10

Events

The Events Configuration features allow you to set up tests that collect information about your UNIX systems that you can use to assist in tasks such as system tuning, load balancing, resource planning, and upgrade analysis. This chapter describes the basics of building testtab files. For an overview of events capabilities and features and information about using the Events Configuration window (Figure 10-1) to add, modify, and delete tests using the Events graphical user interface, refer to Chapter 5, “Monitoring the Network,” in the *ENlighten/DSM User Guide*.

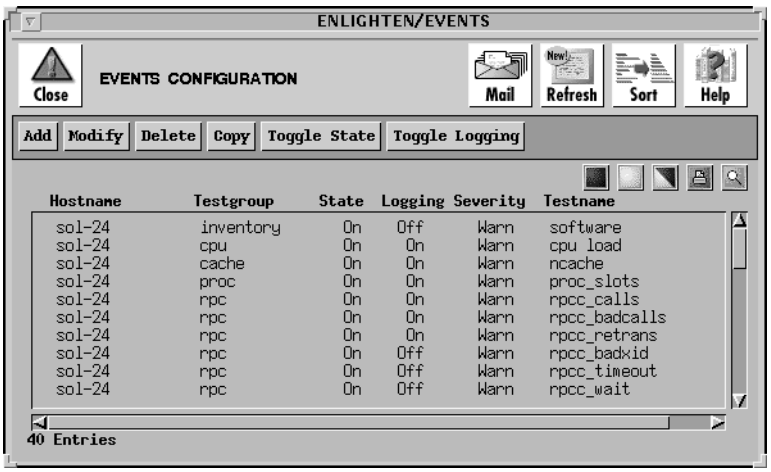

Figure 10-1 Events Configuration

Events is a distributed systems management feature that provides for the unattended monitoring of your UNIX systems. It provides extensive automated data collection for use by both local System Administrators and Network Managers. Events can help you predict when a problem is about to occur, where it will occur, and report the event while taking user-definable corrective action.

How EVENTS Works

You can use the data collected by Events to assist in tasks, such as:

- System Tuning
- Load Balancing
- Resource Planning/Justification
- Upgrade Requirement Analysis

Events collects this data by monitoring the following:

- Memory Subsystems
- Individual Files
- Directory Queues
- File Systems
- Printer Queues
- Critical Processes

- Network Statistics
- Hardware Inventory
- Software Inventory
- User Provided Data

An appropriate message can be sent to Network Managers, System Managers, or both when an alarm condition occurs. Events can send alarms using one or more of the following methods:

- as SNMP (Simple Network Management Protocol) trap messages
- as email
- as PEP (Programmable Events Processor) messages

Events can also pass the alarm to a process you've defined for possible corrective action. You can specify the same process for all tests or a separate process for each test.

Inventory Tracking

One of the unique features of Events is its hardware inventory tracking mechanism. At system start-up, Events assembles an inventory record, including hard disks, tape drives, RAM, network interfaces, and software where applicable. If a list exists from a previous start-up, then the two lists are compared. Additions and deletions are reported via email and to the EMD.

On most systems, Events also includes a software inventory. The software inventory process is done similarly to the hardware inventory tracking mechanism. You can set how frequently the software inventory is generated in the `testtab` file. See "The testtab File" on page 10-10 for more details.

Communications

Events communicates to SNMP management systems via SNMP. As an SNMP agent, Events initiates error messages and alerts, and provides information to the SNMP management system. By responding to inquiries from the SNMP management system, Events makes workstation monitoring an interactive process.

Practical Use

For the System Manager, Events is a process that runs in the background and looks after the system. The System Manager can specify whether or not measured values are stored to a database, who should be notified if a test fails, and how to notify someone when a test fails for each test that is performed. Also, other tests can easily be added to the built-in tests suite.

For the Network Manager, Events is an SNMP agent that emits enterprise-specific traps to notify the appropriate Network Manager of a failed test condition. All tests are manageable via SNMP. MIB II (Management Interface Base) is also supported.

Alarm Thresholds

Each Events test measures some numeric value. It is easier to figure out which alarm threshold is the most appropriate if you know what the value represents. The following examples use tests that are the most often misunderstood.

The paradigm is:

1) Select a test.
2) If there is an alarm threshold specified or if logging is enabled:
    - Execute the test (go and count something).
    - Log the count if it changed significantly from the last logged value.
    - Compare the count with each alarm threshold.
    - Send alarms if any of the thresholds were exceeded.
3) Schedule the time of the next test.

File Clamping

This test looks in ASCII log files for the recent addition of message types you have defined. Though an alarm message may contain text from the monitored file, the alarm itself is not the message text; it's the number of offending messages found.

Since the test value (count) is the number of messages found (regular expression matches), you must set one of the alarm thresholds if you desire an alarm. By setting the High Level limit equal to one, an alarm occurs every time an offending message is found.

