshot**

To run hoover, simply run the ’hoover’ program and specify IP addresses for the machines on which the Nexxus is running. It will default to ’localhost’ if nothing is specified on the command line. Most actions are accomplished by right mouse button selecting on either the cluster entries or the node entries in the trees. You can add or remove nodes using this technique. You may also use the Options->’Scan for Nexxus’ menu item to search an address range for Nexxus. Also, each tabbed panel has a menu available on the tab, using rightmost mouse button that allows for actions on items in that panel.

# Chapter 9. Writing VACM Clients

VACM Clients are the interface between the user and Nexxus. They dispatch the IPC commands to Nexxus and present the responses to the user in either a raw or a cooked format. As VACM installations evolve it is possible to have a highly customized client which reacts to your special set of rules. VACM Clients, like Modules, are stand alone programs. They can be launched at any time by users or by scripts to perform given tasks.

## 9.1. Libvacmclient function prototypes

Clients should be written using the VACM Client API defined below. To use this API we just need to include `vacmclient_api.h` and to link with `libvacmclient.a`.

```
int api_nexxus_connect(char *addr, char *username, char *password, void **new_handle
Connect to the given nexxus with username and password via an AF_INET socket
turns API_RETURN_OK on success, < 0 with vacm_errno set on failure.
```

```
int api_nexxus_disconnect(void *handle);
Disconnect from the nexxus associated with handle.  Returns API_RETURN_OK on
cess, < 0 with vacm_errno set on failure.
```

```
int api_nexxus_send_ipc(void *handle, char *pkt, __uint32_t len);
```

Sends an IPC message of length len to the Nexxus associated with handle.

int **api_nexxus_recv_ipc**(void *_handle_, char **_buffer_, __uint32_t *_len_);
Read an IPC message from the Nexxus associated with handle. The length of t
sage read is stored in len. Returns API_RETURN_OK on success, < 0 with vacm_
ure. It is the callers responsibility to free buffer.

int **api_nexxus_wait_for_data**(void *_handle_, char **_buffer_, __uint32_t *_len_, int _time-
_out_);
Waits for IPC data from the Nexxus associated with handle with a time-
out of 'timeout' seconds. Returns API_RETURN_OK on success, < 0 with vacm_Er
ure. On
success, it is the caller's responsibility to free buffer.

int **api_nexxus_wait_short_for_data**(void *_handle_, char **_buffer_, __uint32_t *_len_, int
_out_);
Identical to api_nexxus_wait_for_data except timeout is a length in
microseconds instead of seconds.

int **api_nexxus_return_fd**(void *_handle_, int _fd_);

Gets the file descriptor of the Nexxus asssociated with handle. Returns API_RETURN_OK on success, < 0 with vacm_errno set on failure.

int **api_nexxus_return_ip**(void **handle*, struct in_addr **ip_addr*);
Get the ip address of the Nexxus associated with handle. Returns API_RETURN
cess, < 0 with vacm_errno set on failure.
-1 on failure with vacm_errno set.

int **api_nexxus_return_handle**(void ***handle*, char **addr*);
Get the handle associated with the nexxus at *addr. Returns API_RETURN_OK o
cess, < 0 with vacm_errno set on failure.

void **api_nexxus_perror**(char **msg*);
Similar to perror(char *), but prints the value of vacm_errno.

char ***api_nexxus_get_error**(void);
Returns the stringified version of vacm_errno.

```
int api_nexxus_ping(struct in_addr *ip_addr, struct timeval *timeout);
Sends a ping message to the remote host specified by 'in_addr' and waits for
a period specified by ´timeout'.
Returns API_RETURN_OK if the remote host has a Nexxus running, API_RETURN_TI
mote host was not running a Nexxus, and API_RETURN_MISC_ERROR on er-
ror.
```

