User's Guide

NMServer

SNMP System Management Agent

for VOS

COMTEK

NMServer SNMP Products

NMServer
for VOS

User's Guide Data

Date: **January 2008**

NMServer Version: **1.5.8**

NMServer
also available for
HP OpenVMS and IBM OS/400

COMTEK Contact Information

For more information on the NMServer product suite, pricing, sales, marketing, and product information, contact:

COMTEK Services, LLC
99 Ledgewood Hills Drive
Nashua, New Hampshire 03062

Sales: (703) 751-3997
FAX: (603) 881-5504
sales@COMTEKServices.com
support@COMTEKServices.com
www.comtekservices.com

Table of Contents

Chapter

1

Overview of NMServer for VOS

VOS NMServer Product Suite

NMServer is a Simple Network Management Protocol (SNMP) agent. It provides monitoring and management data for VOS Stratus and IBM System/88 systems to network managers, such as NetView or OpenView.

SNMP is an Internet Standard Protocol designed specifically for management of multi-vendor networks. Since SNMP was developed in 1988, it has become the de facto standard, and is being used to manage thousands of networks worldwide.

The data that is gathered and maintained by NMServer is represented in the Management Information Base (MIB). The MIB supported by the VOS NMServer product contains information about system resources, CPU utilization, disk utilization, process statistics, as well as hardware errors.

NMServer uses SNMP trap messages to instantly alert the network manager of critical situations on the managed system. Traps are sent for the following conditions:

- disk, page file, CPU and interrupt usage exceed user specified thresholds
- user specified critical process or program is missing
- system hardware log messages (red light event)
- user defined messages

NMServer for VOS consists of the following components:

- NM∗SysMgr - SNMP system management agent process (snmpd)
- NM∗Console - remote console process (snmpconsole)

NM∗SysMgr Agent Description

NM∗SysMgr communicates with Network Management Station(s) (NMS) by SNMP packets over UDP. The NM∗SysMgr agent collects system information on the VOS module(s) that it is configured to

monitor, responds to SNMP get, getnext and set requests from authenticated managers, and sends SNMP trap messages to configured trap destinations when system conditions exceed user configured thresholds.

In addition, the NM∗SysMgr agent provides the ability to send user data as traps. The NM∗SysMgr agent reads user data from a user trap queue, places the data into an SNMP trap message and sends it to the configured user trap destinations.

The NM∗SysMgr agent is also involved in the execution of remote console commands in that it passes remote console commands to the NM∗Console process and converts the command results placed in the user trap queue to traps and transmits them to the NMS.

The NM∗SysMgr agent supports the system and SNMP groups of MIB-II in addition to a VOS system management MIB. Refer to the NMServer MIB Subtrees section of this document for more details.

NM∗Console Process Description

The NM∗Console process handles execution of remotely entered VOS console commands. The NM∗SysMgr agent receives each console command from the NMS as an SNMP set-request on a variable in the VOS system management MIB and places the command in a command input queue for the NM∗Console process. The NM∗Console process reads the VOS command from the input queue and launches the specified process. The user may optionally specify that NM∗Console is to wait for the launched command to complete. In waited mode, NM∗Console waits for the launched command to complete and then places each line of the command results (i.e., the launched process' output file) in the NM∗SysMgr user trap queue. NM∗SysMgr is then responsible for transmitting this data as user trap messages to the NMS. In non-waited mode, NM∗Console simply launches the command and becomes immediately available to process the next command. Multiple copies of the NM∗Console process may be started, if desired.

Chapter

2

Product Installation

The following steps are required to install and configure NMServer in a new operating environment.

1. Unzip the product on your PC.
2. FTP the product to the VOS computer following instructions in the rdmevos.txt file.
3. Place the mib.conf file in the same directory as the snmpd.pm.
4. Tailor the configuration, community, trap destination, and user trap destination files (if applicable) and place them in the same directory as the snmpd.pm file.
5. Add the comtek.mib and vossysmgr.mib MIB files and the vostrapsysmgr.mib trap MIB file (if applicable) to the NMS MIB database.
6. Add the process startup commands to the system startup file.
7. Start the NMServer processes.

Setting up the Environment

Operating System Versions

For V Series (Xeon processor), VOS version 14.3 or later is required; for Continuum systems with OS TCP/IP, VOS version 13 or later is required and for Continuum systems with STREAMS TCP/IP, VOS version 14.3 or later is required.

The NMServer Program Modules

The NMServer product is compatible with VOS Continuum and V Series system architectures. Use the continuum_os_tcp directory for Continuum OS TCP/IP program modules, use the continuum_streams directory for Continuum STREAMS TCP/IP program modules, and use the VSeries_streams directory for V Series STREAMS TCP/IP program modules.

Required TCP/IP Version

NMServer for Continuum program modules are available for either OS TCP/IP or STREAMS TCP/IP; the V-Series program modules support only STREAMS TCP/IP.

SNMP Ports

NMServer utilizes the well-known SNMP ports, 161 and 162 for communication with Network Management Stations. In order for the NMServer software to function properly, it is essential that it have exclusive access to these ports.

Customize Configuration Files

Customizing the configuration files consists of the following steps:

1. Setting up the NM∗SysMgr configuration file
2. Setting up the community file
3. Setting up the trap destination file
4. Setting up the user trap destination file, if needed

Refer to the Configuration Files section of this manual for complete details on the format and content of each of the configuration files.

Setting up the NM∗SysMgr Configuration File

Edit the snmpd.config file to fill in the location and contact values. Enter any other values that you wish to supersede the predefined defaults.

Setting up the Community File

The community file is used by the NM∗SysMgr to define authentication data against which every received SNMP request will be validated. SNMP requests which fail authentication are ignored and are reported as authentication failure traps if these traps are enabled. Edit the snmpd.communities file to define the valid community strings, manager IP addresses, and privileges. The configuration file parameter AuthTrap may be used to enable/disable these traps.

Setting up the Trap Destination File

The trap destination file is used to identify the manager(s) which will receive trap messages from the NM∗SysMgr agent. Edit the

snmpd.trap_comm file to define the community strings, IP addresses, and ports for all managers that are to receive traps from the NM∗SysMgr.

Setting up the User Trap Destination File

The user trap destination file is used to identify the manager(s) which will receive user trap messages (i.e., traps of the type userQueueMessage) from the NM∗SysMgr agent. Edit the snmpd.user_trap file to define the community strings, IP addresses, and ports for all managers that are to receive user traps from the NM∗SysMgr. If this file is not present, user traps are sent to the destinations defined in the trap destination file described above.

Starting NMServer

The following process startup commands may be added to the VOS "module_start_up.cm" module startup command procedure. Note: The NM∗Console process will not start unless the NM∗SysMgr process is running.

Starting the NM∗SysMgr Agent

To start the NM∗SysMgr agent, enter the command:

```
start_process snmpd
```

This command will start the NM∗SysMgr agent process snmpd. At times it may be useful for debugging to see the data that this process is sending and receiving. This process may be started interactively in debug mode with the following command:

```
snmpd -d
```

In debug mode, the snmpd process will display every packet sent or received to the terminal. This command should be performed from a terminal with pause lines set to 0.

If the snmpd process does not run,

1. Verify that the mib.conf, snmpd.communities, snmpd.confug, snmpd.trap_comm, and snmpd.user_trap (if applicable) files are in the same directory as the snmpd.pm file.
2. Verify that OS TCP/IP is running.
3. Verify that ports 161 and 162 are not in use.
4. Use the ping command to verify that the OS TCP/IP configuration is valid.

5. Verify that the required data has been supplied in the configuration, community, and trap destination files.

To terminate the NM∗SysMgr agent, enter the following command:

stop_process snmpd

Starting the NM∗Console Process

The NM∗Console process provides the capability to enter VOS commands from the NMS. To start the NM∗Console process on the VOS system, enter the command:

start_process 'snmpconsole <input queue> <trap queue>'
-process_name <xyz>
-output_path <xyz>

The <input queue> and <trap queue> parameters are required. The full path name of both of these parameters must be specified. The <input queue> parameter must specify the name of the queue used by the NM∗SysMgr agent to send information to the NM∗Console process. The <trap queue> parameter must specify the name of the user trap queue where NM∗Console is to place its output. The <input queue> parameter and the <trap queue> parameter values must be identical to the InputQueue and TrapQueue values specified in the NM∗SysMgr configuration file snmpd.config. Since the NM∗SysMgr agent is responsible for setting up these queues, the NM∗Console process requires that the NM∗SysMgr agent be started first.

The -process_name parameter is optional. If you do not use the -process_name parameter, the name of the started process will be snmpconsole. When running more than one instance of the NM∗Console process, it may be useful to provide a unique process name to each instance so that they may be controlled individually. The -output_path parameter is required when multiple instances of this process are started in order for each process instance to have its own output file. The default output file name is snmpconsole.out.

Each remote console command may specify that either the NM∗Console process wait for the launched process to complete and then send the results of the command as a series of traps to the NMS or that NM∗Console simply launch the process and not wait for the command to complete. In order to execute multiple remote console commands simultaneously with the NM∗Console process waiting for launched process completion, more than one instance of the NM∗Console process must be started on the module. All instances of the NM∗Console process

wait for data to be placed in the same input queue, however only one instance will actually receive the data and launch the specified command. Which instance of the NM∗Console process receives the data depends upon which process VOS dispatches first.

Additional parameters may be useful on the start process command for the NM∗Console process since the parameters used to start the NM∗Console process also serve to limit the processes that the NM∗Console process launches. If the NM∗Console process is not started as privileged, then launched processes will not be privileged. Similarly, a privileged NM∗Console process may not set the priority of a launched process to be higher than its own priority.

Chapter

3

NMServer Component Operation

Sending User Data as Traps

NM*SysMgr permits user data to be converted into traps and sent to the configured trap destinations. The NM*SysMgr agent looks for user data to be converted into traps in the VOS message queue defined by the configuration file parameter TrapQueue. User data placed in this message queue must be less than 256 bytes long. This data need not be ASCII. The NM*SysMgr agent associates a unique sequence number with each message read from the TrapQueue, places the user data in the userTrapMsgsMessage instance for that sequence number, and sends the data and the sequence number as a userQueueMessage trap.

These user trap messages may be reliably received by the NMS through the use of the user trap sequence number and the table of user trap messages stored by the NM*SysMgr agent. By monitoring the highest assigned user trap sequence number so far (userNumTrapMsgs) MIB variable and/or by comparing the next expected user trap sequence number to the sequence number (userTrapMsgsNumber) received in each userQueueMessage trap message, the NMS can determine if any traps were lost during transmission. Any missing user traps may then be accessed by performing an SNMP get-request on the missing userTrapMsgsMessage instance. For example, if user trap number 3654 had not been received by the NMS, then a get-request on userTrapMsgsMessage.3654 would be performed. The number of traps retained in the user trap table may be set by the UserTrapNum configuration file parameter.

By default, user traps are sent to the same trap destinations as all other traps (as defined in the snmpd.trap_comm file). However, these traps may be routed to different trap destinations if the user supplies a user trap destination file (snmpd.user_trap). See the Configuration Files section of this document for more details.