File Accessed

This test checks the time stamp associated with the file. The time stamp is actually a number. If the number increases, the file has been accessed. The number should never decrease. A decrease in value would be suspicious and may indicate a security breach.

Process Instances

This test counts the number of processes containing the same name that are running.

Process Size

This test looks at the size of the processes you've specified. If more than one process of the specified name is found, only the size of the last process found is reported.

Components

Events consists of the following parts:

- AgentENL—An SNMP Agent for sites not currently running a multi-MIB SMUX (SNMP Multiplexer) compliant agent.
- AgentMon—A subagent required for each workstation you want to monitor.
- EventsCli—A command line interface syntax to the Events management framework. You can use this to generate events from your third-party applications.

Component Relationship

AgentMon performs all tests. When an alarm condition occurs, AgentMon may notify someone via traditional methods, such as email specified by the local user, and/or it may also notify network management via an enterprise-specific SNMP trap.

AgentENL is an interface between the AgentMon SubAgent and the network management application, such as SunNet Manager or HP OpenView. AgentENL must be started before AgentMon starts, even if you do not intend to use the SNMP interface.

You can also use EventsCli with any current monitoring scripts and programs to handle any special needs that AgentMon can't cover. By calling EventsCli from the alarm notification section of your script, you gain much greater flexibility and control over how the alarm is treated. EventsCli will send an SNMP trap, it will log the alarm to the EMD, and it will notify PEP of its activity. Refer to Appendix F, "Events Commands," for more information.

Standards Compliance

Events complies with the following Internet standards:

RFC 1155	Structure and identification of management information of TCP/IP-based internets: MIB-II
RFC 1157	Simple Network Management Protocol (SNMP)
RFC 1212	Concise MIB definitions
RFC 1213	Management Information Base for network management of TCP/IP-based internets: MIB-II
RFC 1215	Convention for defining traps for use with the SNMP
RFC 1227	SNMP MUX (SMUX) protocol and MIB

Configuration

Events has three different types of stored tests you can configure:

1) Group Tests

   Each of these tests is predefined with a purpose and a name, such as cpu load, cache_usr, or kernel_traps. See "Group Tests" on page 10-13 for more information.

2) Item Tests

   These tests are File, Directory, and Processes. Each of these three tests types is predefined with a purpose and may have further subcategories of tests. Each test name will be the name of the particular item being tested.

   You can have as many of each of the three types of tests as you want. For example, you could have 18 File tests, 24 Directory tests, and 26 Processes tests. See "Item Tests" on page 10-32 for more information.

3) API Tests

   There are six of these tests; each is predefined with a name and nothing else. You can use these test names, api1 through api6, to create your own tests with shell scripts, SQLs, or even compiled programs, and incorporate them into Events. See "API Tests" on page 10-37 for more information.

Configuring Tests

All the information that comprises an Events test is described in (a file called) the `testtab` file as an entry. Each entry must begin with a test name, and the subsequent lines must contain the test's parameters. Tests are not required to have any associated alarm thresholds; you may, for instance, want to enable logging.

A test is ignored if:

- there are no alarm thresholds and/or
- logging is not enabled.

You can disable a test by using the `off` parameter. AgentMon modifies the `testtab` file whenever the configuration is changed via SNMP or the Events GUI.

If the default testtab values are acceptable, the simplest test entry need only contain the following:

- a valid test name AND
- an alarm threshold or the logging is enabled.

Tests, or entries, can be added, modified, or deleted using any editor or the Events GUI. Try to use the Events GUI where possible. See the section "Monitoring UNIX Systems with Events" in Chapter 5, "Monitoring Your Network System," of the *ENlighten/DSM User Guide*.

Deleting the entry for a built-in test will not turn the test off, but merely makes that test run with the default values. To turn a test off, reconfigure the entry by changing the `on` capability to `off`.

In reality, the `testtab` file need only contain changes from the default settings.

Data Logging

Logging to the EMD (Enterprise Management Database) is enabled by specifying the parameter `log` for each test from which you wish to collect data. Also, you can specify a value for `delta` for each test you want to log. Logging will then occur if the previously logged value varies by more than `delta` units from the current value. Setting `delta=0` results in the measured value always being logged.

Delta

You can use `delta` to set your tests so they monitor more frequently, log less often, and still not lose any logging information. This is most useful in one of two situations:

- monitoring an object whose value seldom changes
- monitoring an object whose value has more precision than a log needs

An example of the first case: Monitoring a file that seldom changes size

With traditional logging (log every measurement), if you monitor this file once per minute and its size changes only once each hour, the other 59 out of 60 loggings are identical. With `delta` logging, you can specify that logging should only occur if the value changes significantly.

In this case, you could specify you want logging to occur only if the file size changes by one or more bytes (set `delta=1`). Now the log only contains the *same* two critical points (and not the other 58 instances of the repeated measurement).