## 9.2. An Example Client

An example client is provided below to demonstrate how to write a VACM client using `libvacmclient.a`.

```
/*
 * This program is original work by Zac Sprackett <zacs@valinux.com> .
 * This program is distributed under the GNU General Public License (GPL)
 * as outlined in the COPYING file.
 *
 * Copyright (C) 2000, Zac Sprackett, and VA Linux Systems, Inc.
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
```

```
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-
1307, USA.
 *
 *
 */

#include <stdio.h>
#include <sys/socket.h>
#include <signal.h>
#include <netinet/in.h>
#include <sys/un.h>
#include <stdlib.h>
#include <sys/time.h>
#include <netdb.h>
#include <fcntl.h>
#include <termios.h>
#include <unistd.h>
#include <vacmclient_api.h>

int compatibility = 0;

/* Function prototypes so we don't generate errors */
void display_usage(void);
int do_uptime(void *handle, char *node);

void display_usage(void)
{
  printf("Usage: ruptime -c <Nexxus> -u <Username> -p <Password> -
n <Node>\n");
  printf("\nOptional Flags\n");
  printf("       -f  used to force compatibility with uptime\n");
  exit(-1);
}

int do_uptime(void *handle, char *node)
{
  int rc;
```

```
char *bword;
char send_pkt[4096];
char *rcvd_pkt;
int rcvd_len;
int upminutes, uphours, updays;
int numuser = 0;
double load_avg[3];
double raw_uptime;

time_t raw_time;
struct tm *time_tm;
char buf[128];
int pos = 0;

/* Get the current time */
time(&raw_time);
time_tm = localtime(&raw_time);
pos += sprintf(buf, " %2d:%02d%s  ",
  time_tm->tm_hour%12 ? time_tm->tm_hour%12 : 12,
  time_tm->tm_min,time_tm->tm_hour > 11 ? "pm" : "am");

/* Get the amount of uptime */
snprintf(send_pkt,sizeof(send_pkt),"SYSSTAT:UPTIME:%s",node);
api_nexxus_send_ipc(handle,send_pkt,strlen(send_pkt)+1);
while(1)
  {
  if((rc=api_nexxus_wait_for_data(handle,&rcvd_pkt,&rcvd_len,10))<0)
    {
    printf("Error reading uptime from Nexxus\n");
    exit(-1);
    }
  if(!(bword = strtok(rcvd_pkt,":"))) /* SYSSTAT */
    break;
  if(!(bword = strtok(NULL,":"))) /* Job ID */
    break;
  if(!(bword = strtok(NULL,":"))) /* UPTIME */
    break;
```

```
  if (!strcmp(bword,"UPTIME"))
    {
    bword = strtok(NULL,":");
    sscanf(bword,"%lf",&raw_uptime);
    free(rcvd_pkt);
    }
  else if (!strcmp(bword,"JOB_STARTED"))
    {
    free(rcvd_pkt);
    continue;
    }
  else if (!strcmp(bword,"JOB_COMPLETED"))
    {
    free(rcvd_pkt);
    break;
    }
  else if (!strcmp(bword,"JOB_ERROR"))
    {
    bword = strtok(NULL,":");
    printf("Error reading uptime from Nexxus (%s)\n",bword);
    exit(-1);
    }
  else
    {
    printf("Unexpected Response (%s)\n",bword);
    }
  }
/* Convert it to a string */
updays = (int)raw_uptime / (60*60*24);
strcat (buf,"up ");
pos += 3;
if (updays)
  pos += sprintf(buf + pos, "%d day%s, ", updays, (updays != 1) ? "s":"");
upminutes = (int)raw_uptime / 60;
uphours = upminutes / 60;
uphours = uphours % 24;
upminutes = upminutes % 60;
```

```
if (uphours)
  pos += sprintf (buf + pos, "%2d:%02d, ", uphours, upminutes);
else
  pos += sprintf (buf + pos, "%d min, ", upminutes);


/* Count the number of users */
snprintf(send_pkt,sizeof(send_pkt),"SYSSTAT:WHO:%s",node);
api_nexxus_send_ipc(handle,send_pkt,strlen(send_pkt)+1);
while(1)
  {
  if((rc=api_nexxus_wait_for_data(handle,&rcvd_pkt,&rcvd_len,10))<0)
    {
    printf("Error reading user information from Nexxus\n");
    exit(-1);
    }
  if(!(bword = strtok(rcvd_pkt,":"))) /* SYSSTAT */
    break;
  if(!(bword = strtok(NULL,":"))) /* Job ID */
    break;
