Solstice Site/SunNet/Domain Manager Reference Manual

Sun Microsystems, Inc.
2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.

Preface

OVERVIEW

A man page is provided for both the naive user, and sophisticated user who is familiar with the Sun Network system and is in need of online information. A man page is intended to answer concisely the question ‘‘What does it do?’’ The man pages in general comprise a reference manual. They are not intended to be a tutorial.

The following contains a brief description of each section in the man pages and the information it references:

- Section 1 describes, in alphabetical order, commands available with the Network User commands.
- Section 3N describes the Network Functions.
- Section 5 describes the Headers, Tables, and Macros available with Site/SunNet/Domain Manager.
- Section 8 describes the Maintenance Procedures.

Below is a generic format for man pages. The man pages of each manual section generally follow this order, but include only needed headings. For example, if there are no bugs to report, there is no BUGS section. See the intro pages for more information and detail about each section, and **man**(1) for more information about man pages in general.

NAME

This section gives the names of the commands or functions documented, followed by a brief description of what they do.

SYNOPSIS

This section shows the syntax of commands or functions. When a command or file does not exist in the standard path, its full pathname is shown. Literal characters (commands and options) are in **bold** font and variables (arguments, parameters and substitution characters) are in *italic* font. Options and arguments are alphabetized, with single letter arguments first, and options with arguments next, unless a different argument order is required.

The following special characters are used in this section:

[ ] The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument *must* be specified.

... Ellipses. Several values may be provided for the previous argument, or the previous argument can be specified multiple times, for example, '*filename* ...'.

| Separator. Only one of the arguments separated by this character can be specified at time.

{ } Braces. The options and/or arguments enclosed within braces are interdependent, such that everything enclosed must be treated as a unit.

AVAILABILITY

This section briefly states any limitations on the availabilty of the command. These limitations could be hardware or software specific.

A specification of a class of hardware platform, such as **x86** or **SPARC**, denotes that the command or interface is applicable for the hardware platform specified.

In Section 1 and Section 1M, **AVAILABILITY** indicates which package contains the command being described on the manual page. In order to use the command, the specified package must have been installed with the operating system. If the package was not installed, see **pkgadd**(1) for information on how to upgrade.

DESCRIPTION

This section defines the functionality and behavior of the service. Thus it describes concisely what the command does. It does not discuss OPTIONS or cite EXAMPLES. Interactive commands, subcommands, requests, macros, functions and such, are described under USAGE.

OPTIONS

This lists the command options with a concise summary of what each option does. The options are listed literally and in the order they appear in the SYNOPSIS section. Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.

DEFAULTS

This lists any default settings or values a function might have and describe them in details.

RETURN VALUES

If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or –1, these values are listed in tagged paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared as **void** do not return values, so they are not discussed in RETURN VALUES.

ERRORS

On failure, most functions place an error code in the global variable **errno** indicating why they failed. This section lists alphabetically all error codes a function can generate and describes the conditions that cause each error. When more than one condition can cause the same error, each condition is described in a separate paragraph under the error code.

USAGE

This section is provided as a *guidance* on use. This section lists special rules, features and commands that require in-depth explanations. The subsections listed below are used to explain built-in functionality:

Commands

Modifiers
Variables
Expressions
Input Grammar

EXAMPLES

This section provides examples of usage or of how to use a command or function. Wherever possible a complete example including command line entry and machine response is shown. Whenever an example is given, the prompt is shown as

example%

or if the user must be super-user,

example#

Examples are followed by explanations, variable substitution rules, or returned values. Most examples illustrate concepts from the SYNOPSIS, DESCRIPTION, OPTIONS and USAGE sections.

ENVIRONMENT

This section lists any environment variables that the command or function affects, followed by a brief description of the effect.

FILES

This section lists all filenames referred to by the man page, files of interest, and files created or required by commands. Each is followed by a descriptive summary or explanation.

SEE ALSO

This section lists references to other man pages, in-house documentation and outside publications.

DIAGNOSTICS

This section lists diagnostic messages with a brief explanation of the condition causing the error. Messages appear in **bold** font with the exception of variables, which are in *italic* font.