An example of the second case: Monitoring an object whose value has a greater precision than is required

Suppose you are monitoring a filesystem's free space and logging the data so you could later use the data to help predict when to buy additional storage. If you set `delta=1000`, then logging will only occur when the available disk space changes by `>1000 units` from the previously logged value. In effect, you are specifying the resolution of your data.

Alarm Messages

If the test generates an alarm message, logging will always occur (even if the test has logging disabled). AgentMon also logs itself each time it is started, restarted (warm start), or normally terminated.

Automated Corrective Action

You can use a command parameter, which is a pathname, in the `testtab` entry for a test to perform some automated corrective activity. The pathname is assumed to be the name of a user-provided program. When an alarm condition occurs, the named process is started with the following arguments (listed in order):

1) Test name
2) Value
3) Unit of measure
4) Alarm type
5) Time of measurement

Examples of corrective action scripts can be found in the subdirectory `$ENLIGHTEN/policy`.

You can also include the `pep` capability in the `testtab` entry to notify PEP of the alarm.

The testtab File

The file `$ENLIGHTEN/config/testtab.hostname` contains the entries defining AgentMon's current configuration. If this file does not exist, it will be created upon start-up. If AgentMon's configuration is changed via SNMP or the Events GUI, this file will be rewritten to reflect those changes. If the file is manually edited, AgentMon will do a "warm" restart and configure itself in accordance with the new contents.

The rest of this chapter details the basics of building a testtab file. You can also use the Events GUI to easily build and alter the configuration of any of your tests. See the section "Monitoring UNIX Systems with Events" in Chapter 5, "Monitoring Your Network System," of the *ENlighten/DSM User Guide*.

Configuration File Parameters

Each test, or `testtab` entry, is composed of a test name and optional parameters that serve as "keywords" to define the scope and behavior of the test. The parameters are:

Name	Type	Default	Comment
testfreq	int	5	Test interval, in minutes.
alarmfreq	int	60	Minimum time in minutes between alarms. For process monitoring, the default is zero.
command	str	nonw	Pathname of any processes to start when an alarm occurs.
mailer	str	/bin/mail	Program that will deliver the alarm if a username was given for 'notify'.
notify	str	root	Where alarms are sent. May be set to 'nobody' for no notification, except on Solaris 2.x systems.
log	boolean	false	Indicates logging is enabled. Alarms are always logged.
!log	boolean	true	Turns off logging.
delta	int/float	0	If logging is enabled, the most recent value measured will be recorded if it differs by at least this amount from the previous value.
pep	boolean	varies	Notify PEP when an alarm occurs.
!pep	boolean	varies	Do not notify PEP when an alarm occurs.

The following is an example test entry using some of these parameters.

```
cpu load |:\
   :on:testfreq=1:alarmfreq=60:mailer=/bin/mail:notify=root:log:\
   :high=5.0:units=units:delta=50:pep:command=/policy/myproc:\
   :/* end cpu load */:
```

Refer to Appendix G, "Sample Events Files," for a sample of a complete `testtab` file.

Alarm Set Points

Alarm set points are "keywords" used in the `testtab` file to set alarm thresholds. An alarm set point can be specified as an absolute set point, a percentage change, or as an incremental set point. AgentMon checks the conditions and sends alarms based on the following order of precedence:

Name	Type	Default	Comment
high	int/float	0	Absolute set point—high-level alarm set point.
low	int/float	0	Absolute set point—low-level alarm set point.
+rate	float	0.0	Percent change set point—positive rate change set point.
-rate	float	0.0	Percent change set point—negative rate change set point.
+jump	int/float	0	Incremental set point—positive shift set point.
-jump	int/float	0	Incremental set point—negative shift set point.
age	int	0	For monitoring directory queues only, in minutes.

Most tests measure integer values; others measure floating point values. Set point parameters should be given by using the same variable type as the test result. Failure to do so, however, will only result in rounding errors.

The jump and rate thresholds compare the current test value with the last measured value (change over time). For process monitoring, the process built-in first/last alarm is checked first, then the alarm set points are checked in the order listed in the preceding table (see "Processes Tests" on page 10-36 for more details).

Group Tests

This section describes all the modifiable tests that can appear in the `testtab` file. Each test name is listed with the corresponding Events MIB Group name and a brief description.

Refer to Appendix H, "O/S Compatibility," to determine if any particular test is supported on your operating system.

General Tests

The following table shows the general Events tests. These tests are only available to SNMP-based network management software.