  if(!(bword = strtok(NULL,":"))) /* WHO */
    break;
  if (!strcmp(bword,"WHO"))
    {
    numuser++;
    free(rcvd_pkt);
    }
  else if (!strcmp(bword,"JOB_STARTED"))
    {
    free(rcvd_pkt);
    continue;
    }
  else if (!strcmp(bword,"JOB_COMPLETED"))
    {
    free(rcvd_pkt);
    break;
    }
```

```
  else if (!strcmp(bword,"JOB_ERROR"))
    {
    bword = strtok(NULL,":");
    printf("Error reading user information from Nexxus (%s)\n",bword);
    exit(-1);
    }
  else
    {
    printf("Error reading user information from Nexxus\n");
    printf("Unexpected Response (%s)\n",bword);
    }
  }
/* Add it to our buffer */
pos += sprintf(buf + pos, "%2d user%s, ", numuser, numuser == 1 ? "":"s");


/* Get the load average */
snprintf(send_pkt,sizeof(send_pkt),"SYSSTAT:CPU_LOAD:%s",node);
api_nexxus_send_ipc(handle,send_pkt,strlen(send_pkt)+1);
while(1)
  {
  if((rc=api_nexxus_wait_for_data(handle,&rcvd_pkt,&rcvd_len,10))<0)
    {
    printf("Error reading cpu load from nexxus\n");
    exit(-1);
    }
  if(!(bword = strtok(rcvd_pkt,":"))) /* SYSSTAT */
    break;
  if(!(bword = strtok(NULL,":"))) /* Job ID */
    break;
  if(!(bword = strtok(NULL,":"))) /* UPTIME */
    break;
  if (!strcmp(bword,"CPU_LOAD"))
    {
    bword = strtok(NULL,":");
    sscanf(bword,"%lf",&load_avg[0]);
    bword = strtok(NULL,":");
```

```
      sscanf(bword,"%lf",&load_avg[1]);
      bword = strtok(NULL,":");
      sscanf(bword,"%lf",&load_avg[2]);
      free(rcvd_pkt);
      }
    else if (!strcmp(bword,"JOB_STARTED"))
      {
      free(rcvd_pkt);
      continue;
      }
    else if (!strcmp(bword,"JOB_COMPLETED"))
      {
      free(rcvd_pkt);
      break;
      }
    else if (!strcmp(bword,"JOB_ERROR"))
      {
      bword = strtok(NULL,":");
      printf("Error reading cpu load from nexxus (%s)\n",bword);
      exit(-1);
      }
    else
      {
      printf("Unexpected Response (%s)\n",bword);
      }
    }
  pos += sprintf(buf + pos, " load average: %.2f, %.2f, %.2f",
    load_avg[0], load_avg[1], load_avg[2]);

  if (!compatibility)
    printf("  Uptime Information for Node %s\n", node);
  printf("%s\n", buf);

  return(0);
}

int main(int argc, char **argv)
```

```
{
  int rc;
  char *nexxus = NULL;
  char *username = NULL;
  char *password = NULL;
  char *node = NULL;

  void *handle;
  int rcvd_len;
  char *rcvd_pkt;

  /* Deal with our commandline arguments */
  opterr = 0;
  while ((rc = getopt(argc,argv,"c:n:p:u:f"))!=EOF)
    {
    if((char)rc == 'c')
      nexxus = optarg;
    else if((char)rc == 'n')
      node = optarg;
    else if((char)rc == 'p')
      password = optarg;
    else if((char)rc == 'u')
      username = optarg;
    else if((char)rc == 'f')
      compatibility++;
    else
      display_usage();
    }
  if ((!nexxus)||(!username)||(!password)||(!node))
    display_usage();

  /* Connect to the Nexxus */
  if(api_nexxus_connect(nexxus,username,password,&handle)<0)
    {
    printf("Unable to connect to nexxus at %s (%s)\n",
      nexxus,
      api_nexxus_get_error());
```

```
    exit(-1);
    }

  /* Wait for NEXXUS_READY message */
  if((rc=api_nexxus_wait_for_data(
    handle,
    &rcvd_pkt,
    &rcvd_len,
    10))<0)
      {
      printf("ruptime: timed out waiting for NEXXUS_READY\n");
      exit(-1);
      }
  if(strcmp(rcvd_pkt, "NEXXUS_READY"))
    {
    printf("ruptime: Received unexpected data during connect (%s)\n",rcvd_pk
    exit(-1);
    }
  /* Its our responsibility to free the buffer */
  free(rcvd_pkt);

  do_uptime(handle,node);
  return(0);
}
```