WARNINGS

This section lists warnings about special conditions which could seriously affect your working conditions — this is not a list of diagnostics.

NOTES

This section lists additional information that does not belong anywhere else on the page. It takes the form of an *aside* to the user, covering points of special interest. Critical information is never covered here.

BUGS

This section describes known bugs and wherever possible suggests workarounds.

NAME

Intro – introduction to the Site / SunNet / Domain Manager Reference Manual

OVERVIEW

Site / SunNet / Domain Manager is a platform for the management of distributed work group networks. The product features mechanisms for extending the platform, providing the ability to continue effective management as requirements change. It also features a collection of tools immediately useful for management.

The *Site/SunNet/Domain Manager Reference Manual* describes the commands and utility programs included with Site / SunNet / Domain Manager Release 2.3. This manual covers all man pages under <*installation_path*>**/snm/man**.

To use the Site / SunNet / Domain Manager man pages, the MANPATH environment variable should be set to <*installation_path*>**/snm/man** where the <*installation_path*> is **/usr** if the default location is used for a Solaris 1.x installation and **/opt/SUNWconn** if the default location is used for a Solaris 2.x installation.

Name	**Command Description**
acl.pty(5)	access control configuration for SNMPv2 entities
agt.pty(5)	party configuration for SNMPv2 agents
build_mdb(1)	build mdb from NIS
build_oid(1)	build Object Identifier Database
build_tt(1)	build textual convention types database
context.pty(5)	context info configuration for SNMPv2 entities
discover.conf(5)	configuration file for SunNet Manager Discover Tool
getagents(8)	get Site / SunNet / Domain Manager agent environment
linkmap(5)	format of SunNet Manager linkmap file
mgr.cnf(5)	SNMPv2 manager configuration
mgr.pty(5)	party configuration for SNMPv2 managers
mib2schema(1)	translates MIB files to SunNet Manager schema files.
na.activity(8)	management activity daemon
na.cpustat(8)	per cpu statistics (Solaris 2.x)
na.diskinfo(8)	na.diskinfo — report disk space information
na.etherif(8)	ethernet interface statistics
na.etherif2(8)	ethernet interface statistics (Solaris 2.x)
na.event(8)	event dispatcher
na.hostif(8)	interface statistics
na.hostmem(8)	network memory buffer pool usage and statistics
na.hostmem2(8)	network memory buffer pool usage and statistics (for

	Solaris 2.x)
na.hostperf(8)	host statistics
na.iostat(8)	Input / Output statistics
na.iostat2(8)	Input / Output statistics
na.ippath(8)	na.ippath — print the path that packets take to a network host
na.iproutes(8)	routing table statistics
na.layers(8)	network layers statistics 3for SunOS 4.1.x1
na.layers2(8)	network layers statistics for 3Solaris 2.x1
na.logger(8)	management logger
na.lpstat(8)	na.lpstat — print the line printer status and information
na.ping(8)	na.ping — machine reachability information
na.rpcnfs(8)	RPC and NFS statistics
na.snmp-trap(8)	SNMP trap daemon
na.snmp(8)	Simple Network Management Protocol proxy agent
na.snmp.hostfile(5)	Site / SunNet / Domain Manager SNMP target configuration file
na.snmp.trapfile(5)	SunNet Manager SNMP trap file
na.snmp.traplog(5)	SunNet Manager SNMP trap logfile
na.snmpv2(8)	Simple Network Management Protocol - Version 2 proxy agent
na.sync(8)	synchronous interface statistics
na.traffic(8)	network traffic statistics
na.x25(8)	virtual circuit statistics
netmgt_alloc_manager_id(3N)	manipulate the manager application ID
netmgt_build_report(3N)	add a statistic to a report
netmgt_dbg(3N)	print debugging output
netmgt_fetch_argument(3N)	fetch a request argument
netmgt_fetch_data(3N)	get data statistics
netmgt_fetch_error(3N)	fetch a network management error buffer
netmgt_fetch_event(3N)	fetch event report statistics
netmgt_fetch_msginfo(3N)	fetch received message information
netmgt_fetch_setval(3N)	fetch set request argument
netmgt_free_manager_id(3N)	manipulate the manager application ID
netmgt_get_manager_id(3N)	manipulate the manager application ID