Test Name	MIB Group	Description
N/A	Limits	This group is accessed via your NMS (Network Management Software) and allows the network manager to turn tests on or off, adjust alarm thresholds, and control data logging. Changes made in this group take effect immediately and become part of AgentMon's new start-up configuration. If an item does not appear in the limits group, then the network manager cannot edit it via the NMS. The path for perspective is: `testtab file > limits group > NMS GUI > Network Manager`
N/A	TrapManage	The SNMP protocol used by AgentMon and many NMS applications defines an alarm messaging facility called TRAPs. AgentMon has several traps and allows the network manager to enable and disable them. Changes made in this group take effect immediately and become part of AgentMon's new start-up configuration.

O/S Tests

The following table shows the O/S tests.

Test Name	MIB Group	Description
cpu load	OS	CPU load average, one minute average. This shows the average number of jobs in a run queue. Default Alarm: high=5.0 This is a relative indication of how busy the system is. Some slowness in system response may be noticed when the load exceeds a value of approximately 5.0. Interactive use, like editing, can become aggravating when loads are heavy. A low value of cpu load factor is preferred. No value is too low. A high value indicates that the system is being overworked. The most common causes are: • Too many people logged on • One or more CPU intensive programs running • System needs more RAM • Something causing an excessive number of interrupts When any cpu load alarm threshold has been breached, a TRAP PDU will be sent to the Network Management System (NMS) if the NMS has SET this trap to ENABLE (see the TrapManage Group). The TRAP message will include the current value of cpu load factor.
cpu user	OS	Percentage of time spent handling user processes.
cpu idle	OS	Percentage of time the CPU is idle.

Test Name	MIB Group	Description
cpu kernel	OS	Percentage of time spent handling system processes.
cpu wait	OS	Percentage of time spent waiting for I/O procedures to complete.

The following OS Group tests may be of occasional usefulness to your local UNIX performance expert and can also be an excellent troubleshooting aid for certain kinds of problems. These tests do not normally need to be active.

Test Name	MIB Group	Description
kernel_cxt	OS	Number of kernel context switches since the last reboot.
kernel_traps	OS	Number of kernel traps since the last reboot.
kernel_syscalls	OS	Number of kernel mode system calls made since the last reboot.
kernel_devints	OS	Number of device interrupts since the last reboot.
forks	OS	Number of Forks since the last reboot.
vforks	OS	Number of VForks since the last reboot.
fork_pages	OS	Number of Forked Pages since the last reboot.
vfork_pages	OS	Number of VForked pages since the last reboot.

File System Tests

The following table shows the File System tests.

Test Name	MIB Group	Description
/<fs> blocks free where **<fs>** is a File System name	File System	Amount of free space on each logical (partition) disk drive. For example: `/usr blocks free` For each file system, the default low threshold limit for blocks free value is set to 10% of that file system's size (in 512-byte blocks). Blocks free refers to the number of blocks available on the disk. There will be as many tests as there are filesystems. These tests are created dynamically. Each filesystem is automatically discovered at program start-up time. Alarm thresholds are also automatically computed. Since near-full disks are typical, the default low-level limit for this test will be adjusted if the filesystem is already in an alarm condition at installation. When the amount of free space has breached an alarm threshold, a TRAP PDU will be sent to the NMS if the TRAP for that particular filesystem has been ENABLED (see the TrapManage Group).

Test Name	MIB Group	Description
/<fs> inodes free where **<fs>** is a File System name	File System	Maximum number of new files that can be added to the disk. For example: `/usr inodes free` For each filesystem, the default low threshold limit for inodes free is set equal to 10% of the total allocated for that system. Since near-full disks are typical, the default low level-limit for this test will be adjusted if the filesystem is already in an alarm condition at installation. When the number of free inodes has breached an alarm threshold, a TRAP PDU will be sent to the NMS if the TRAP for that particular filesystem has been ENABLED (see the TrapManage Group).

Printer Tests

The following table shows the Printer test.

Test Name	MIB Group	Description
printers	Printer	Reports changes in printer status for each monitored printer. By default, only local printers are monitored. An alarm will be sent each time the monitored printer changes state. Some printers are smarter than others, so an alarm could consist of anything from "not printing" to "out of toner." At start-up time for AgentMon, printers are automatically discovered and the tests are automatically configured. When the status of a printer changes state, a TRAP will be sent if the NMS has ENABLED the trap for that particular printer (see the TrapManage Group). The TRAP message will include the printer's name and a description of the new status. For SunOS 4.1.3 only: To monitor a remote printer, the printer must be defined in `/etc/printcap` and contain the boolean capability: "`enlightened`".

Process Tests

The following table shows the Process test.

Test Name	MIB Group	Description
proc_slots	Process	Number of additional processes that may be started. The default low-level alarm threshold for this is set to a value equivalent to 20% of the total number of process slots for which your kernel was configured. This test refers to the maximum number of new processes, applications, and programs that can be started (assuming other adequate resources exist). At program start time, AgentMon computes an alarm threshold based on your system's resources. A large number indicates you have relatively few programs running and can have many more started. A small number, especially one that became small quickly, could indicate a problem is developing. When this alarm occurs, you must act quickly to find the cause before the number of available slots reaches zero. When the number of available slots reaches zero, there is nothing to do but reboot. Even the simplest commands will fail to load. When the number of available slots breaches the alarm threshold, a TRAP PDU will be sent if the NMS has ENABLED this trap. The TRAP message includes the current number of available slots.