# Chapter 10. Writing VACM Modules

VACM modules are the actual workhorses of VACM. IPC requests from clients are dispatched to the appropriate module via Nexxus. It is up to the module to actually take whatever action may be required to complete and satisfy the request. As management techniques, technologies, and protocols improve or evolve, it will be necessary to provide new or upgraded modules in order to provide services from within VACM. Also, there may be custom features that one installation may require that may not be desirable to another. In any event, a new module may need to be written.

VACM modules are stand alone programs which Nexxus starts upon initialization. The module connects to an AF_UNIX socket maintained by Nexxus for communication with the rest of the system. There is a utility library called libloose which makes communicating with Nexxus a lot easier, and it is reccommended that anyone writing a module use this library to maintain forward compatability with future versions of Nexxus. Once a module has connected and registered with Nexxus, it will receive IPC data relating to the nodes that are available for monitoring. The module will receive data for each node in the VACM nodelist, including any global variables set for that node. When in normal operating state, the module will also receive commands from clients over this Nexxus IPC pipe.

## 10.1. Libloose function prototypes

The libloose module API library is defined and described below. To use, simply include `libloose.h` and link against `libloose.a`. This library should be used by ALL modules to maintain future compatability.

```
int lm_init(void);
  Initialize the library. Must be your first call into the library. Al-
ways returns 0.
```

int **lm_nexxus_connect**(void);
  Connect to Nexxus via the AF_UNIX socket. Returns 0 on success, -1 with errno set on error.

int **lm_nexxus_disconnect**(void);
  Disconnect from the Nexxus AF_UNIX socket. Returns 0 on success, -1 if not connected. You should only *ever* disconnect from Nexxus if you are down

int **lm_register**(char *_short_name_, char *_long_name_, char *_desc_, char *_author_, char *_major_version_, int _minor_version_);
  Register the module with Nexxus. This should be called immediatly after connecting. Returns 0 on success, -1 with errno set on error.

int **lm_timer_add**(int _timeout_, int _(*cb) (void *)_, void *_arg_);
  Add a timer that will call function ’cb’ with argument ’arg’ in ’timeout’ seconds. If TRUE is returned from the callback function, the timer will uled. If FALSE is returned, the timer will be destroyed.

int **lm_timer_remove**(int *tag*);
  Removes a timer that was set with lm_timer_add. Returns 0 on success, -1 if timer was not set.

int **lm_watch_fd**(int *fd*, void *(*cb (int, void *)*, void **arg*);
  Add the file descriptor 'fd' to the libraries async i/o list and call function 'cb' with the FD and argument 'arg' when there is data waiting to be read on the descriptor. Returns 0 on success, -1 if no free FD watchers available.

int **lm_unwatch_fd**(int *fd*);
  Remove the file descriptor 'I_fd' from the list of descriptors being watched. Returns 0 on success, -1 if the FD is not being watched.

void **lm_main_loop**(void *(*config_cb) (__uint32_t, int, char *)*, void *(*deinit_cb) (__u*
*con_cb) (__uint32_t, int, char *)*, void *(*ipc_cb) (char *, __uint32_t, int, char *))*
  Pass control of execution to the libraries main loop. The library will execute timers and FD watchers as required. 'config_cb', is called when configuration information is sent from Nexxus. 'deinit_cb' is called when Nexxu
quests that the module
shut down. 'discon_cb' is called when Nexxus sends notification that a clien
connected. 'ipc_cb' is called whenever Nexxus dispatches an IPC message from a client to be acted on. This function never returns.