Entering Remote Console Commands

NM∗Console allows the NMS to perform commands on the target VOS module and receive the results of those commands as user traps. Commands that may be initiated in this manner include standard VOS commands (e.g., list_users, start_process) as well as user written commands.

NM∗Console commands are entered as set-requests to the MIB variable vosInputQueueMessage. The NM∗SysMgr agent writes this data to the input queue specified by the configuration file variable InputQueue. The NM∗Console process reads the message and launches the specified command, optionally reporting the command results as a series of user trap messages.

NM∗Console accepts commands to be executed in the following format. Note: Command interpretation is position dependent, commands must be entered exactly as shown.

Bytes	Name	Description
1-5	Tag	Name of the process to be started and its associated .out file. This tag serves as a prefix to each of the user traps that are generated to report the command results (if the command is entered with the wait flag set). This field is intended to facilitate the association of user traps with specific remote console commands.
6	Wait/No Wait Flag	Setting this byte to an ASCII 1 (0x31) indicates that the NM∗Console process should wait for the launched process to finish and send the contents of the .out file to the user trap queue. Setting this byte to an ASCII 0 (0x30) causes NM∗Console to launch the process and not wait for the process to complete.
7-10	Blank	Must be blank.

11-<n>	Command	The command(s) to be launched. This field must be enclosed in single quotes. Multiple commands may be performed by a single launched process if each command is separated by a semicolon (;) within the single quotes.
<n>-255	Parameters	Following the terminating single quote, parameters to be used on the s$start_process call may be specified. A blank must appear before every keyword and every value. Available parameters are -privileged, -priority, -module, and -timeout. The -timeout value specifies, in tenths of a second, how long NM∗Console should wait for the launched process to complete. If the launched process does not complete before this time period expires, the launched process is terminated with a s$stop_process call.

Examples

In each example below a column scale is provided, followed by the command, followed by an explanation.

```
(column) 1         2         3         4         5
12345678901234567890123456789012345678901234567890123456
ABC011     'ddu'
```

The NM∗Console performs a s$start_process on the VOS command display_disk_usage and waits for the process to complete. For each line in the resulting output file, NM∗Console reads the line, prefixes it with the ABC01 tag, and writes the line to the to the user trap queue. NM∗SysMgr then reads each of the lines from the user trap queue and sends the line as a user trap to the configured user trap destination(s).

```
(column) 1         2         3         4         5
12345678901234567890123456789012345678901234567890123456
```

```
A12341    'scan_disk_util #d03' -timeout 300 -privileged
```

NM*Console starts a customer written program named scan_disk_util as a privileged process. The parameter #d03 is passed as a command line parameter to the started process. If this process does not complete within 30 seconds, the NM*Console process executes a s$stop_process on this process. The resulting output file A1234.out is read and each line is prefixed with the tag and placed in the user trap queue. NM*SysMgr sends the output data as user traps to the configured user trap destination(s).

```
(column) 1         2         3         4         5
12345678901234567890123456789012345678901234567890123456789012345678901234567
ZZTOP1    'ddu;lu' -module %SI#M3
```

The ZZTOP process is started on module %SI#M3 to execute the display_disk_usage command followed by the list_users command. The results of this process are reported as user traps.

Chapter

4

Configuration Files

Distribution Kit

The VOS NMServer product zip file contains the NMServer product. Once the zip file has been unzipped, the following directories will be present:

Agent\v1.5.8\continuum_os_tcp
Agent\v1.5.8\continuum_streams
Agent\v1.5.8\VSeries_streams
Doc
MIB
OpenView

Product installation instructions can be found in the file **rdmevos.txt**.

In addition to the platform-specific product directories, the **v1.5.8** directory contains the configuration files for NMServer. These files must be placed in the directory where snmpd.pm will be run. Note that the mib.conf file needs no modification.

Configuration File	Description
mib.conf	NM∗SysMgr license data
send_trap.c	Contains example program for placing data in the user trap queue.
snmpd.communities	NM∗SysMgr community file
snmpd.config	NM∗SysMgr configuration file
snmpd.trap_comm	NM∗SysMgr trap destination file

The **continuum_os_tcp** directory contains the program modules for use on Continuum VOS modules running OS TCP/IP.

Program Module	Description
getinfo.pm	Agent debugging tool for use by COMTEK engineers.
monitor_log.pm	Log file monitor program module.

snmpconsole.pm	The NM∗Console process program module.
snmpd.pm	The NM∗SysMgr agent program module.
snmptrapd.pm	Trap receipt debugging tool.
snmpwalk.pm	MIB walk debugging tool.

The **continuum_streams** directory contains the program modules for use on Continuum VOS modules running STREAMS TCP/IP.

Program Module	Description
getinfo.pm	Agent debugging tool for use by COMTEK engineers.
monitor_log.pm	Log file monitor program module.
snmpconsole.pm	The NM∗Console process program module.
snmpd.pm	The NM∗SysMgr agent program module.
snmptrapd.pm	Trap receipt debugging tool.
snmpwalk.pm	MIB walk debugging tool.

The **VSeries_streams** directory contains the program modules for use on VOS V Series modules running STREAMS TCP/IP.

Program Module	Description
getinfo.pm	Agent debugging tool for use by COMTEK engineers.
monitor_log.pm	Log file monitor program module.
snmpconsole.pm	The NM∗Console process program module.
snmpd.pm	The NM∗SysMgr agent program module.
snmptrapd.pm	Trap receipt debugging tool.
snmpwalk.pm	MIB walk debugging tool.

The **Doc** directory contains product documentation.

MIB	Description
Vos.pdf	This user's guide.
Vosfaq.txt	Frequently asked questions.

The **mib** directory contains COMTEK's MIB and Trap MIB files for the VOS NM∗SysMgr agent. These are not used by the NMServer software but are to be exported to the network management stations that will be querying this system and receiving traps. The trap MIB document is in RFC 1215 format which is informational, and is not supported by some Network Management Stations.

MIB	Description

comtek.mib	Contains COMTEK-DEFINITIONS-MIB which is standard for all COMTEK NMServer products.
vossysmgr.mib	Contains COMTEK-MIB, the MIB for the VOS NM*SysMgr agent.
vostrapsysmgr.mib	Contains COMTEK-TRAP-MIB, the trap MIB for the VOS NM*SysMgr agent.

The **OpenView** directory contains tools for use with HP OpenView NNM.

MIB	Description
NMServer_VOS_Traps.xls	Spreadsheet showing the default NNM trap configuration.
Nmservervos_openview.zip	Zip file containing NNM tools.
Openview_readme.txt	Instructions for installing NMServer VOS tools into OpenView NNM.

Process Configuration Files

NM*SysMgr Configuration File

The configuration file is read by the NM*SysMgr agent when the process is started. If changes are made to this file, the NM*SysMgr agent must be stopped and restarted for the changes to take effect.

The following is an example of the NM*SysMgr confguration file, snmpd.config:

```
#
# NMServer for VOS Configuration File
#
AuthTrap=yes
Contact=COMTEK Services (703) 751-3997
CpuUtil=90
DiskAlarm=30
DiskUse=90
InputQueue=%StratSys#d01>system>snmp>RemoteConsole
IntUtil=20
Location=Westwood Center Drive, Vienna VA
LocalModule=Yes
MibConf=%StratSys#d01>system>snmp>mib.conf
MinFreeRecords=%dev2#d02,19500
```

```
Module=%Comtek#m1
Name=comtk1.comtek.com
NewPagingCheck=1
PageUse=90
ProcTrap=%dev1#m1,ProcessAbc,5
ProgTrap=%dev1#m1,x25.pm,10
RedLight=Yes
RptProcAlarm=120
TrapQueue=%StratSys#d01>system>snmp>UserTrapQueue
Traps=Yes
UserTrapNum=100
```

This sample demonstrates the use of each variable which may be placed in the configuration. Note that in actual use, the LocalModule and Module parameters should not be combined in the same configuration file.

The format of the file is a left justified variable name followed by an '=' without intervening spaces or tabs. The variable names may be expressed in upper, lower or mixed case. If one of the optional variables is missing, misspelled, or has an invalid value, that variable is ignored and a default value is used. The agent writes a synopsis of its configuration to its .out file. Examine this file to verify that configuration values are as expected. See below for MIB equivalents, optional variables and default values. SNMP set-requests performed on MIB variables associated with these configuration variables modify the agent's active values but do not affect the configuration file on disk. To make any such changes permanent, they must be made to the configuration file manually.

Comments may be inserted in the configuration file by placing a pound sign (#) as the first character in the line.

AuthTrap=This variable determines whether any authentication failure traps are sent. If this variable is set to YES, authentication failure traps are sent. If this variable is set to NO, no authentication failure traps are sent.

Characteristics	Allowed values
MIB-II Variable	snmpEnableAuthenTraps
Required/Optional	Optional
Valid Values	YES, NO
Default Value	NO
Set-requests	Not allowed.

Contact=This variable is usually the name and means of contacting the administrator for this machine.

Characteristics	Allowed values
MIB-II Variable	sysContact
Required/Optional	Optional
Default Value	""
Set-requests	Take effect immediately

CpuUtil=This variable specifies the threshold percentage of CPU usage by processes and interrupts at which cpuUsageExcessive traps begin to be sent. Once this threshold is reached, cpuUsageExcessive traps are sent once per minute until the CPU utilization drops below this threshold. This value is calculated on a per module basis and is normalized for multiple CPU machines so that it expresses the total CPU capacity of the system during a one minute interval. A value of zero disables these trap messages.

Characteristics	Allowed values
Required/Optional	Optional
Valid Values	0..100
Default Value	0

DiskAlarm=This variable specifies how frequently the trap messages diskFull, diskNoFreeSpace, and pageFull are sent while utilization remains above the specified use limit. A value of zero causes only the initial trap message to be sent. This value is expressed in minutes.

Characteristics	Allowed values
Required/Optional	Optional
Default Value	0

DskUse=This variable specifies the percentage of disk file partition utilization at which diskFull trap messages begin to be sent. Once this threshold has been reached, diskFull traps are repeated as specified by the diskAlarm parameter until the situation is resolved, when the diskFullClear trap is sent. A value of zero disables diskFull trap messages.

Characteristics	Allowed values
Required/Optional	Optional
Valid Values	0..100
Default Value	0

InputQueue=This variable specifies the queue name where the NM*SysMgr agent is to place remote console commands to be executed by the NM*Console process. There must be an exact match between the value specified for this parameter and the <input queue> parameter used when starting the NM*Console snmpconsole process. This variable must specify the fully qualified path name for this queue. If the specified queue does not exist when the NM*SysMgr is started, it will be created.

Characteristics	Allowed values
Required/Optional	Required for NM*Console process to function.

IntUtil=This variable specifies the threshold percentage of total CPU usage by interrupts at which interruptCpuUsageExcessive traps begin to be sent. Once this threshold is reached, interruptCpuUsageExcessive traps are sent once per minute until the CPU utilization by interrupts drops below this threshold. This value is calculated on a per module basis and is normalized for multiple CPU systems so that it expresses the total CPU capacity of the system. A value of zero disables these trap messages.

Characteristics	Allowed values
Required/Optional	Optional
Valid Values	0..100
Default Value	0

Location=This variable is typically a description of the system location which is meaningful to the network administrator.