Inventory Tests

The following table shows the Inventory tests.

Test Name	MIB Group	Description
hardware	Inventory	At start-up, an inventory list of the host machine is made. This is a list of display strings listing the hardware items found in the kernel at boot time. Each listed item is a device name followed by a description. The list of hardware is stored in the text file $ENLIGHTEN/data/hardware.`hostname`/`inventory`. For each start-up, a new list is made and compared to the previous one, if it exists. Whenever the current inventory list differs from the previous list, an alarm message is issued indicating the detected hardware addition(s) and/or subtraction(s). There are no alarm thresholds or TRAPS associated with this group. *This test cannot be turned off.* The following are example inventory files. For SunOS: mach: Sun 4/40 ID# 289102334 sdo: sd0: Hard Disk, 3662 RPM, Intrlv 1:1 450Mbytes zso: zs0: Serial com chip (Zilog 8530) RAM: RAM: 8335360 bytes OS: sunOS 4.1.3

Test Name	MIB Group	Description
hardware (cont'd)	Inventory	For HP/UX: CPU FPU CORE-GRAPHICS-L on /dev/diag/crt100 CORE-SCSI SEAGATETEST11200N on /dev/diag/dsk/c201d6 CORE-LAN on /dev/diag/lan202 CORE-RS232-1 CORE-CENT CD-NB-AUDIO PC-FLOPPY-INTERFACE FD235HG on /dev/diag/pcflpyc20ad1 CORE-PS2-1 CORE-PS2-2 RAM: 33554432 bytes OS: HP-UX A. 09.05 IP: 129.1.2.130
software	Inventory	Similar to hardware, but only detects software installed by 'custom' and/or 'pkgadd'. This test can be run periodically. It can also be turned off.

RPC Tests

The following table shows the Remote Procedure Call (RPC) tests. These tests report various types of errors that can occur with the RPC protocol. This information is not normally required, but can be very useful when tracing network problems related to RPCs. Alarm thresholds may be set for each test, but there are no TRAPs associated with them.

Your local network specialist and O/S provider can provide more specific information about these tests. Specifics vary from O/S to O/S and most O/Ss do not support everything on this list.

The RPC statistics are *not* available on HP/UX.

Test Name	MIB Group	Description
rpcc_calls	RPC	Number of client RPC calls since the last reboot.
rpcc_badcalls	RPC	Number of bad client RPC calls since the last reboot.
rpcc_retrans	RPC	Number of client RPC retransmissions since the last reboot.
rpcc_badxid	RPC	Number of unexpected packets received (client).
rpcc_timeout	RPC	Number of timeouts (client) since the last reboot.
rpcc_wait	RPC	Number of client waits since the last reboot.
rpcc_newcred	RPC	Number of times client authentication refreshed since the last reboot.
rpcc_timers	RPC	Number of client timers.
rpcs_calls	RPC	Number of server calls received since the last reboot.
rpcs_badcalls	RPC	Number of server calls rejected since the last reboot.
rpcs_nullrecv	RPC	Number of server calls not available, though received.

Test Name	MIB Group	Description
rpcs_badlen	RPC	Number of server truncated packets received since the last reboot.
rpcs_xdrcall	RPC	Number of server undecodable headers since the last reboot.

VM Tests

The following table shows the Virtual Memory (VM) tests. The virtual memory system uses a portion of disk, called SWAP, as though it were RAM memory. Virtual memory is organized into "pages," typically 4096 bytes per page. This size varies greatly from system to system.

A machine that frequently runs out of virtual memory often requires more RAM. Another solution may be to off-load some of its work to other, less burdened systems. When the number of available virtual memory pages breeches an alarm threshold, a TRAP will be sent to the NMS if the NMS has ENABLED this trap (see the TrapManage Group). The TRAP message includes the current number of pages free.

The VM group is *not* available on HP/UX systems.

Test Name	MIB Group	Description
vm_locked	VM	Number of virtual memory pages currently locked. The default alarm is: the high limit is set to an integer value equal to 80% of the total number of pages on your system. Special note for SCO systems: This is initially set to ((total memory+total swap) -vmClaimed). This indicates the amount of free pages for user page storage. As this value nears zero, processes begin to fail as the malloc() and calloc() system calls refuse new memory allocation requests.
vm_claimed	VM	Number of virtual memory pages claimed. Special note for SCO systems: This tracks the number of memory pages not currently "locked down." This represents all of memory minus whatever the kernel is using. Typically, this value starts at some value and then decreases slightly for awhile. If it continues to decrease, or decreases by a large increment, then there is probably a memory leak in the kernel or in a device driver.