## 10.2. An Example Module

The following example module connects to Nexxus, retrieves the configuration for the system, and then goes idle. It impliments a few IPC commands. The 'PING' command will cause the module to attempt to ping the remote node. The AUTO_PING_SET command will enable or disable asynchronous event notification. If enabled, an unsolicited message with a job id of 0 is sent to the client with the results of the ping. It is a very simple module, and is here only to demonstrate how to use libloose.a to simplify the process of writing VACM modules. For an example of a more complex module, examine the EMP module source code.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/un.h>
#include <sys/time.h>
#include <libloose.h>

static void  config_cb(__uint32_t job_id, int client_fd, char *msg);
static void  deinit_cb(__uint32_t job_id, int client_fd, char *msg);
static void  discon_cb(__uint32_t job_id, int client_fd, char *msg);
static void  ipc_cb(char *node_id, __uint32_t job_id, int client_fd, char *m
static int  do_ping(__uint32_t job_id, int client_fd, char *node_id);
static int  timer_cb(void *arg);

#define TRUE  1
#define FALSE  !TRUE
#define KNOWN_UNUSED_PARAM(x) { int __bleh; __bleh = (int) x;}

typedef struct  nodelist
  {
```

```
  char            node_id[64];
  unsigned int    ip_address;
  int              subscribers[255];
  int              timer_tag;

  struct nodelist  *next;
  struct nodelist  *prev;
  } NODELIST;

static NODELIST  *node_create(char *node_id);
static NODELIST  *lookup_node(char *node_id);
static void      node_destroy(NODELIST  *node);
static void     node_remove_client(NODELIST *node, int client_fd);
static int      node_add_client(NODELIST *node, int client_fd);

static NODELIST  *nodelist_head = NULL;
static NODELIST  *nodelist_tail = NULL;

int main(argc, argv)
int    argc;
char  **argv;
{
  KNOWN_UNUSED_PARAM(argc);
  KNOWN_UNUSED_PARAM(argv);

  lm_init();
  lm_nexxus_connect();
  if (lm_register("ICMP ECHO",
                  "ICMP Echo Tester",
                  "Sends ICMP ECHO messages to a remote host to test connect
                  "San Mehat (nettwerk@valinux.com)",
                  "ICMP_ECHO",
                  1,
                  0) < 0)
    {
    printf("Unable to register module with Nexxus\n");
    exit(-1);
```

```
    }
  lm_main_loop(config_cb, deinit_cb, discon_cb, ipc_cb);
  exit(0);
}

static int timer_cb(void *arg)
{
  NODELIST  *node = (NODELIST *) arg;
  int        rc;

  rc = do_ping(0, 0, node->node_id);
#if 0
  printf("[PING] Node '%s' -> %s\n",
         node->node_id,
         (rc == 0 ? "OK" : "FAIL"));
#endif
  return(TRUE);
}

static void config_cb(__uint32_t job_id, int client_fd, char *msg)
{
  NODELIST  *node;
  char      *cmd_module;
  char      *node_id;

  KNOWN_UNUSED_PARAM(job_id);
  KNOWN_UNUSED_PARAM(client_fd);

  cmd_module = strtok(msg, ":");
  node_id = strtok(NULL, ":");

  node = lookup_node(node_id);

  if (!strcasecmp(cmd_module, "NODE"))
    node = node_create(node_id);
  else if (!strcasecmp(cmd_module, "END_NODE"))
    {
```

```
  }
else if (!strcasecmp(cmd_module, "ADDITION"))
  {
  node = node_create(node_id);
  }
else if (!strcasecmp(cmd_module, "DELETION"))
  {
  node_destroy(node);
  }
else if (!strcasecmp(cmd_module, "RENAME"))
  {
  char  *new_name;

  if (!(new_name = strtok(NULL,":")))
    {
    printf("[ICMP ECHO] Malformatted cfg msg\n");
    return;
    }
  strcpy(node->node_id, new_name);
  }
else if (!strcasecmp(cmd_module, "GLOBAL"))
  {
  char  *variable;
  char  *value;

  if (!(variable = strtok(NULL,":")))
    {
    printf("[ICMP ECHO] Malformatted cfg msg\n");
    return;
    }
  if (!(value = strtok(NULL,":")))
    {
    printf("[ICMP ECHO] Malformatted cfg msg\n");
    return;
    }
  if (!strcasecmp(variable, "IP_ADDRESS"))
    {
```