netmgt_init_rpc_agent(3N)	initialize an RPC-based agent
netmgt_kill_request(3N)	terminate a request
netmgt_kill_request2(3N)	terminate requests started by one manager
netmgt_mark_end_of_row(3N)	set table end-of-row
netmgt_oid2string(3N)	convert an object identifier to a printable string
netmgt_register_callback(3N)	register a transient RPC callback function
netmgt_register_rendez(3N)	register with the event dispatcher
netmgt_request_agent_id(3N)	request identification from an agent
netmgt_request_data(3N)	request data reports from an agent
netmgt_request_deferred(3N)	get any available deferred data reports
netmgt_request_events(3N)	request event reports from an agent
netmgt_request_set(3N)	send a set request to an agent
netmgt_request_set2(3N)	send a set request to an agent
netmgt_restore_argument(3N)	restore optional argument
netmgt_restore_request(3N)	restore current request state
netmgt_restore_threshold(3N)	restore event threshold
netmgt_save_argument(3N)	save optional argument
netmgt_save_attribute(3N)	save a data attribute
netmgt_save_request(3N)	save current request state
netmgt_save_threshold(3N)	save event threshold
netmgt_send_error(3N)	send an error report
netmgt_send_report(3N)	send a data or event report
netmgt_set_argument(3N)	append an argument to the request
netmgt_set_attribute(3N)	request an attribute in a data report
netmgt_set_debug(3N)	set the debugging level
netmgt_set_instance(3N)	initialize a request
netmgt_set_manager_id(3N)	manipulate the manager application ID
netmgt_set_threshold(3N)	set the event report threshold for an attribute
netmgt_set_value(3N)	append a set argument to the request
netmgt_shutdown_agent(3N)	unregister and terminate agent
netmgt_sperror(3N)	get a description of an error
netmgt_start_agent(3N)	start the agent
netmgt_start_trap(3N)	set context for sending traps
netmgt_unregister_callback(3N)	unregister a transient RPC callback function
netmgt_unregister_rendez(3N)	unregister from the event dispatcher

snm(1)	Site / SunNet / Domain Manager Console
snm.conf(5)	Site / SunNet / Domain Manager configuration file for agents and daemons
snm.logfile(5)	format of Site / SunNet / Domain Manager log files
snm_br(1)	Site / SunNet / Domain Manager results browser
snm_cmd(1)	Site / SunNet / Domain Manager command-line manager application
snm_cmdtool(1)	execute a cmdtool that exec's a command line
snm_cvtlog(1)	convert log to old format
snm_discover(8)	Site / SunNet / Domain Manager network element discoverer / monitor
snm_error(3N)	error code for snmdb routines
snm_exec(1)	execute a command and pause after the command exits
snm_gr(1)	Site / SunNet / Domain Manager results grapher
snm_ipx_discover(8)	IPX / SPX nodes discoverer for Site / SunNet / Domain Manager
snm_kill(1)	stop one or more agent requests
snm_parser(1)	Site / SunNet / Domain Manager schema parser
snm_schema(5)	format of Site / SunNet / Domain Manager schema
snm_set(1)	Site / SunNet / Domain Manager Set Tool
snm_version(8)	print version information about Site / SunNet / Domain Manager binaries
snmdb_add(3N)	adds a new element into the database.
snmdb_add_agent(3N)	adds an agent record for the element
snmdb_add_alias(3N)	adds an alias record for the element
snmdb_add_connection(3N)	adds a connection record for the element
snmdb_add_to_view(3N)	adds the element into a particular view
snmdb_console_load(3N)	causes the Site / SunNet / Domain Manager Console to load a specified ASCII management database (MDB) file
snmdb_console_reload(3N)	causes the Site / SunNet / Domain Manager Console to reinitialize the runtime database and load the specific server
snmdb_console_save_components(3N)	causes the Site / SunNet / Domain Manager Console to save all instances in its runtime database to a specified ASCII management database (MDB) file