Test Name	MIB Group	Description
vm_free	VM	Number of free vm blocks (30-second moving average). Special note for SCO systems: This represents the number of memory pages not currently in use. If it is large, then little RAM is being used. If it is near zero, then the system is having to use swap space.
cache_ctx	VM	Number of Ctx Cache flushes since the last reboot.
cache_seg	VM	Number of Segment Cache flushes since the last reboot.
cache_pag	VM	Number of Page Cache flushes since the last reboot.
cache_par	VM	Number of Partial Page Cache flushes since the last reboot.
cache_usr	VM	Number of User Cache flushes since the last reboot.
cache_reg	VM	Number of Region Cache flushes since the last reboot.

MBUF Tests

The following table shows the MBUF tests. These tests refer to the BSD UNIX buffer monitoring group. This group is not supported on System V based Operating Systems.

The mbufs and mbuf_drain tests are *not* supported by HP/UX systems.

Test Name	MIB Group	Description
mbufs	MBUF	Current number of mbufs obtained from the page pool.
mbuf_clusters	MBUF	Number of mbuf clusters obtained from the page pool.
mbuf_clfree	MBUF	Number of free clusters.
mbuf_drops	MBUF	Number of times failed to find space.
mbuf_space	MBUF	Number of interface pages obtained from the page pool.
mbuf_wait	MBUF	Number of times waited for space.
mbuf_drain	MBUF	Number of times drained protocols for space.
mbufs	MBUF	Current number of mbufs obtained from the page pool.

Ncache Tests

The following table shows the Ncache test. This test refers to the Name Cache, which is another memory subsystem. Its size is tunable on some versions of UNIX.

The size of this cache, like any cache, will affect its hit/miss ratio and, to a lesser degree, its purge frequency. Adjusting the size of your name cache may require recompiling the kernel and is not recommended. You can also use alarm thresholds and data logging to verify that any changes in size had the desired result.

Test Name	MIB Group	Description
ncache	Ncache	Percent of name cache misses. Specifically: misses/(hits+misses)

MIB II Tests

This section details the MIB II objects AgentMon can manage.

System Group

This group is used to store basic information about the workstation, who should be contacted, the system's location, and other administrative details.

For Solaris 2.x, only the systems group is implemented. The network statistics normally available on other O/Ss are not available on Solaris.

Interfaces Group

These groups list the various network interfaces and information relevant to their current state. They contain various statistics for the different networking protocols that are in use on the workstation.

The information in these groups can help pinpoint host-based network problems, aid in bandwidth utilization, and assist in resource planning. The groups include:

- IP—the Internet Protocol
- ICMP—the Internet Control Message Protocol
- TCP—the Transmission Control Protocol
- UDP—the Unreliable Datagram Protocol
- SNMP—the Simple Network Management Protocol
- Transmission group—shows current network connections

Typically, these groups contain:

- the number of packets (and/or bytes) sent and received
- the number of packets that were bad for various reasons
- the number of protocol errors

The following table shows which MIB II tests AgentMon can manage.

Test Name	MIB Group	Description
ip_total	MIB II	Total IP packets received.
ip_badsum	MIB II	Total IP packets having the wrong checksum.
ip_tooshort	MIB II	Number of IP packets that were "too short."
ip_toosmall	MIB II	Number of IP packets that were "too small."
ip_badhlen	MIB II	Number of IP headers having a bad length.
ip_badlen	MIB II	Number of IP packets of wrong length.
ip_fragments	MIB II	Number of fragmented IP packets.
ip_fragdropped	MIB II	Number of IP fragments discarded.
ip_fragtimeout	MIB II	Number of timeouts waiting for an IP fragment.

Test Name	MIB Group	Description
ip_forward	MIB II	Number of IP packets forwarded.
ip_cantforward	MIB II	Number of IP packets that could not be forwarded.
ip_redirectsend	MIB II	Number of redirected IP packets.
icmp_error	MIB II	Number of ICMP errors since the last reboot.
icmp_badcode	MIB II	Number of ICMP packets having an unknown icmp code.
icmp_tooshort	MIB II	Number of short ICMP packets received.
icmp_checksum	MIB II	Number of ICMP packets having a wrong checksum.
icmp_badlen	MIB II	Number of ICMP packets having a bad length.
icmp_reflect	MIB II	Number of ICMP packets received.
tcp_psent	MIB II	Number of TCP packets sent since the last reboot.
tcp_bsent	MIB II	Number of TCP bytes sent since the last reboot.
tcp_pgot	MIB II	Number of TCP packets received since the last reboot.
tcp_bgot	MIB II	Number of TCP bytes received since the last reboot.
tcp_dropped	MIB II	Number of TCP connections dropped since the last reboot.
udp_badhead	MIB II	Number of udp packets that arrived with bad headers.