```
      struct  in_addr  addr;
      char    msg[255];

      node->ip_address = inet_addr(value);
      addr.s_addr = node->ip_address;
      snprintf(msg,sizeof(msg),"Node '%s' (ip %s) ready",
            node_id,inet_ntoa(addr));
      lm_log(msg);
      }
    }
  else if (!strcasecmp(cmd_module, "ICMP_ECHO"))
    {
    char  *variable;

    if (!(variable = strtok(NULL,":")))
      {
      printf("[ICMP ECHO] Malformatted cfg msg\n");
      return;
      }
    printf("[ICMP ECHO] Received unknown local module variable '%s'\n",
         variable);

    }
  else
    printf("[ICMP ECHO] Received unknown config event <%s>\n",cmd_module);
}

static void deinit_cb(__uint32_t job_id, int client_fd, char *msg)
{
  KNOWN_UNUSED_PARAM(job_id);
  KNOWN_UNUSED_PARAM(client_fd);
  KNOWN_UNUSED_PARAM(msg);

  printf("[ICMP ECHO] Received request to shutdown\n");
  lm_nexxus_disconnect();
  exit(0);
}
```

```
static void discon_cb(__uint32_t job_id, int client_fd, char *msg)
{
  NODELIST  *node;

  KNOWN_UNUSED_PARAM(job_id);
  KNOWN_UNUSED_PARAM(msg);

  node = nodelist_head;
  while(node)
    {
    node_remove_client(node, client_fd);
    node = node->next;
    }
}

static void ipc_cb(char *node_id, __uint32_t job_id, int client_fd, char *ms
{
  char      *cmd;
  char      response_msg[255];
  char      *node_id_glob;
  NODELIST  *node;

  if (!(cmd = strtok(msg, ":")))
    goto out_malformatted;

  if (!(node_id_glob = strtok(msg, ":")))
    goto out_malformatted;

  if (!(node = lookup_node(node_id)))
    {
    sprintf(response_msg,
            "%d:%d:FOR:JOB_ERROR:NODE_NOT_FOUND",
            job_id,
            client_fd);
    lm_send_to_nexxus(response_msg);
    return;
```

```
    }
  if (!strcasecmp(cmd, "PING"))
    {
    do_ping(job_id, client_fd, node_id);
    }
  else if (!strcasecmp(cmd, "AUTO_PING_SET"))
    {
    char      *mode;

    if (!(mode = strtok(NULL, ":")))
      goto out_malformatted;

    if (!strcasecmp(mode, "ON"))
      node_add_client(node, client_fd);
    else if (!strcasecmp(mode, "OFF"))
      node_remove_client(node, client_fd);
    else
      goto out_malformatted;
    sprintf(response_msg,
            "%d:%d:FOR:JOB_COMPLETED",
            job_id,
            client_fd);
    lm_send_to_nexxus(response_msg);
    }
  else
    {
    sprintf(response_msg,
            "%d:%d:FOR:JOB_ERROR:UNSUPPORTED_MESSAGE",
            job_id,
            client_fd);
    lm_send_to_nexxus(response_msg);
    }
  return;
out_malformatted:
  sprintf(response_msg,
          "%d:%d:FOR:JOB_ERROR:MALFORMATTED_MESSAGE",
          job_id,
```

```
          client_fd);
  lm_send_to_nexxus(response_msg);
}

static int node_add_client(NODELIST *node, int client_fd)
{
  int  I_c1;

  for (I_c1 = 0; I_c1 < 255; I_c1 ++)
    {
    if (node->subscribers[I_c1] == client_fd)
      {
      printf("[ICMP ECHO] client fd %d already subscribed!\n",client_fd);
      return(0);
      }
    }

  /* New subscription */
  if (node->timer_tag == -1)
    node->timer_tag = lm_timer_add(15,
                                    timer_cb,
                                   node);
  for (I_c1 = 0; I_c1 < 255; I_c1 ++)
    {
    if (node->subscribers[I_c1] == -1)
      {
      node->subscribers[I_c1] = client_fd;
      return(0);
      }
    }
  if (I_c1 == 255)
    {
    printf("[ICMP ECHO] No client slots available\n");
    return(-1);
    }
  return(0);
}
```