Characteristics	Allowed values
MIB-II Variable	sysLocation
Required/Optional	Optional
Default Value	“”
Set-requests	Take effect immediately

LocalModule=This variable is used to indicate that system information is only to be gathered on the module on which the NM*SysMgr agent is running. This variable supersedes any values specified by the Module variable. This variable is only meaningful in a multi-module environment. If this option is not desired, do not set this value to NO, rather remove the entire entry from the configuration file.

Characteristics	Allowed values
Required/Optional	Optional

Valid Values	YES

MibConf=This variable may be used to specify the full path name of the license file. By default, the license key file is assumed to mib.conf and assumed to be in the same directory as the snmpd.pm program module. Specifying this parameter allows the license key file to be placed elsewhere.

Characteristics	Allowed values
Required/Optional	Optional
Default Value	mib.conf

MinFreeRecords=This variable specifies the minimum number of free 4096 byte blocks required on the specified disk. If the number of free blocks for the named disk drops below the specified level, a diskNoFreeSpace trap message is sent. This trap message will be repeated as defined by the DiskAlarm configuration item until the amount of free space on the specified disk exceeds the specified minimum at which time the diskNoFreeSpaceClear trap is sent. The format of this entry is:

MinFreeRecords=<disk name>,<minimum free blocks>

A separate MinFreeRecords entry is made for each disk to be monitored.

Characteristics	Allowed values
Required/Optional	Optional
Multiple Entries	Permitted

Module=In a multi-module environment, this parameter is used to select specific modules for which NM*SysMgr is to gather information. The LocalModule parameter, if used, supersedes this parameter. If neither the LocalModule nor the Module parameter is used, all modules will be monitored. Multiple instances of this parameter may be used in the configuration file.

Characteristics	Allowed values
Required/Optional	Optional
Multiple Entries	Permitted

Name=This variable is the name of the system on which the agent is running. If this value is not supplied in the configuration file, the name returned from gethostname() is used.

Characteristics	Allowed values

MIB-II Variable	sysName
Required/Optional	Optional
Default	Determined by gethostname()
Set-requests	Take effect immediately

NewPagingCheck=This variable is used to enable a more accurate set of system calls to get paging space usage. This parameter is only effective on VOS 11.5 or higher. By default, the agent uses an older, less accurate method of calculating page usage. To enable the more accurate paging calculation, set this parameter to any non-zero value. When the agent detects this parameter set at startup, it checks the version of VOS that is running to determine if the more accurate paging space system calls are available. If the VOS operating system version is 11.5 or higher, the new paging space calculations will be used and messages similar to the following will appear in the process output file:

```
VOS Release 11.7.2t
This system is at VOS 11.7 so new paging check is enabled.
Paging: Use 2911, Total 30000
```

If the version of VOS will not support this, the older page space calculations will be performed and messages similar to the following will be printed to the output file:

```
VOS Release 11.1a
Must be at least VOS 11.5 to do new paging. This system is at
VOS 11.1 so new paging check is disabled.
```

Characteristics	Allowed values
Required/Optional	Optional
Valid Values	0 (disabled), non-zero (enabled)
Default Value	0

PageUse=This variable specifies the percentage of paging partition utilization at which pageFull trap messages begin to be sent. Once this threshold has been reached, pageFull traps are repeated as specified by the diskAlarm parameter until the situation is resolved and the pageFullClear trap is sent. A value of zero disables pageFull trap messages.

Characteristics	Allowed values
Required/Optional	Optional
Valid Values	0..100
Default Value	0

ProcTrap=This variable is used to define processes that are critical in the user's operating environment. The NM*SysMgr agent monitors the specified module for the named process and verifies that the required number of instances of that process are running. If the number of process instances drops below the required number, then a criticalProcessMissing trap is sent. The format of ProcTrap entries is:

ProcTrap=<module name>,<process name>,<required count>

A separate ProcTrap entry is made in the configuration file for each process that is to be monitored.

Characteristics	Allowed values
Required/Optional	Optional
Multiple Entries	100 ProgTrap and ProcTrap combined entries are permitted

ProgTrap=This variable is used to define programs that are critical in the user's operating environment. The NM*SysMgr agent monitors the specified module for the named program and verifies that the required number of instances of that program are running. If the number of program instances drops below the required number, then a criticalProgramMissing trap is sent. The format of ProgTrap entries is:

ProgTrap=<module name>,<program name>,<required count>

A separate ProgTrap entry is made in the configuration file for each program that is to be monitored.

Characteristics	Allowed values
Required/Optional	Optional
Multiple Entries	100 ProgTrap and ProcTrap combined entries are permitted

RedLight=This variable determines whether hardware log events are sent as traps. When this variable is set to YES, each new entry placed in the >system>hardware_log.* file on each monitored module will be added to the vosRedLightTable table and will generate a new redLightEvent trap. If this variable is set to NO, no hardware error traps are sent.

Characteristics	Allowed values
Required/Optional	Optional
Valid Values	YES, NO
Default Value	YES

RptProcAlarm=This variable defines the number of minutes between repeat traps for a given critical process or critical program missing condition. Setting this variable to zero causes only the initial detection of a missing process or program to generate a trap.

Characteristics	Allowed values
Required/Optional	Optional
Default Value	0

TrapQueue=This variable specifies the name of the queue where NM∗SysMgr is to receive user data that is to be sent to the NMS(s) as traps. Each message that is placed in this queue is assigned a unique userTrapMsgsNumber sequence number, converted into a userQueueMessage trap, sent to the appropriate trap destinations. and stored in the userTrapMsgsTable table. There must be an exact match between the value specified for this parameter and the <trap queue> parameter used when starting the NM∗Console snmpconsole process. This variable must specify the fully qualified path name for this queue. If the specified queue does not exist when the NM∗SysMgr is started, it will be created.

Characteristics	Allowed values
Required/Optional	Required

Traps=This variable determines whether any traps are sent. When this variable is set to YES, SNMP standard and COMTEK enterprise specific traps are sent to the configured trap destinations. If this variable is set to NO, no traps are sent.

Characteristics	Allowed values
Required/Optional	Optional
Valid Values	YES, NO
Default Value	YES

UserTrapNum=This variable determines the maximum number of entries in the userTrapMsgsTable user trap table. If the value specified is less than 0, the default value is used. If the specified value is greater than the maximum, the maximum value is used. A value of 0 disables user traps.

Characteristics	Allowed values
Required/Optional	Optional
Valid Values	0..10,000
Default Value	40

NM∗SysMgr Community File

The community file is used by the NM∗SysMgr agent to specify data to be used in community based message authentication. This file is named snmpd.communities. Since the NM∗SysMgr agent only reads this file at startup, any changes made while the agent is running will not take effect until the agent is stopped and restarted.

The following is an example of a community file:

```
#<session name>  <IP address in dot notation> <privileges>
public 146.170.228.211 NONE
private 162.54.1.2  WRITE
private 162.54.1.110  WRITE
public  0.0.0.0 read
snarf  0.0.0.0 write
```

The first field is the community field. This identifies the community string which is to be part of the SNMP message. This field is case sensitive.

The second field is the IP address of the remote site which is to be associated with the given community string and privileges. An IP address of 0.0.0.0 indicates that any IP address may communicate with the NM∗SysMgr agent using the specified community string and be granted the specified privileges.

The third field contains the privilege type to be granted the entry. Valid privilege values are: READ (for read-only), WRITE (for read-write), and NONE (for no access). This field is not case sensitive.

This file may contain a maximum of 100 entries. Comments may be inserted in the community file by placing a pound sign (#) as the first character of the line. Fields in this file must be separated by one or more blanks or tabs.

NM∗SysMgr Trap Destination File

The trap destination file is used to specify the community string, port number, and IP address of managers that are to receive the trap messages sent by the NM∗SysMgr agent. This file is named snmpd.trap_comm. If this file is left blank, no trap messages will be sent.

The following is an example of a trap destination file:

```
#<community>  <IP address>  <port>
public 127.0.0.1 162
```

```
public 162.54.2.121 162
public 162.54.1.8 162
```

The first field contains the community string which is to accompany the trap. The second field contains the IP address of the host which is to receive the trap. The third field contains the port number to which the trap is to be sent. This field typically has the value 162.

The NM*SysMgr agent provides the writeable MIB variable vosTrapCommFileName which, when modified by a set-request, causes the NM*SysMgr to reinitialize its trap destinations using the data in the specified file.

A maximum of 100 entries may be in this file. Comment lines have a pound sign (#) as the first character in the line. Fields in this file must be separated by one or more blanks or tabs.

NM*SysMgr User Trap Destination File

The user trap destination file is used to specify the community string, port number, and IP address of managers that are to receive the user trap messages sent by the NM*SysMgr agent. This file is named snmpd.user_trap. If this file does not exist, all trap messages will be routed as defined in the snmpd.trap_comm file. This file provides the ability to send user trap messages to a different manager or managers than all other traps. This file is the same format as the snmpd.trap_comm file. The MIB variable vosUserTrapCommFileName may be modified by a SNMP set-request to change the user trap destinations without stopping the agent.

MIB Files

All MIB files are supplied for use by Network Management Stations. These files are not accessed by the NMServer products. MIB files are written in ASN.1 format.

comtek.mib File

The comtek.mib file contains the COMTEK-DEFINITIONS-MIB which defines the COMTEK enterprise and the base object identifier for the NM*SysMgr agent. This file is referenced by all other COMTEK MIBs. This file is dependent on RFC1155-SMI.

vossysmgr.mib File

The vossysmgr.mib file contains the COMTEK-MIB MIB which defines objects supported by the VOS NM∗SysMgr agent. This file is dependent on the COMTEK-DEFINITIONS-MIB (comtek.mib), RFC1155-SMI and RFC1213-MIB.

Trap Files

All trap files are written in ASN.1 format in compliance with RFC1215, which is informational. Use of trap files is not supported by all NMS systems. These files are not accessed by the NMServer products.

vostrapsysmgr.mib File

This file contains the COMTEK-TRAP-MIB Trap MIB which describes the traps and associated variables generated by the VOS NM∗SysMgr agent. This file in dependent on the COMTEK-DEFINITIONS-MIB (comtek.mib), COMTEK-MIB (vossysmgr.mib), RFC-1215, and RFC1213-MIB.

Chapter

5

NMServer Traps

NM*SysMgr Traps

The following information provides an overview of the traps which may be generated by the NM*SysMgr Agent. Refer to the trap MIB COMTEK-TRAP-MIB (vostrapsysmgr.mib) for a complete description of each trap, the variables which accompany it, and the circumstances under which it will be generated.

SNMP Generic Traps

Trap	Description
coldStart(0)	The NM*SysMgr agent has completed a cold start.
authenticationFailure(4)	The NM*SysMgr agent received an SNMP message which was not properly authenticated.