Test Name	MIB Group	Description
udp_badsum	MIB II	Number of udp packets that arrived with a wrong checksum.
udp_badlen	MIB II	Number of udp packets that arrived with a bad length.
udp_overflow	MIB II	Number of udp socket overflows that have occurred.

Item Tests

The Group and API built-in tests have static names and functionality. ENlighten also provides support for certain types of additional tests you can define. These tests are File, Directory, and Processes tests.

Each of these three tests types is predefined with a purpose and may have further subcategories of tests. You can have as many of each of the three types of tests as you want; each test name will be the name of the particular item being tested.

The following list shows the types of these "item" tests and their associated naming conventions:

- File size

  The test will monitor the size of the specified file, for example, `/myfile size`. Use this to monitor the many files that are allowed to grow without bound. See the file `example1.sh` and the `command` capability for one possible way of automating corrective action for files that get too big.

- File accessed

  Use this test to monitor any files that have been read, for example, `/myfile accessed`.

- File modified

  Use this test to monitor any files that have been modified, for example, `/myfile modified`.

- File clamped

  Use this test to monitor logfiles for specific message types that you define, for example, `/myfile clamped`.

- Directories

  If a test name refers to a directory, the test will monitor the number of files in the directory. This can be used with an alarm set point to provide notification of a queue that is filling. Another possibility would be to use the data to show how queues fill and empty throughout the day.

  If the `age =` parameter is specified, then only files more than `age` minutes old will be counted.

- Processes instances

  A test name preceded by an exclamation point ("!") is assumed to refer to a process. Alarm thresholds can be used to issue notification when the process is started, stopped, or if the number of instances of the process changes.

- Processes size

  This test monitors the size of the named process in pages of the swappable process's image in main memory.

- Processes time

  This test monitors the total amount of CPU time used by the process.

File Tests

This section details the File subcategory tests. When the file size has breached an alarm threshold, a TRAP PDU will be sent to the NMS if the NMS has ENABLED the TRAP associated with that file. To create a test entry in the `testtab` file, enter the full pathname to the particular file you want AgentMon to monitor.

File Size

You can use the files group to monitor files. Since many UNIX files can grow without bound, AgentMon provides a method of automatically archiving, truncating, or otherwise averting a file size problem. For example, monitoring the size of `sulog` and other system files could aid in preserving the security of your system.

For example:

```
/opt/ENLIGHTEN/agent/history.pizza size| :\
   :on:testfreq=1:alarmfreq=60:\
   :command=/opt/ENLIGHTEN/example.sh:\
   :!log:high=1000000:units=bytes:\
   :/*end /opt/ENLIGHTEN/agent/history.pizza */:
```

If a test name is a full pathname and it does not refer to a directory, the test is assumed to refer to a file of the same name. The size of the file is then monitored and compared with alarm thresholds. If the named file does not exist, the test is ignored (until the file does exist).

File Accessed

You can also test if the file was accessed.

For example:

```
/etc/shadow accessed | :\
   :on:testfreq=1:alarmfreq=1:log:delta=1:\
   :+jump=1:
   :/*alarms if anyone reads the file */:
```

File Modified

Use this test to send an alarm if a file was modified.

For example:

```
!/usr/adm/sulog modified | :\
   :on:testfreq=1:+jump=1:\
   :/*end /usr/adm/sulog modified */:
```

File Clamped

This test evaluates any recent additions in ASCII logfiles to matches in the regular expressions regx1 - regx32. An alarm is generated if one or more of the regular expressions match one or more of the “new” log entries. A “new” entry is any entry added to the log since this test was last run.

For example:

```
/usr/adm/sulog clamped| :\
   :on:testfreq=5:alarmfreq=5:\
   :regx1 = (root)*(fail)*:high=1: \
   :/*end /usr/adm/sulog clamped */:
```

Directories Tests

The directories group is used to monitor the number of files in a directory. The directory may be a print queue, an email queue, or any other queue where files in transition are temporarily stored. An individual queue can be monitored by watching the number of files and/or by watching the number of "old" files.

Since queues are bottlenecks, it can become a problem if the number of files becomes large because this could then also lead to a shortage of disk space. An email queue that has many old files could mean that a remote site is no longer reachable.

When the number of files in a queue breaches an alarm threshold, a TRAP PDU will be sent to the NMS if the NMS has ENABLED the trap associated with that queue. The TRAP message will contain the name of the queue and the number of (old) files in it.

To create a test entry in the `testtab` file, use the full pathname of a directory as the test name. Then the number of files in that directory is tracked. By setting a high alarm threshold, you can be notified when queues are getting too big (backlogged).