```
static void node_remove_client(NODELIST *node, int client_fd)
{
  int  I_c1;
  int  cnt = 0;

  for (I_c1 = 0; I_c1 < 255; I_c1 ++)
    {
    if (node->subscribers[I_c1] == client_fd)
      {
      node->subscribers[I_c1] = -1;
      break;
      }
    }

  if (I_c1 == 255)
    return; // Not subscribed
  for (I_c1 = 0; I_c1 < 255; I_c1 ++)
    if (node->subscribers[I_c1] != -1)
      cnt++;
  if (cnt == 0)
    {
    lm_timer_remove(node->timer_tag);
    node->timer_tag = -1;
    }
}

static int do_ping(__uint32_t job_id, int client_fd, char *node_id)
{
  char            response_msg[255];
  char            SZ_command[255];
  char            SZ_line[255];
  NODELIST        *node;
  FILE            *pipe;
  struct in_addr  addr;
  char            *p = NULL;
  char            *q;
```

```
int              I_c1;

if (!(node = lookup_node(node_id)))
  {
  if (job_id)
    {
    sprintf(response_msg,
            "%d:%d:FOR:JOB_ERROR:NODE_NOT_FOUND",
            job_id,
            client_fd);
    lm_send_to_nexxus(response_msg);
    }
  return(-1);
  }
addr.s_addr = node->ip_address;

sprintf(SZ_command,"/bin/ping -c 1 %s", inet_ntoa(addr));

if (!(pipe = popen(SZ_command,"r")))
  {
  if (job_id)
    {
    sprintf(response_msg,
            "%d:%d:FOR:JOB_ERROR:INTERNAL_ERROR (%m)",
            job_id,
            client_fd);
    lm_send_to_nexxus(response_msg);
    }
  return(-1);
  }
while(fgets(SZ_line, sizeof(SZ_line), pipe))
  {
  p = q = NULL;
  SZ_line[(strlen(SZ_line)-1)] = 0x00;
  if (strstr(SZ_line,"bytes from"))
    {
    if (!(p = strstr(SZ_line, "time=")))
```

```
      {
      sprintf(response_msg,
              "%d:%d:FOR:JOB_ERROR:INTERNAL_ERROR (NO TIME)",
              job_id,
              client_fd);
      lm_send_to_nexxus(response_msg);
      pclose(pipe);
      return(-1);
      }
    p+=5;
    if (!(q = rindex(SZ_line, ' ')))
      {
      sprintf(response_msg,
              "%d:%d:FOR:JOB_ERROR:INTERNAL_ERROR (NO TIME)",
              job_id,
              client_fd);
      lm_send_to_nexxus(response_msg);
      pclose(pipe);
      return(-1);
      }
    *q=0;
    break;
    }
  }
if (job_id)
  {
  if (!p)
    {
    sprintf(response_msg,
            "%d:%d:FOR:%s:TIMED_OUT",
            job_id,
            client_fd,
            node->node_id);
    lm_send_to_nexxus(response_msg);
    }
  else
    {
```

```
    sprintf(response_msg,
            "%d:%d:FOR:%s",
            job_id,
            client_fd,
            p);
    lm_send_to_nexxus(response_msg);
    }
  sprintf(response_msg,
          "%d:%d:FOR:JOB_COMPLETED",
          job_id,
          client_fd);
  lm_send_to_nexxus(response_msg);
  }
else
  {
  for (I_c1 = 0; I_c1 < 255; I_c1 ++)
    {
    if (node->subscribers[I_c1] != -1)
      {
      if (!p)
        {
        sprintf(response_msg,
                "0:%d:FOR:%s:TIMED_OUT",
                node->subscribers[I_c1],
                node->node_id);
        }
      else
        {
        sprintf(response_msg,
                "%d:%d:FOR:%s",
                job_id,
                client_fd,
                p);
        }
      lm_send_to_nexxus(response_msg);
      }
    }
```