Enterprise Specific Traps

Trap	Description
diskFull(0)	Disk system usage reached or exceeded critical threshold.
diskFullClear(1)	Disk system usage has gone below critical threshold.
pageFull(2)	Page partition usage has reached or exceeded critical threshold.
pageFullClear(3)	Page partition usage has gone below critical threshold.
cpuUsageExcessive(4)	Excessive CPU utilization by all processes and interrupts.
interruptCpuUsageExcessive (5)	Excessive CPU utilization by interrupts.
readLightEvent(6)	Item added to >system>hardware_log.
userQueueMessage(7)	Item added to agent trap queue by another process.

criticalProcessMissing(8)	Too few instances of critical process name in process table.
criticalProgramMissing(9)	Too few instances of critical program name in process table.
diskNoFreeSpace(10)	Critical threshold for free space on disk has been reached.
diskNoFreeSpaceClear(11)	Disk usage has now gone below critical threshold.

Chapter

6

NMServer MIB Subtrees

MIB-II

System Group

System Description
System Object Identifier
System Up Time
System Contact
System Name
System Location
System Services

SNMP Group

SNMP In Packets
SNMP Out Packets
SNMP In Bad Versions
SNMP In Bad Community Names
SNMP In Bad Community Uses
SNMP In ASN Parse Errors
SNMP In Too Bigs
SNMP In No Such Names
SNMP In Bad Values
SNMP In Read Onlys
SNMP In General Errors
SNMP In Total Req Vars
SNMP In Total Set Vars
SNMP In Get Requests
SNMP In Get Nexts
SNMP In Set Requests
SNMP In Get Responses
SNMP In Traps
SNMP Out Too Bigs
SNMP Out No Such Names

SNMP Out Bad Values
SNMP Out General Errors
SNMP Out Get Requests
SNMP Out Get Nexts
SNMP Out Set Requests
SNMP Out Get Responses
SNMP Out Traps
SNMP Enable Authentication Traps

COMTEK-MIB Subtree

VOS Group

Number of Modules
Module Usage Table
- Module Number
- Module Name
- Operating System Version
- CPU Type
- Number of CPUs
- Percent of Module Paging Used

Number of Entries in Red Light Table
Red Light Table
- Red Light Number
- Red Light Module Name
- Red Light Message

Configuration File Name
Input Queue Message
Input Queue Name
Trap Community File Name
User Trap Community File Name

CPU Group

CPU Load Table
- Module Load Index
- Module Name
- CPU One Minute Load
- CPU Five Minute Load
- CPU Fifteen Minute Load
- Interrupt One Minute Load
- Interrupt Five Minute Load
- Interrupt Fifteen Minute Load
- Page Fault One Minute Load

Page Fault Five Minute Load
Page Fault Fifteen Minute Load
CPU Usage Table
Module Usage Index
Module Name
Percent of CPU Used by User Processes
Percent of CPU Used by System Processes
Percent of CPU Used by Module Interrupts
Percent of CPU Used by Network Server Processes
Percent of CPU Idle

Disk Group

Number of Disks in Disk Table
Disk Table
Disk Table Index
Module Name
Name of Disk for Module
Type of Disk
Disk Size
File Partition Size
Percent File Partition Used
Page Partition Size
Percent Page Partition Used
Fatal Errors for Disk
Data Errors for Disk
Disk Read Count
Disk Write Count
Amount of File Partition Free

Process Group

Number of Running Processes
Process Table
VOS Process Identifier
Module Name
Process Name
User Name of Process Owner
Group Name of Process Owner
Program Name
Invoking Process Identifier
Process State
Process Priority
CPU Time Used by the Process
CPU Time Used for Page Faults

Number of Page Faults
Working Set Size
Disk Reads
Disk Writes
Number of Copies of this Process on this Module
Number of Copies of this Program on this Module

User Trap Message Group

Highest Sequence Number in User Trap Message Table
User Trap Message Table
User Trap Message Sequence Number
User Trap Message

Appendix

A

Sample Programs

The following are sample C source programs to assist the user in placing user data to be sent as traps in the TrapQueue.

Bind Control File

```
/* bind control file for testtrap.c */

test.bind
name:     testtrap;
entry:    main;
modules:  testtrap, send_trap;
size: small;
end;

/*  end of bind  control file for testtrap.c */
```

send_trap.c

```
/*  program which places message in special queue */

/*
   The open_trapq routine must be executed before writing to the trap_q.
   You must supply a full path name in the form of %sys#xxx>aa>tq of the
   trap_q; this name must be identical to the name supplied to NMServer
   in the configuration file.  If your process runs before you bring up
   NMServer, your process will create the trap_q and then Open it;
   otherwise, your process will only Open the queue.

   The port_id and an event_id will be returned if the error_code is zero.
   You must save the port_id for use in the send_trapq function, but the
   event_id may be discarded if you only send to the trap_q and do not
   intend to read from it.

   If your return is not zero, the Operation failed and you cannot send
   any traps until the Open problem is resolved.

   The calling sequence is as follows:

char  fpath[100];   I   full path name of the queue to be opened
short port_id;      I   port id
```

```
long  event_id;     O   event id
short error_code;
     error_code = open_trapq(fpath, &port_id, &event_id);
*/

/* open_trapq routine */

extern void s$attach_port();
extern void s$create_file();
extern void s$set_no_wait_mode();
extern void s$msg_open();
extern void s$msg_send();


short open_trapq(fpath,portid,eventid)
char  *fpath;      /* I full path name of the queue to be opened */
short *portid;     /* O port id of the queue */
long  *eventid;    /* O event id */
{
    short error;
    char_varying(256) fullpath;

    strcpy(&fullpath,fpath);
    s$create_file(&fullpath,&(short)6,&(short)1,&error);
    if(error && error != 1050) /* 1050 = e$file_exists */
        return error;
    s$attach_port(&(char_varying(0))"",&fullpath,&(short)1,portid,&error);
    if(error && error != 1008) /* 1008 = e$port_already_attached */
        return error;
    s$msg_open(portid,&(short)5,&error);  /* open as requester I/O type */
    if(error)
        return error;
    s$set_no_wait_mode(portid,eventid,&error);
    return error;
}

/*
   The open_trapq routine must have been executed before using send_trapq
   to write to the trap_q.
   You must use the saved port_id to call to send_trapq function. Additionally
   you must supply the address of your message and its length including
   the null terminator.

   If your return is not zero, the Operation failed and you did not write
   the trap into the queue.

   The calling sequence is as follows:

short port_id;      I   port id
char  msg[256];     I   the data area where the trap msg is put
long  msg_length;   I   number of bytes to send;must not exceed 256
short error_code;
```

```
     error_code = send_trapq(&port_id, msg, &msg_length);
*/

/* send_trapq routine */

short send_trapq(portid,msg,msg_length)
short *portid;      /* I port id */
char  *msg;         /* I   ptr to the data area where the trap msg is  */
long  *msg_length;  /* I   size of the msg to send  */
{
    static char_varying(32) msg_subject = ""; /* no subject  */
    short error;
    static short msg_priority = 0;  /* always use lowest priority */
    static long msg_id = 0;         /* constant 0  */

    s$msg_send(portid, &msg_priority, &msg_subject, msg_length, msg,
            &msg_id, &error);

    return error;

}
```

testtrap.c

```
/*  test trap generator  program */

/*
   Generates strings of variable lengths and sends them to trap_Q.  Messages
   start with contents of "0" and go thru "9" and wrap back to "0".  User must
   change path from %dev1#d02>users>....>trap_Q to the path used in his
   installation (in the snmpd.config file), recompile and bind -control test.bind.
   The user may freely change these programs to suit his needs.
*/

#include <stdio.h>
extern short send_trapq();
extern short open_trapq();
main()
{
    char  path[256];
    short port;
    long  event;
    short er;
    long  bufferlen;
    char  buffer[257];
    static unsigned char fill = 0x30;
    unsigned long a = 1229;
    unsigned long b = 199363;
    strcpy(path,"%dev1#d02>users>Don_Winans>snmp>apps>trap_Q");
    er = open_trapq(path,&port,&event);
    if(!er)
    {
        while(!er)
```

```
        {
            a = a * b;
            bufferlen = a % 256;
            if (!bufferlen) continue;
            memset(buffer, fill, bufferlen);
            if ((fill += 1)>0x39) fill = 0x30;
            er = send_trapq(&port,buffer,&bufferlen);
            sleep(30);  /* rest half minute */
        }
    }
}

/*  end of sample programs  */
```

Appendix

B

NMServer MIBs

comtek.mib

```
--
-- file: comtek.mib
--
-- COMTEK Services, Inc.
-- Date         March 1997
-- Author       Nancy Fink
--
-- Copyright 1994-2008 COMTEK Services, Inc.  All Rights Reserved.
--
-- This COMTEK Services SNMP Management Information Base Specification
-- (Specification) embodies COMTEK Services' confidential and
-- proprietary intellectual property.  COMTEK Services retains all
-- title and ownership in the Specification, including any
-- revisions.
--
-- This Specification is supplied "AS IS," and COMTEK Services makes
-- no warranty, either express or implied, as to the use,
-- operation, condition, or performance of the Specification.
--

COMTEK-DEFINITIONS-MIB DEFINITIONS ::= BEGIN

IMPORTS
     enterprises FROM RFC1155-SMI;

comtek OBJECT IDENTIFIER ::= { enterprises 597 }

comtekvosMib OBJECT IDENTIFIER                  ::= { comtek 1 }
comtekVosAgent OBJECT IDENTIFIER                ::= { comtek 2 }
comtekos400Mib OBJECT IDENTIFIER                ::= { comtek 3 }
comtekVms OBJECT IDENTIFIER                     ::= { comtek 4 }

--SUBAGENT DEFINITIONS

--VOS Subagents:

--OS/400 Subagents:

--OpenVMS Subagents:
--Note:  The following pairs of object identifiers must match the values
--in by the subagent code. These numbers uniquely identify the MIB and
--subagent.  Object identifiers for new subagents and their corresponding
--MIBs should be added to the end of this list.
comtekVmsNMMasterMib OBJECT IDENTIFIER          ::= { comtekVms 1 }
comtekVmsNMMasterAgent OBJECT IDENTIFIER        ::= { comtekVms 2 }
comtekVmsNMSysMgrMib OBJECT IDENTIFIER          ::= { comtekVms 3 }
comtekVmsNMSysMgrSubagent OBJECT IDENTIFIER     ::= { comtekVms 4 }
```

```
comtekVmsNMTrpMgrMib OBJECT IDENTIFIER          ::= { comtekVms 5 }
comtekVmsNMTrpMgrSubagent OBJECT IDENTIFIER     ::= { comtekVms 6 }
comtekVmsNMConsoleMib OBJECT IDENTIFIER         ::= { comtekVms 7 }
comtekVmsNMConsoleSubagent OBJECT IDENTIFIER    ::= { comtekVms 8 }
comtekVmsNMTemplateMib OBJECT IDENTIFIER        ::= { comtekVms 9 }
comtekVmsNMTemplateSubagent OBJECT IDENTIFIER   ::= { comtekVms 10 }

END
```