For example:

```
/usr/spool/ps|:\
   :on:testfreq=5:alarmfreq=60:mailer=/bin/mail:\
   :age=5:notify=root:!log:high=20:units=old_files:\
   :/*end /usr/spool/ps */:
```

Since some queues have accounting files (and other non-queued files) in the queue directory, be sure to consider their number when setting alarm thresholds. If the named directory does not exist, the test is ignored. If the test description for this test contains the `age` parameter, then only files more than `age` minutes old will be counted.

Processes Tests

The processes group measures the number of currently running processes having the specified name. Given a list of processes and alarm thresholds, AgentMon can tell you when your daemons die, your programs terminate, when you don't have enough instances of a program running, how long a process has been executing, and the amount of memory a process has consumed.

In addition to the usual six types of alarms, two others are available: first instance started and last instance terminated. This means if you are monitoring a process called ABC, an alarm will occur when the first instance of ABC starts up, and when the last one terminates.

For each named process, a TRAP PDU will be sent if an alarm threshold for that process is breached and if the NMS has ENABLED the corresponding trap. The TRAP message will include the name of the process that went into the alarm condition and the number of processes with the same name that are currently running.

On BSD flavors of UNIX, you must specify the real process name, that is, the name returned by the `ps -c` command.

Processes Instances

To create a test entry in the `testtab` file, enter the process name (preceded by an exclamation point (!)) you want AgentMon to monitor.

For example:

```
!syslogd instances|:\
   :on:testfreq=5:mailer=/bin/mail:notify=root:!log:\
   :units=process(es):+jump=1:-jump=1:\
   :/*end!syslogd*/:
```

An alarm will be issued if any of the following are true:

- The named process starts and no other processes by that name were previously running (first instance start-up). This is enabled five minutes after AgentMon starts up to prohibit alarm at reboot time.
- The named process terminates and there are no other currently running processes having the same name.
- There is more than one instance of the named process running, and the number of instances changes by more than some user-specified alarm threshold (low limit, high limit, etc.).

Processes Size

This test monitors the size of the named process.

Processes Time

This test monitors the total amount of CPU time used.

API Tests

In addition to the many built-in tests, AgentMon also has six general-purpose test names you can use for tests you create. You can write the tests using any shell script or SQL, or you can use a compiled program. You can use any method you want.

All you have to do is have the test write results to an ordinary ASCII file. The file may contain many columns of data and many data records. AgentMon will watch the data, compare it to the alarm thresholds you have set, and notify you of any faults.

Designing an API Test

To use the API tests, follow these steps:

1) Write a script or program that will collect the data, for example:

```
ls -l /usr/spool/mail > mydata
```

2) Edit the `crontab` file to make your script run periodically.

3) Use the Events menu in the Combo GUI or edit the `testtab` file directly and define a NEW test. You can chose from the following names:
   - api1
   - api2
   - api3
   - api4
   - api5
   - api6

4) The "api" tests are set up just like any other test, except they have three additional fields. These fields are:

   - *filename*

     Specifies the full pathname of the file where your test writes data.

   - *data*

     Specifies which field or column the data is in. The value assigned is a digit prefaced by either an 'f' for field number or a 'c' for column number. In the absence of a qualifier, the default is 'f' for field.

     The field/column delineator is any blank space. Each character in a row is considered a column.

   - *label*

     Specifies the field or column containing a descriptive word or label.

At each `testfreq` interval, AgentMon will check the file pointed to by `filename` to see if its modification time has changed. If the file has changed, AgentMon will reread it. For each line it reads, AgentMon will find the value stored in field (column) `data` and compare it to any alarm thresholds you have defined for this test (api1, api2, ..., api6).

An Example API Test

The following is an example for creating an API Group test.

Scenario: You have several databases and want to know when one or more begin to run out of space. Have cron execute a script that will record the free space and database name to a file called `dbsizes`. Suppose the file looks like:

```
1289 Kbytes parts.db
9023 Kbytes customer.db
389 Kbytes phones.db
```

You could create the new test with the following additional parameters:

`Test name: api1`	(pick any unused API test name)
`:file=/dbsizes:`	(full pathname to file holding the data)
`:data=f1:`	(monitor the data in field one (column one if `c1`))
`:label=f3:`	(the label in field #3 is the database name)
`:low=400:`	(set a low-level alarm at 400 Kbytes)

as shown in the following example:

```
api1 |:\
   :on:file=/dbsizes:data=f1:label=f3:low=400:\
   :testfreq=1:alarmfreq=60:\
   :mailer=/bin/mail:notify=root:!log:\
   :/*end api1*/:
```

Generating Reports

You can use the Status Map to view all logs and events and the Query Events function to search for relevant event logs. See Chapter 11, “Navigate,” to use both of these features.

You can also use SQL to generate Events reports from the EMD. Refer to your *SQL User's Manual* for more details.