```
    }

  pclose(pipe);
  if (p)
    return(0);
  else
    return(-1);
}

static NODELIST  *node_create(char *node_id)
{
  NODELIST  *new;
  int        I_c1;

  if (!(new = (NODELIST  *) malloc(sizeof(NODELIST))))
    return(0);
  memset(new, 0, sizeof(NODELIST));
  if (!nodelist_head)
    nodelist_head = nodelist_tail = new;
  else
    {
    nodelist_tail->next = new;
    new->prev = nodelist_tail;
    nodelist_tail = new;
    }
  for (I_c1=0; I_c1 <255;I_c1++)
    new->subscribers[I_c1] = -1;
  new->timer_tag = -1;
  strcpy(new->node_id, node_id);
  return(new);
}

static NODELIST  *lookup_node(char *node_id)
{
  NODELIST  *scan;

  if (!node_id)
```

```
    return(NULL);
  scan = nodelist_head;
  while(scan)
    {
    if (!strcmp(node_id, scan->node_id))
      return(scan);
    scan = scan->next;
    }
  return(NULL);
}

static void  node_destroy(NODELIST  *node)
{
  if (node == nodelist_head)
    {
    nodelist_head = nodelist_head->next;
    if (nodelist_head)
      nodelist_head->prev = NULL;
    else
      nodelist_tail = NULL;
    }
  else if (node == nodelist_tail)
    {
    nodelist_tail = nodelist_tail->prev;
    if (nodelist_tail)
      nodelist_tail->next = NULL;
    else
      nodelist_head = NULL;
    }
  else
    {
    node->next->prev = node->prev;
    node->prev->next = node->next;
    }
  free(node);
}
```

# Chapter 11. Credits

The following people have contributed to this text:

- San Mehat, <nettwerk@valinux.com>
- Zac Sprackett, <zacs@valinux.com>
- Dean Johnson, <dtj@sgi.com>
- Jerry Katzung, <katzung@valinux.com>
- Carsten Haitzler, <raster@valinux.com>

# Chapter 12. Manual Copyright and Permissions Notice

# Appendix A. Troubleshooting

# Appendix B. Contacting and Contributing

# Appendix C. Clustering VA 1000 Nodes

VA 1000 nodes use a dedicated cluster management bus (CMBus) to monitor and control the individual nodes in a cluster. The nodes are connected in a daisy-chained configuration, using the Console ports on the back of the nodes. This connection between the nodes forms the cluster management bus. It also serves as a built-in KVM switch, allowing one monitor, keyboard, and mouse to connect to any one of the nodes in the cluster.

For practical purposes, a typical cluster of VA 1000 nodes will be limited to one rack. One of the nodes is used to control the cluster, and will have the VACM nexxus installed on it. The VA 1000's CMBus is formed by connecting the Console port 1 connector from one unit to the Console port 2 connector of the next unit. This is repeated for all nodes in the cluster. A console adapter can be connected to the unused Console connector on either end of the cluster. (An additional console adapter connected to the other end of the cluster may help improve the console's video quality in clusters with more than six nodes.)

The CMBus relies on each node having a unique CMBus address. Nodes determine their addresses dynamically the first time that power is applied. The address is saved in a local EEPROM, so it is not forgotten if power is removed.

*NOTE: Because the addresses are determined the first time power is applied, it is very important to use the following procedure to power up the cluster for the first time.*

To configure a rack of VA 1000 nodes and apply power for the first time:

- Place all the VA 1000 nodes in the rack (or stack less than 6 high)
- Make sure the power switches are in the off (0) position.
- Daisy-chain the Console ports using console cables.
- Connect a Console Adapter to either end of the daisy-chain.
- Attach a keyboard, mouse, and monitor to the Console Adapter.
- Connect Ethernet and any other interface cables to the nodes.

- Connect power to all the nodes, but do not turn them on.

Finally, power up each node in the cluster, using the following procedure:

- Set the power switch on the rear of the node to the On (1) position.
- Press and release the front panel Power switch two times within a three second period. Both the blue Power and green Console LED's light up.
- Wait for the node to boot to the `login` prompt before starting the next node.

To configure VA 1000 nodes under VACM, you need to know the CMBus address of the node. To determine the address of a node, use the following procedure:

- Press and hold the Power switch down for at least 5 seconds.
- Release the Power switch.
- The two rightmost LED's will flash once, indicating the start of the CMBus address transmission.
- The three rightmost LED's will flash individually to indicate the CMBus address. From left to right, the LED's represent 100's, 10's, and 1's of the address.
- Finally, the two rightmost LED's blink together once more to indicate the end of the address transmission.