snmdb_delete(3N)	deletes an existing element from the database
snmdb_delete_agent(3N)	deletes an agent record for the element
snmdb_delete_alias(3N)	deletes an alias record for the element
snmdb_delete_color(3N)	deletes the color value of an element
snmdb_delete_connection(3N)	deletes a connection record for the element
snmdb_delete_from_view(3N)	deletes the element from a particular view
snmdb_enumerate_agents(3N)	enumerates the agents that apply to the element
snmdb_enumerate_aliases(3N)	enumerates all the alias records of the element
snmdb_enumerate_connections(3N)	enumerates all the connections of the element
snmdb_enumerate_elements(3N)	enumerates all the elements in a view
snmdb_enumerate_views(3N)	enumerates the views in which the element exists.
snmdb_free_enumeration_handle(3N)	frees enumeration storage of elements
snmdb_free_list(3N)	frees enumeration storage of agent list, view list, and connection list
snmdb_get_agent(3N)	gets an agent record for the element
snmdb_get_color(3N)	reads the color value of an element
snmdb_get_element_glyph(3N)	returns the glyph file name of an element type
snmdb_get_element_type(3N)	returns the type of an element
snmdb_get_next_element(3N)	get the next element in the enumerated list returned from 3snmdb_enumerate_elements1
snmdb_get_property(3N)	gets the value and data type of a particular property of an element that has been read into the internal buffer
snmdb_get_view(3N)	reads the coordinates of the element in a particular view
snmdb_init_buffer(3N)	initialize a buffer with the element's name and type
snmdb_lock(3N)	locks the database
snmdb_open(3N)	open the database
snmdb_read(3N)	reads an existing element from the database into the specified internal buffer. This is necessary for retrieving or updating any properties of a particular element
snmdb_save_element(3N)	save an element record into the file specified in ASCII format
snmdb_set_color(3N)	sets the color of an element
snmdb_set_property(3N)	sets the value of a particular property of an element

snmdb_unlock(3N)	unlocks the database
snmdb_update(3N)	update an existing element in the database
snmp-mibII.schema(5)	SNMP MIB-II schema.
snmp.schema(5)	SNMP MIB-I schema.
snmpd(8)	snmpd - Sun SNMP Agent
snmpv2d(8)	snmpv2d - Sun SNMPv2 Agent
snmpv2d.conf(5)	the SNMP agent configuration
snmp_set(8)	perform an SNMP set request on a device
v2install(1)	generate SNMPv2 configuration files for agents and managers
v2mib2schema(1)	Translates SNMPv2 MIB files to SunNet Manager schema files.
view.pty(5)	MIB view access configuration for SNMPv2 entities

NAME

acl.pty – access control configuration for SNMPv2 entities

SYNOPSIS

Target Subject Resources Privileges StorageType

DESCRIPTION

The configuration file **acl.pty** is one of several configuration files required by the SNMPv2 entities. The default location of **acl.pty** is **/etc/opt/snm/agent** or **/etc/opt/snm/manager** for Solaris 2.x and **/etc/snm/agent** or **/etc/snm/manager** for Solaris 1.x, but can be specified by the environment variables **SR_AGT_CONF_DIR** and **SR_MGR_CONF_DIR.**

The **acl.pty** file defines information for the Access Control List (ACL) table, which in turn contains information about the access privileges for the target-party / subject-party pair. There must be one entry for every intended target that defines the access privileges for the target and subject in their SNMPv2 context.

In other words, for every agent and manager that will be communicating with each other, there must be an entry in the **acl.pty** file that, along with a corresponding entry in the **context.pty** file, defines the access privileges for the two entities. See **context.pty**(5) for more information on the context.

Each entry in the file consists of 1 line:

Target Subject Resources Privileges StorageType

where

Target represents the target party for this ACL entry. This index must match the *partyIndex* of an entry in the party table, *agt.pty*. This party's performance of management operations is constrained by the set of access privileges for this entry (see *Privileges* below).