vossysmgr.mib

```
--
-- file: vossysmgr.mib
--
-- COMTEK Services, Inc. NM*SysMgr MIB for VOS (Stratus, IBM System 88)
-- Release      1.5.8
-- Date         Feb, 1997
-- Author       Steve Harris
--
-- Copyright 1997-2008 COMTEK Services, Inc.  All Rights Reserved.
--
-- This COMTEK Services SNMP Management Information Base Specification
-- (Specification) embodies COMTEK Services' confidential and
-- proprietary intellectual property.  COMTEK Services retains all
-- title and ownership in the Specification, including any
-- revisions.
--
-- This Specification is supplied "AS IS," and COMTEK Services makes
-- no warranty, either express or implied, as to the use,
-- operation, condition, or performance of the Specification.


-- The operation of the NM*SysMgr agent for VOS is controlled via
-- the configuration file (snmpd.config), the community file (snmpd.comm),
-- the trap destination file (snmpd.trap_comm), and the user trap
-- destination file (snmpd.user_trap).  See the User's Guide for a
-- complete description of each of these files.


-- VOS times are put into time ticks.  CPU utilization statistics are
-- normalized by number of processors so they will fall in the range of
-- 0-100.


COMTEK-MIB DEFINITIONS ::= BEGIN

IMPORTS
     DisplayString                    FROM RFC1213-MIB
     OBJECT-TYPE, TimeTicks, Counter, enterprises
                                      FROM RFC1155-SMI
     comtek, comtekvosMib, comtekvosAgent
                                       FROM COMTEK-DEFINITIONS-MIB;

-- The following is used to identify versions of the VOS agent
comtekVosAgent OBJECT IDENTIFIER   ::= { comtekvosAgent 1 }

-- This group lists system resources available (OS, CPU, disk)
resources OBJECT IDENTIFIER       ::= { comtekvosMib 1 }

-- This group contains a table of VOS processes and statistics
processes OBJECT IDENTIFIER       ::= { comtekvosMib 2 }

-- This group contains a table of user trap data
userinfo OBJECT IDENTIFIER       ::= { comtekvosMib 3 }

vos  OBJECT IDENTIFIER      ::= { resources 1 }
cpu  OBJECT IDENTIFIER      ::= { resources 2 }
disk OBJECT IDENTIFIER      ::= { resources 3 }

-- The VOS agent may be configured to process data for one, selected,
-- or all modules.  By default, the agent gathers data for all accessible
```

```
-- modules.  To limit data gathering to the module on which the agent is
-- run, specify "localModule=1" in the snmpd.config configuration file.  To
-- select a subset of modules that the agent is to monitor, specify each
-- module to be monitored using the "monitor=<module name>" configuration
-- parameter.

vosNumModules OBJECT-TYPE
     SYNTAX  INTEGER
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Number of modules."
     ::= { vos 1 }

vosTable  OBJECT-TYPE
     SYNTAX  SEQUENCE OF VosTableEntry
     ACCESS  not-accessible
     STATUS  mandatory
     DESCRIPTION
         "A simplified version of what is read from
         s$get_module_usage()."
     ::= { vos 2 }

vosTableEntry  OBJECT-TYPE
     SYNTAX  VosTableEntry
     ACCESS  not-accessible
     STATUS  mandatory
     DESCRIPTION
         "A row in the module table."
     INDEX { vosModuleNumber }
     ::= { vosTable 1 }

VosTableEntry ::= SEQUENCE {
     vosModuleNumber   INTEGER,
     vosModuleName     DisplayString,
     vosVersion        DisplayString,
     vosCPUtype        DisplayString,
     vosNumCPUs        INTEGER,
     vosModPaging      INTEGER
 }

vosModuleNumber   OBJECT-TYPE
     SYNTAX  INTEGER
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Table index number assigned to this module."
     ::= { vosTableEntry 1 }

vosModuleName  OBJECT-TYPE
     SYNTAX  DisplayString (SIZE (0..66))
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Name of this module."
     ::= { vosTableEntry 2 }

vosVersion OBJECT-TYPE
     SYNTAX  DisplayString (SIZE (0..100))
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "The version numbers for this copy of VOS."
```

```
     ::= { vosTableEntry 3 }

vosCPUtype OBJECT-TYPE
     SYNTAX  DisplayString (SIZE (0..100))
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "The type and model number of the CPU(s)."
     ::= { vosTableEntry 4 }

vosNumCPUs OBJECT-TYPE
     SYNTAX  INTEGER (1..32)
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Number of CPU(s) currently running."
     ::= { vosTableEntry 5 }

vosModPaging OBJECT-TYPE
     SYNTAX  INTEGER (0..100)
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Percent of paging space used for module."
     ::= { vosTableEntry 6 }

-- The red light table is a circular buffer of hardware log messages from
-- monitored modules.  This table is not organized by module, rather the
-- messages are placed in the table in the order in which they arrive.
-- This table is being implemented as a circular buffer to avoid the
-- problem of the user who enables these messages and never cleans the
-- table.  There is currently a limit of 100 log items in this table.

vosNumRedLights OBJECT-TYPE
     SYNTAX  INTEGER
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Number of entries in the red light table."
     ::= { vos 3 }

vosRedLightTable  OBJECT-TYPE
     SYNTAX  SEQUENCE OF VosRedLightTableEntry
     ACCESS  not-accessible
     STATUS  mandatory
     DESCRIPTION
         "A table of messages from the >system>hardware_log.*."
     ::= { vos 4 }

vosRedLightTableEntry  OBJECT-TYPE
     SYNTAX  VosRedLightTableEntry
     ACCESS  not-accessible
     STATUS  mandatory
     DESCRIPTION
         "A row in the red light table."
     INDEX { vosRedLightNumber }
     ::= { vosRedLightTable 1 }

VosRedLightTableEntry ::= SEQUENCE {
     vosRedLightNumber      INTEGER,
     vosRedLightModuleName  DisplayString,
     vosRedLightMessage     DisplayString
 }
```

```
vosRedLightNumber   OBJECT-TYPE
     SYNTAX  INTEGER
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Table index number assigned to this red light entry."
     ::= { vosRedLightTableEntry 1 }

vosRedLightModuleName  OBJECT-TYPE
     SYNTAX  DisplayString (SIZE (0..66))
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Name of module event occurred on."
     ::= { vosRedLightTableEntry 2 }

vosRedLightMessage  OBJECT-TYPE
     SYNTAX  DisplayString (SIZE (0..112))
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Message from hardware log."
     ::= { vosRedLightTableEntry 3 }

vosConfigFileName OBJECT-TYPE
     SYNTAX  DisplayString (SIZE (0..256))
     ACCESS  read-write
     STATUS  mandatory
     DESCRIPTION
         "Full path name of current configuration file used by agent.
         The agent always uses the file snmpd.config as its configuration
         file on startup.  Set requests on this variable have not been
         implemented.  To modify configuration file parameters, make
         the necessary changes to the snmpd.config file and stop and
         restart the agent."
     ::= { vos 5 }

vosInputQueueMessage  OBJECT-TYPE
     SYNTAX  OCTET STRING (SIZE (0..512))
     ACCESS  read-write
     STATUS  mandatory
     DESCRIPTION
         "Most recent remote console message sent by NMS to agent.
         This MIB item is the method in which remote console
         commands are entered by the NMS.  Performing a set-request
         on this MIB variable creates a new remote console command.
         The agent places each new remote console command in the
         Input Queue named below for processing by the NM*Console
         process."
     ::= { vos 6 }

vosInputQueueName  OBJECT-TYPE
     SYNTAX  DisplayString (SIZE (0..256))
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Input Queue name where the agent is to write remote
         console messages for processing by NM*Console.  This
         name is specified in the configuration file and must
         be identical to the name of the queue specified when
         starting the NM*Console process."
     ::= { vos 7 }
```

```
vosTrapCommFileName OBJECT-TYPE
     SYNTAX  DisplayString (SIZE (0..256))
     ACCESS  read-write
     STATUS  mandatory
     DESCRIPTION
         "Full path name of the current trap destination file being used
         by the agent. The agent always starts up using the file
         snmpd.trap_comm.  Set-requests on this file name cause the agent
         to reinitialize its trap destinations using the data in the new
         file name."
     ::= { vos 8 }

vosUserTrapCommFileName OBJECT-TYPE
     SYNTAX  DisplayString (SIZE (0..256))
     ACCESS  read-write
     STATUS  mandatory
     DESCRIPTION
         "Full path name of the current user trap destination file being
         used by the agent.  If the file snmpd.user_trap exists when
         the agent starts up, user traps are routed using this file.  If
         this file is not found when the agent starts up, the file
         snmpd.trap_comm will be used to specify user trap destinations.
         Set-requests on this file name cause the agent to reinitialize
         its user trap destinations using the data in the new file name."
     ::= { vos 9 }

cpuModLoad      OBJECT IDENTIFIER     ::= { cpu 1 }
cpuModUsage     OBJECT IDENTIFIER     ::= { cpu 2 }

-- The cpuModLoadTable collects load statistics by module.

cpuModLoadTable  OBJECT-TYPE
     SYNTAX  SEQUENCE OF CpuModLoadEntry
     ACCESS  not-accessible
     STATUS  mandatory
     DESCRIPTION
         "CPU load by module."
     ::= { cpuModLoad 1 }

cpuModLoadEntry  OBJECT-TYPE
     SYNTAX  CpuModLoadEntry
     ACCESS  not-accessible
     STATUS  mandatory
     DESCRIPTION
         "CPU load table structure."
     INDEX { cpuModLoadIndex }
     ::= { cpuModLoadTable 1 }

CpuModLoadEntry ::= SEQUENCE {
     cpuModLoadIndex             INTEGER,
     cpuModLoadName              DisplayString,
     cpuModLoadOneMinute         INTEGER,
     cpuModLoadFiveMinute        INTEGER,
     cpuModLoadFifteenMinute     INTEGER,
     cpuModLoadIntsOneMinute     INTEGER,
     cpuModLoadIntsFiveMinute    INTEGER,
     cpuModLoadIntsFifteenMinute INTEGER,
     cpuModLoadPFsOneMinute      INTEGER,
     cpuModLoadPFsFiveMinute     INTEGER,
     cpuModLoadPFsFifteenMinute  INTEGER
 }
```

```
cpuModLoadIndex  OBJECT-TYPE
     SYNTAX  INTEGER
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Location in table."
     ::= { cpuModLoadEntry 1 }

cpuModLoadName  OBJECT-TYPE
     SYNTAX  DisplayString (SIZE (0..66))
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Name of module."
     ::= { cpuModLoadEntry 2 }

cpuModLoadOneMinute  OBJECT-TYPE
     SYNTAX  TimeTicks
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "CPU seconds used in the previous minute by module."
     ::= { cpuModLoadEntry 3 }

cpuModLoadFiveMinute  OBJECT-TYPE
     SYNTAX  TimeTicks
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "CPU seconds used in previous five minutes by module."
     ::= { cpuModLoadEntry 4 }

cpuModLoadFifteenMinute  OBJECT-TYPE
     SYNTAX  TimeTicks
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "CPU seconds used in previous fifteen minutes by module."
     ::= { cpuModLoadEntry 5 }

cpuModLoadIntsOneMinute  OBJECT-TYPE
     SYNTAX  INTEGER
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Number of interrupts in one minute by module."
     ::= { cpuModLoadEntry 6 }

cpuModLoadIntsFiveMinute  OBJECT-TYPE
     SYNTAX  INTEGER
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Number of interrupts in five minutes by module."
     ::= { cpuModLoadEntry 7 }

cpuModLoadIntsFifteenMinute  OBJECT-TYPE
     SYNTAX  INTEGER
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Number of interrupts in fifteen minutes by module."
     ::= { cpuModLoadEntry 8 }
```

```
cpuModLoadPFsOneMinute  OBJECT-TYPE
     SYNTAX  INTEGER
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Number of page faults (user, system, and server) in one minute
         by module."
     ::= { cpuModLoadEntry 9 }

cpuModLoadPFsFiveMinute  OBJECT-TYPE
     SYNTAX  INTEGER
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Number of page faults (user, system and server) in five minutes
         by module."
     ::= { cpuModLoadEntry 10 }

cpuModLoadPFsFifteenMinute  OBJECT-TYPE
     SYNTAX  INTEGER
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Number of page faults (user, system and server) in fifteen
minutes
         by module."
     ::= { cpuModLoadEntry 11 }

-- The cpuModUsageTable shows the amount of CPU usage in percent.  This is
-- normalized by the number of processors so that the range is 0-100.  The
-- data in this table can be used to generate traps whenever overall CPU
-- usage or interrupt CPU usage reaches or exceeds user set thresholds.
-- The variables CpuUtil and IntUtil in the configuration file may be
-- used to specify the threshold for excessive CPU usage traps
-- (cpuUsageExcessive) and excessive CPU usage by interrupt traps
-- (interruptCpuUsageExcessive), respectively.

cpuModUsageTable  OBJECT-TYPE
     SYNTAX  SEQUENCE OF CpuModUsageEntry
     ACCESS  not-accessible
     STATUS  mandatory
     DESCRIPTION
         "CPU usage by module."
     ::= { cpuModUsage 1 }

cpuModUsageEntry  OBJECT-TYPE
        SYNTAX  CpuModUsageEntry
        ACCESS  not-accessible
        STATUS  mandatory
        DESCRIPTION
            "CPU usage table structure."
        INDEX { cpuModUsageIndex }
        ::= { cpuModUsageTable 1 }

CpuModUsageEntry ::= SEQUENCE {
     cpuModUsageIndex      INTEGER,
     cpuModUsageName       DisplayString,
     cpuModUsageUser       INTEGER,
     cpuModUsageSystem     INTEGER,
     cpuModUsageInterrupts INTEGER,
     cpuModUsageServer     INTEGER,
     cpuModUsageIdle       INTEGER
```

```
 }

cpuModUsageIndex  OBJECT-TYPE
     SYNTAX  INTEGER
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Location in table."
     ::= { cpuModUsageEntry 1 }

cpuModUsageName  OBJECT-TYPE
     SYNTAX  DisplayString (SIZE (0..66))
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Name of module."
     ::= { cpuModUsageEntry 2 }

cpuModUsageUser  OBJECT-TYPE
     SYNTAX  INTEGER (0..100)
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Percent of CPU utilized by user processes on this module
         during the previous minute."
     ::= { cpuModUsageEntry 3 }

cpuModUsageSystem  OBJECT-TYPE
     SYNTAX  INTEGER (0..100)
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Percent of CPU utilized by system processes on this module
         during the previous minute."
     ::= { cpuModUsageEntry 4 }

cpuModUsageInterrupts  OBJECT-TYPE
     SYNTAX  INTEGER (0..100)
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Percent of CPU utilized by interrupts on this module during
         the previous minute."
     ::= { cpuModUsageEntry 5 }

cpuModUsageServer  OBJECT-TYPE
     SYNTAX  INTEGER (0..100)
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Percent of CPU utilized by network server processes on this
         module during the previous minute."
     ::= { cpuModUsageEntry 6 }

cpuModUsageIdle  OBJECT-TYPE
     SYNTAX  INTEGER (0..100)
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Percent of time the CPUs were idle while no processes were
          ready to run during the previous minute."
     ::= { cpuModUsageEntry 7 }
```

```
-- The disk system statistics are stored in a table, one row per disk. This
-- is for all monitored modules.  Traps are sent when disk utilization
-- by the file partition or page partition reaches or exceeds limits set by
-- the agent configuration files.  The "diskSystemNumberEntries" should be
-- read first to avoid querying nonexistent rows in the table.  The
-- configuration file variables DiskUse, MinFreeRecords, PageUse, and
-- DiskAlarm are used to control trap message generation based on disk
-- information. DiskUse specifies the file partition threshold at which
-- diskFull traps are sent, MinFreeRecords specifies the threshold for
-- the diskNoFreeSpace trap, and PageUse specifies the paging file
-- partition at which pageFull traps are sent.  The DiskAlarm parameter
-- specifies how frequently these trap messages should be repeated.

diskSystemNumberEntries OBJECT-TYPE
     SYNTAX  INTEGER
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Number of entries in the disk system table."
     ::= { disk 1 }

diskSystemTable  OBJECT-TYPE
     SYNTAX  SEQUENCE OF DiskSystemEntry
     ACCESS  not-accessible
     STATUS  mandatory
     DESCRIPTION
         "Table of disk system statistics."
     ::= { disk 2 }

diskSystemEntry  OBJECT-TYPE
     SYNTAX  DiskSystemEntry
     ACCESS  not-accessible
     STATUS  mandatory
     DESCRIPTION
         "Disk system table structure."
     INDEX { diskSystemIndex }
     ::= { diskSystemTable 1 }

DiskSystemEntry ::= SEQUENCE {
     diskSystemIndex        INTEGER,
     diskModuleName         DisplayString,
     diskSystemName         DisplayString,
     diskSystemType         DisplayString,
     diskSystemSize         INTEGER,
     diskPartitionSize      INTEGER,
     diskPartitionUsed      INTEGER,
     diskPagePartitionSize  INTEGER,
     diskPagePartitionUsed  INTEGER,
     diskSystemFatalErrors  Counter,
     diskSystemDataErrors   Counter,
     diskSystemDiskReads    Counter,
     diskSystemDiskWrites   Counter,
     diskPartitionFree      INTEGER
 }

diskSystemIndex  OBJECT-TYPE
     SYNTAX  INTEGER
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Location in table."
     ::= { diskSystemEntry 1 }
```

```
diskModuleName  OBJECT-TYPE
     SYNTAX  DisplayString (SIZE (0..66))
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Name of module."
     ::= { diskSystemEntry 2 }

diskSystemName  OBJECT-TYPE
     SYNTAX  DisplayString (SIZE (0..66))
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Name of disk for module."
     ::= { diskSystemEntry 3 }

diskSystemType  OBJECT-TYPE
     SYNTAX  DisplayString (SIZE (0..66))
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Type of disk: Logical, Physical or Duplexed."
     ::= { diskSystemEntry 4 }

diskSystemSize  OBJECT-TYPE
     SYNTAX  INTEGER
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Size of disk in 4096 byte blocks."
     ::= { diskSystemEntry 5 }

diskPartitionSize OBJECT-TYPE
     SYNTAX  INTEGER
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Size of file partition in 4096 byte blocks."
     ::= { diskSystemEntry 6 }

diskPartitionUsed OBJECT-TYPE
     SYNTAX  INTEGER (0..100)
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Percent of disk partition used."
     ::= { diskSystemEntry 7 }

diskPagePartitionSize OBJECT-TYPE
     SYNTAX  INTEGER
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
        "Size of page partition in 4096 byte blocks."
     ::= { diskSystemEntry 8 }

diskPagePartitionUsed OBJECT-TYPE
     SYNTAX  INTEGER (0..100)
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Percent of page partition used."
     ::= { diskSystemEntry 9 }
```

```
diskSystemFatalErrors  OBJECT-TYPE
     SYNTAX  Counter
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Number of fatal errors for disk."
     ::= { diskSystemEntry 10 }

diskSystemDataErrors  OBJECT-TYPE
     SYNTAX  Counter
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Number of data errors for disk."
     ::= { diskSystemEntry 11 }

diskSystemDiskReads OBJECT-TYPE
     SYNTAX  Counter
     ACCESS read-only
     STATUS mandatory
     DESCRIPTION
         "The total number of disk reads since the module was loaded."
     ::= { diskSystemEntry 12 }

diskSystemDiskWrites OBJECT-TYPE
     SYNTAX  Counter
     ACCESS read-only
     STATUS mandatory
     DESCRIPTION
         "The total number of disk writes since the module was loaded."
     ::= { diskSystemEntry 13 }

--NEW in version 1.5.8
diskPartitionFree OBJECT-TYPE
     SYNTAX  INTEGER
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Number of unused 4096 byte blocks in partition."
     ::= { diskSystemEntry 14 }

-- The process statistics are stored in a table, one row per process.  This
-- is for all monitored modules.  The "psNumProcs" should be read first
-- to avoid querying nonexistent rows in the table.  Since this table is
-- indexed on the process identifier assigned by VOS when the process
-- starts up, this is a sparse table.

psNumProcs OBJECT-TYPE
     SYNTAX  INTEGER
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Number of processes."
     ::= { processes 1 }

psTable  OBJECT-TYPE
     SYNTAX  SEQUENCE OF PsTableEntry
     ACCESS  not-accessible
     STATUS  mandatory
     DESCRIPTION
         "A simplified version of what is read from
         s$get_processes_info()."
```

```
     ::= { processes 2 }

psTableEntry  OBJECT-TYPE
     SYNTAX  PsTableEntry
     ACCESS  not-accessible
     STATUS  mandatory
     DESCRIPTION
         "A row in the process table."
     INDEX { psProcessId }
     ::= { psTable 1 }

PsTableEntry ::= SEQUENCE {
     psProcessId           INTEGER,
     psModuleName          DisplayString,
     psName                DisplayString,
     psPersonName          DisplayString,
     psGroupName           DisplayString,
     psProgramName         DisplayString,
     psInvokingProcessId   INTEGER,
     psState               INTEGER,
     psPriority            INTEGER,
     psCpuTime             TimeTicks,
     psPageFaultTime       TimeTicks,
     psPageFaults          Counter,
     psWorkingSetSize      INTEGER,
     psDiskReads           Counter,
     psDiskWrites          Counter,
     psNumInstances        INTEGER,
     psNumProgramInstances INTEGER
 }

psProcessId   OBJECT-TYPE
     SYNTAX  INTEGER
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "VOS process identifier."
     ::= { psTableEntry 1 }

psModuleName  OBJECT-TYPE
     SYNTAX  DisplayString (SIZE (0..66))
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Name of module."
     ::= { psTableEntry 2 }

psName  OBJECT-TYPE
     SYNTAX  DisplayString (SIZE (0..32))
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Process name."
     ::= { psTableEntry 3 }

psPersonName  OBJECT-TYPE
     SYNTAX  DisplayString (SIZE (0..32))
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Name of user who started the process."
     ::= { psTableEntry 4 }
```

```
psGroupName  OBJECT-TYPE
     SYNTAX  DisplayString (SIZE (0..32))
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "The group name of the user who the started process."
     ::= { psTableEntry 5 }

psProgramName  OBJECT-TYPE
     SYNTAX  DisplayString (SIZE (0..32))
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "The name of the currently executing command or program,
         if any, given to the process."
     ::= { psTableEntry 6 }

psInvokingProcessId   OBJECT-TYPE
     SYNTAX  INTEGER
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "If this is a subprocess, this is the identifier of the process
         that created this process."
     ::= { psTableEntry 7 }

psState OBJECT-TYPE
     SYNTAX INTEGER {
          stopped(1),
          ready(2),
          prePage(3),
          memoryWait(4),
          waitShort(5),
          waitLong(6),
          postPurge(7),
          postPurgeReady(8),
          workingSetCalc(9),
          workingSetCalcReady(10)
     }
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Current state of the process."
     ::= { psTableEntry 8 }

psPriority OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS read-only
     STATUS mandatory
     DESCRIPTION
         "The process priority."
     ::= { psTableEntry 9 }

psCpuTime OBJECT-TYPE
     SYNTAX TimeTicks
     ACCESS read-only
     STATUS mandatory
     DESCRIPTION
         "Actual execution time for process. (1/100s second)"
     ::= { psTableEntry 10 }

psPageFaultTime OBJECT-TYPE
     SYNTAX TimeTicks
```

```
     ACCESS read-only
     STATUS mandatory
     DESCRIPTION
         "Accumulated time the processor has spent in page faults
         for this process.  (1/100s second)"
     ::= { psTableEntry 11 }

psPageFaults OBJECT-TYPE
     SYNTAX  Counter
     ACCESS read-only
     STATUS mandatory
     DESCRIPTION
         "Number of page faults for the process since it was created."
     ::= { psTableEntry 12 }

psWorkingSetSize OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS read-only
     STATUS mandatory
     DESCRIPTION
         "The number of pages in the current working set of the process."
     ::= { psTableEntry 13 }

psDiskReads OBJECT-TYPE
     SYNTAX  Counter
     ACCESS read-only
     STATUS mandatory
     DESCRIPTION
         "The number of times the process has read data from the
         disk into an input buffer since the process was created.
         Does not include page fault accesses."
     ::= { psTableEntry 14 }

psDiskWrites OBJECT-TYPE
     SYNTAX  Counter
     ACCESS read-only
     STATUS mandatory
     DESCRIPTION
         "The number of times the process has written data from an
         output buffer to disk since the process was created."
     ::= { psTableEntry 15 }

psNumInstances OBJECT-TYPE
     SYNTAX  INTEGER
     ACCESS read-only
     STATUS mandatory
     DESCRIPTION
         "The number of copies of this process name running in this
         module."
     ::= { psTableEntry 16 }

psNumProgramInstances OBJECT-TYPE
     SYNTAX  INTEGER
     ACCESS read-only
     STATUS mandatory
     DESCRIPTION
         "The number of copies of this program name running in this module.
          If this name is empty, the value will be 0."
     ::= { psTableEntry 17 }

-- The user trap table is a circular buffer of user trap messages from
-- all modules.  This table is not organized by module, rather the messages
-- are placed in the table in the order in which they arrive. These
```

```
-- messages are accessed using a virtual index, userTrapMsgsNumber, a 32
-- bit unsigned number which keeps incrementing with every new entry.
-- The number of log items retained in this table is specified by the
-- configuration file variable UserTrapNum.  If this variable is not
-- set in the configuation file, this table will default to 40 log items.

userNumTrapMsgs OBJECT-TYPE
     SYNTAX  INTEGER
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Virtual index of largest entry in user trap table."
     ::= { userinfo 1 }

userTrapMsgsTable  OBJECT-TYPE
     SYNTAX  SEQUENCE OF UserTrapMsgsTableEntry
     ACCESS  not-accessible
     STATUS  mandatory
     DESCRIPTION
         "A table of messages from the user trap queue."
     ::= { userinfo 2 }

userTrapMsgsTableEntry  OBJECT-TYPE
     SYNTAX  UserTrapMsgsTableEntry
     ACCESS  not-accessible
     STATUS  mandatory
     DESCRIPTION
         "A row in the user trap message table."
     INDEX { userTrapMsgsNumber }
     ::= { userTrapMsgsTable 1 }

UserTrapMsgsTableEntry ::= SEQUENCE {
     userTrapMsgsNumber   INTEGER,
     userTrapMsgsMessage  OCTET STRING
 }

userTrapMsgsNumber   OBJECT-TYPE
     SYNTAX  INTEGER
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "Index number assigned to this user trap message entry."
     ::= { userTrapMsgsTableEntry 1 }

userTrapMsgsMessage  OBJECT-TYPE
     SYNTAX  OCTET STRING (SIZE (0..256))
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
         "User trap queue message."
     ::= { userTrapMsgsTableEntry 2 }

END
```

Appendix

C

NMServer Trap MIB

vostrapsysmgr.mib

```
-- file:  vostrapsysmgr.mib
--
-- COMTEK Services, Inc. RFC 1225 NM*SysMgr Trap MIB for VOS
-- Release      1.5.8
-- Date         March,1997
-- Author       Steve Harris

-- Copyright 1997-2008 COMTEK Services, Inc.  All Rights Reserved.
--
-- This COMTEK Services SNMP Management Information Base Specification
-- (Specification) embodies COMTEK Services' confidential and
-- proprietary intellectual property.  COMTEK Services retains all
-- title and ownership in the Specification, including any
-- revisions.
--
-- This Specification is supplied "AS IS," and COMTEK Services makes
-- no warranty, either express or implied, as to the use,
-- operation, condition, or performance of the Specification.


-- The operation of the NM*SysMgr agent for VOS is controlled via
-- the configuration file (snmpd.config), the community file (snmpd.comm),
-- the trap destination file (snmpd.trap_comm), and the user trap
-- destination file (snmpd.user_trap).  See the User's Guide for a
-- complete description of each of these files.


COMTEK-TRAP-MIB DEFINITIONS ::= BEGIN

-- SECTION 1: Top Level Definitions

-- Imports

IMPORTS
     snmp, sysDescr          FROM RFC1213-MIB
     TRAP-TYPE               FROM RFC-1215
     comtek                  FROM COMTEK-DEFINITIONS-MIB
     comtekVosAgent, diskModuleName, diskSystemName, diskPartitionUsed,
     diskPagePartitionUsed, cpuModUsageName, cpuModUsageUser,
     cpuModUsageSystem, cpuModUsageServer, cpuModUsageInterrupts,
     psModuleName, psName, psNumInstances, psNumProgramInstances,
     psProgramName, vosModuleName, vosModPaging, vosRedLightNumber,
     vosRedLightModuleName, vosRedLightMessage
                             FROM COMTEK-MIB;

-- SECTION 2: Generic Trap Definitions
```

```
coldStart TRAP-TYPE
     ENTERPRISE        snmp
     VARIABLES         { comtekVosAgent, sysDescr }
--     STATUS          mandatory for all Comtek Agents
     DESCRIPTION
          "A coldStart trap signifies that the sending protocol entity
          is reinitializing itself such that the agent's configuration
          or the protocol entity implementation may be altered."
     ::= 0

authenticationFailure TRAP-TYPE
     ENTERPRISE        snmp
     VARIABLES         { comtekVosAgent, sysDescr }
--     STATUS          mandatory for all Comtek Agents
     DESCRIPTION
          "An authenticationFailure trap signifies that the sending
          protocol entity is the addressee of a protocol message that is
          not properly authenticated. While implementations of the SNMP
          must be capable of generating this trap, they must also be
          capable of suppressing the emission of such traps via an
          implementation-specific mechanism.

          Setting the configuation file variable AuthTrap to No
          disables these traps."
     ::= 4


-- SECTION 3: VOS NM*SysMgr Enterprise Trap Definitions

diskFull TRAP-TYPE
     ENTERPRISE     comtek
     VARIABLES      { comtekVosAgent, sysDescr, diskModuleName,
                      diskSystemName, diskPartitionUsed }
--     STATUS       mandatory for all Comtek VOS Agents
     DESCRIPTION
          "Disk system usage has reached or exceeded critical threshold.
          The configuation file variable DiskUse defines the critical
          threshold for disk usage.  The configuation file variable
          DiskAlarm defines how frequently these traps are repeated."
     ::= 0

diskFullClear TRAP-TYPE
     ENTERPRISE     comtek
     VARIABLES      { comtekVosAgent, sysDescr, diskModuleName,
                      diskSystemName, diskPartitionUsed }
--     STATUS       mandatory for all Comtek VOS Agents
     DESCRIPTION
          "Disk system usage that had reached or exceeded critical
          threshold has now gone below threshold."
     ::= 1

pageFull TRAP-TYPE
     ENTERPRISE     comtek
     VARIABLES     { comtekVosAgent, sysDescr, vosModuleName,
                     vosModPaging }
--     STATUS       mandatory for all Comtek VOS Agents
     DESCRIPTION
          "Page partition usage has reached or exceeded critical
          threshold. The configuation file variable PageUse defines
          the critical threshold for paging usage.  The configuation
          file variable DiskAlarm defines how frequently these traps
          are repeated."
```

```
     ::= 2

pageFullClear TRAP-TYPE
     ENTERPRISE    comtek
     VARIABLES     { comtekVosAgent, sysDescr, vosModuleName,
                     vosModPaging }
--     STATUS      mandatory for all Comtek VOS Agents
     DESCRIPTION
          "Page partition usage that had reached or exceeded critical
          threshold has now gone below threshold."
     ::= 3          -- 0x03

cpuUsageExcessive TRAP-TYPE
     ENTERPRISE    comtek
     VARIABLES     { comtekVosAgent, sysDescr, cpuModUsageName,
                     cpuModUsageInterrupts, cpuModUsageUser,
                     cpuModUsageSystem, cpuModUsageServer }
--     STATUS      mandatory for all Comtek VOS Agents
     DESCRIPTION
          "Excessive CPU utilization by all processes and interrupts.
          The configuration file variable CpuUtil defines the threshold
          for this trap."
     ::= 4

interruptCpuUsageExcessive TRAP-TYPE
     ENTERPRISE    comtek
     VARIABLES     { comtekVosAgent, sysDescr, cpuModUsageName,
                     cpuModUsageInterrupts  }
--     STATUS      mandatory for all Comtek VOS Agents
     DESCRIPTION
          "Excessive CPU utilization by interrupts.  The configuation file
          variable IntUtil defines the threshold for this trap."
     ::= 5

redLightEvent TRAP-TYPE
     ENTERPRISE    comtek
     VARIABLES     { comtekVosAgent, sysDescr, vosRedLightNumber,
                     vosRedLightModuleName, vosRedLightMessage }
--     STATUS      mandatory for all Comtek VOS Agents
     DESCRIPTION
          "Item added to >system>hardware_log.?.  The configuation file
          variable RedLight enables and disables these traps."
     ::= 6

userQueueMessage TRAP-TYPE
     ENTERPRISE    comtek
     VARIABLES     { comtekVosAgent, sysDescr, userTrapMsgsNumber,
                     userTrapMsgsMessage }
--     STATUS      mandatory for all Comtek VOS Agents
     DESCRIPTION
          "Item added to agent trap queue by other process.  The
          configuation file variable InputQueue defines where the
          agent looks for user data to be sent as traps."
     ::= 7

criticalProcessMissing TRAP-TYPE
     ENTERPRISE    comtek
     VARIABLES     { comtekVosAgent, sysDescr, psModuleName, psName,
                     psNumInstances }
--     STATUS      mandatory for all Comtek VOS Agents
     DESCRIPTION
          "Too few instances of this process name in process table.  The
          configuation file variable ProcTrap defines critical processes."
```

```
     ::= 8

criticalProgramMissing TRAP-TYPE
     ENTERPRISE    comtek
     VARIABLES     { comtekVosAgent, sysDescr, psModuleName, psProgramName,
                     psNumProgramInstances }
--     STATUS      mandatory for all Comtek VOS Agents
     DESCRIPTION
          "Too few instances of this program name in process table.  The
          configuation file variable ProgTrap defines critical programs."
     ::= 9

diskNoFreeSpace TRAP-TYPE
     ENTERPRISE     comtek
     VARIABLES      { comtekVosAgent, sysDescr, diskModuleName,
                      diskSystemName, diskPartitionFree }
--     STATUS       mandatory for all Comtek VOS Agents
     DESCRIPTION
          "Critical threshold for free space on disk has been reached.
          The configuation file variable MinFreeRecords defines the
          critical threshold for this trap.  The configuation file variable
          DiskAlarm defines how frequently these traps are repeated."
     ::= 10

diskNoFreeSpaceClear TRAP-TYPE
     ENTERPRISE     comtek
     VARIABLES      { comtekVosAgent, sysDescr, diskModuleName,
                      diskSystemName, diskPartitionFree }
--     STATUS       mandatory for all Comtek VOS Agents
     DESCRIPTION
          "Disk free space has now gone above critical threshold."
     ::= 11

END
```

Appendix

D

Monitoring Log Files

The Command Line

Log Files, sometimes know as Error Journals, can be monitored and their records sent to the SNMP Manager as traps through the NMServer User Trap Queue. If desired, the records may be filtered by the user to either send records if they match at least one filter entry, or to exclude sending records if they match at least one entry. The Log File default is the VOS (master_disk)>system>syserr_log unless the user supplies a Log File name. Many Log Files may be monitored at the same time by starting multiple processes with different Log File names in the **-logfile** parameter.

Be sure to create each process instance with a unique process_name and .out file to avoid VOS conflicts. In almost every case it will be advisable to start the process by using a .cm file because the various parameters can be specified permanently without operator input. The program uses the s$parse_command so if you wish to debug before creating a .cm file, you can issue **monitor_log -form** on the command line to enter the various parameters interactively.

The following parameters apply to the program named **monitor_log.pm**:

license_key – This must be the first parameter. The value specified for this parameter is a 18 character license string which is provided by COMTEK. You will be supplied one of three types of license keys: temporary, permanent system or permanent enterprise. Temporary license keys may be used on any VOS module but are only valid for a specified period of time. These types of license keys are issued to allow for evaluation of COMTEK products. Permanent system license keys are issued one license key per VOS module. A different license key must be used for each VOS module that the monitor_log process is to run on. Permanent enterprise license keys operate on any VOS module.

trap_queue_path -- This must be the second parameter. It is the relative or full path name of the user trap queue. If a relative path is used, its expanded full path name must exactly match the name specified in the

NMServer snmp.config file. An example of a full path name is %system1#d02>snmp>trap_Q. This parameter and the license key parameter are the only input needed if all you want to do is monitor the default Log File (master_disk)>system>syserr_log. Note that if you use a different queue name from the one specified in the snmp.config file, the program will still work and will send messages to the queue named in *trap_queue_path*. This feature could be used to send "traps" to any kind of non-SNMP Manager that the user supports.

Optionally, the user can specify any of the following parameters in any order.

-trap_tag *string* -- A string up to sixteen characters long that is prepended to each User Trap. Its use in intended to help the SNMP Manager Station determine from which log this Trap originated. If no tag is wanted, just blank out the field. The default is the value "syser".

-filter_file *path_name* -- Specify this parameter if the user has a filter file to be used against the records in the File. See the explanation below on how to create a filter file.

-**exclude_if_match** *yes* | *no* -- If a filter file is used, this value determines if a trap is to be sent, or not, based on whether the filter file had a matching entry. The default value (yes) is to exclude records matching any entry in the filter file.

-**logfile** *path_name* -- If the user wishes to monitor a Log File other than the VOS (master_disk)>system>syserr_log, the relative or full path name of that File is specified here.

-position_at_eof *yes* | *no* -- If it is desired to position the Log File at the end_of_file when the file is opened at process start up, then set this parameter to yes (the default). To start at the beginning of the file and read all the records that are in it, set the value to no.

-**logfile_event** *path_name* -- Specify this parameter if the user's Log File has an event associated with it. Otherwise, the monitor_log.pm: will read the File every **-wait_time** seconds to check for new entries.

-midnight *yes* | *no* -- This parameter specifies if the Log File name is created anew at midnight or if it is created only once. If the Log File name is created each midnight, the new name will be suffixed by the monitor_log.pm with a period (dot) followed by the date of the form YY-MM-DD; an example of this Stratus convention is >system >syserr_log.98-03-20. The default (no) is that the Log File name is created by the user's program only once.

-wait_time *seconds* -- This parameter is used when the user's File has no event associated with it. The value is the time, in seconds, that the **monitor_log.pm** will wait before the File will be read for new records. Warning: if you make this value zero, the program will constantly read the file. The default value is ten seconds.

-local_time *yes* | *no* -- Use local time or Greenwich Mean Time (GMT) if Log Files are created at midnight. The default (yes) is local time.

-trap_throttle *traps_per_second* -- This specifies the maximum number of traps per second that the program should send; it regulates how much data goes through the customer's UDP Path to the Manager station for each time unit. If this parameter is omitted, traps will be sent as fast as possible.

-heartbeat *minutes* -- Setting this to a non-zero value causes a heartbeat message to be sent every N minutes. This message consists of the **trap_tag**, the literal " heartbeat ", the version number of the **monitor_log** program, and the full path name of the user's trap_Q. The default is zero minutes which signifies no heartbeat message is to be sent.

About Filter Files

The filter file identifies the Log File messages that are to be sent to the SNMP Manager or are to be discarded rather than sent to the Manager depending on the setting of -**exclude_if_match**. Each Log File message that is received by the **monitor_log.pm** is compared to the entries in the file identified by the user in the command line argument **-filter_file**. If **-exclude_if_match** is specified as *yes* and if the message is found to match one of the entries in the filter file, the message is discarded. If the message does not match any of the entries in the filter file, the message is sent as a User Trap. If **-exclude_if_match** is set to *no*, the opposite applies so that only messages matching one of the entries in the filter file are sent as Traps and all other messages are discarded.

The filter file permits the use of the * wildcard. This wildcard character is used to match one or more characters in the message. The wildcard character must be used carefully to ensure proper filtering. Placing wildcards at the beginning or end of a filter string requires that the string be embedded somewhere in the "middle" of the Log File message. The following is an example of a filter file:

PreLogin
*TEST MESSAGE
Operator * has been *abled, username *

The first entry would cause any Log File message containing the word "PreLogin" to match. However, due to the presence of the leading and trailing wildcards, if the word "PreLogin" were at the beginning or end of the Log File message, it would not cause a filter match. The second entry would cause any message ending with the words "test message" to match. The third line illustrates the use of multiple wildcards within the same message. This entry would cause both messages "Operator X has been enabled, username JONES" and "Operator Y has been disabled, username SMITH" to be matched. Note that entries in the filter file are case insensitive and that each entry must be left justified.

A maximum number of 50 non-commentary lines are permitted in a filter file and each line may contain up to 5 substrings delimited by wildcards.

If special non-printing characters are in the message, they are stripped out. Here are all the different forms of escape sequences subject to stripping:

Form 1: ESC <0x20-0x2F>+ <0x30-0x7E>
Form 2: ESC ; <0x20-0x2F>+ <0x30-0x7E>
Form 3: ESC ? <0x20-0x2F>+ <0x30-0x7E>
Form 4: ESC O <0x20-0x2F>+ <0x40-0x7E>
Form 5: ESC [<0x30-0x3F>+ <0x20-0x2F>+ <0x40-0x7E>
(also called a control sequence)

Examples of Command Macro (.cm) Files

Here are some examples that run **monitor_log.pm** by using a Command macro to set its parameters.

mon_syserr.cm is the simplest example possible. It uses all the default parameters so it monitors the syserr_log.yy-mm-dd file. The only required parameter other than the license key is the SNMP User Trap Queue name (trap_Q), and it is supplied in the relative form; this means that the directory in which the Queue resides is the current_dir from which **monitor_log.pm** is executing.

```
&attach_input
start_process  'monitor_log  YZ2FMS5MBT7WZAC9P5  trap_Q'
```

mon_syserr1.cm shows the use of several options. It uses the default parameters for **-logfile** so it monitors the syserr_log.yy-mm-dd file. The 18 character license key is specified as the first parameter, as required.

The second parameter, the required SNMP User Trap Queue name parameter (filtq) is supplied in the relative form; this means that the directory in which the Queue resides is the current_dir from which **monitor_log.pm** is executing.

The optional -trap_tag causes the 3-character value "flt" to be placed in front of the trap to assist the SNMP Manager in deciding where the trap came from. The -no_exclude_if_match tells the program to create a trap only if there is a match with any filter in the -filter_file named "filt". (Note that in VOS a *yes | no* is a CYCLE parameter; the negation of a CYCLE parameter involves putting the prefix no_ before the parameter name.)

After the ending " ' " there are two parameters that go with the start_process:

process_name don1 gives this instance of **monitor_log.pm** a process name of "don1".
-output_path don1.out defines a unique .out file for this process.

These parameters are necessary when running more than one copy of **monitor_log.pm**.

```
&attach_input
start_process  'monitor_log  YZ2FMS5MBT7WZAC9P5 filtq   &+
               -trap_tag flt              &+
               -no_exclude_if_match &+
               -filter_file filt'   &+
               -process_name don1 -output_path don1.out -privileged
```

mon_remote.cm is similar to **mon_syserr1.cm** above. The 18 character license key is specified as the first parameter, as required. The second required parameter is the SNMP User Trap Queue name (trap_q), and it is supplied in its full-path form; this means that the directory in which the Queue resides need not be the current_dir from which **monitor_log.pm** is executing.

It illustrates the use of the -logfile parameter to specify a log-file other than the syserr log. This example uses the >system>remote_maint_log but no -logfile_event can be specified since the VOS system has not created one. Therefore, the -wait_time value will be used to read the log periodically; the default value of 10 seconds applies since there is no override in the .cm. Because the log is created anew each day (if someone logs in), -midnight must be used to make **monitor_log.pm** create a date appendage so that the name used is remote_maint_log.yy-mm-dd.

The optional -trap_tag causes the 7-character value "remote1" to be placed in front of the trap to assist the SNMP Manager in deciding where the trap came from. The -exclude_if_match tells the program to create a trap only if there is no match with any filter in the -filter_file named "remote_filt". The -heartbeat causes a heartbeat message to be sent every 20 minutes.

```
&attach_input
start_process  'monitor_log YZ2FMS5MBT7WZAC9P5 &+
                %nap1#d01>trap_q  &+
                 -trap_tag  remote1              &+
                 -exclude_if_match          &+
                 -logfile >system>remote_maint_log   &+
                 -midnight           &+
                 -filter_file remote_filt  -heartbeat 20'   &+
                 -process_name don2 -output_path don2.out
```