Note: The *partyAuthProtocol* value for this party must be the same as the one for the *Subject* party in the party table. In other words, the target and the subject parties must have equivalent *PartyDiscriminator* types in the **agt.pty** records. See the *PartyDiscriminator* description in **agt.pty**(5) for an explanation of the types. The party table entries are discussed further **agt.pty**(5) and in the Party MIB RFC, RFC1447.

This field is an integer.

Subject similar to *Target,* represents the subject party for this ACL entry. This index must match the *partyIndex* of an entry in the party table, **agt.pty.** This party's performance of management operations is constrained by the set of access privileges for this entry (see *Privileges* below).

Note: The *partyAuthProtocol* value for this party must be the same as the one for the *Target* party in the party table. In other words, the target and the subject parties must have equivalent *PartyDiscriminator* types in the **agt.pty** records. See the *PartyDiscriminator* description in **agt.pty**(5) for an explanation of the types. The party table entries are discussed further **agt.pty**(5) and in the Party MIB RFC, RFC1447.

This field is an integer.

Resources

represents the context for this ACL entry. This index must match the *contextIndex* of an entry in the context table, **context.pty.** See **context.pty**(5) for more information on context entries.

Privileges

represents the value that governs what management operations a particular target party can perform on behalf of the subject (source) party. This field may have a value ranging from 0 to 255. This value is a sum of values, each of which represents a management operation.

Possible values are:

1	Get
2	GetNext
4	Response
8	Set
16	snmpv1trap
32	GetBulk
64	Inform
128	SNMPv2trap

The default value for this field is *35:* 1 + 2 + 32 or *Get, GetNext,* and *Getbulk.*

partyStorageType

indicates the storage type for this row in the party table. Possible values are:

other
volatile
nonVolatile
permanent

According to RFC1447,

- volatile is lost upon reboot, e. g., in RAM,
- nonVolatile is backed up by stable storage, e. g., in NVRAM,
- permanent cannot be changed or deleted, e. g., in ROM,

and "other" is provided in the unlikely event that someone will find a need for a storage type not covered by the other three.

This field is a case-sensitive string corresponding to one of the above values.

EXAMPLE

An example **acl.pty** entry might be

1 2 1 43 nonVolatile

which defines this entry as follows:

Target The party entry with *partyIndex* of *1* is the destination (or target).

Subject The party entry with *partyIndex* of *2* is the source (or subject).

Resources
The context entry that has a *contextIndex* of *1* is the SNMPv2 context.

Privileges
The entity would expect to be able to perform *get*, *get-next*, *get-bulk,* and *set* requests with this ACL entry.

StorageType
Store this entry in non-volatile memory, e. g. NVRAM.

FILES

Additional SNMPv2 Configuration Files

When the agent is compiled with either SNMPv2 or both SNMPv1 and SNMPv2 defined, the configuration files **acl.pty, agt.pty, context.pty, snmpv2d.conf** and **view.pty** are required for the agent side, and **acl.pty, context.pty, mgr.cnf, mgr.pty,** and **view.pty** are required for the manager side.

acl.pty Access control privileges for the SNMPv2 parties.

agt.pty Initial party table information for the agent.

context.pty
Context information for the SNMPv2 parties.

mgr.cnf
Configuration information for the managers.

mgr.pty
Initial party table information for the managers.

snmpv2d.conf
Configuration information for the SNMPv1 entities.

view.pty
MIB view information for the SNMPv2 parties.

For Solaris 2.x, the files are located under:

/etc/snm/{agent,manager}/acl.pty
/etc/snm/agent/agt.pty
/etc/snm/{agent,manager}/context.pty
/etc/snm/agent/snmpv2d.conf
/etc/snm/{agent,manager}/view.pty

For Solaris 1.x, the files are located under:

/etc/snm/{agent,manager}/acl.pty
/etc/snm/agent/agt.pty
/etc/snm/{agent,manager}/context.pty
/etc/snm/agent/snmpv2d.conf
/etc/snm/{agent,manager}/view.pty

SEE ALSO

v2install(1), **agt.pty**(5), **context.pty**(5), **mgr.cnf**(5), **mgr.pty**(5), **snmpv2d.conf**(5), **view.pty**(5), SNMP RFCs (RFC1155 RFC1157 RFC1212 RFC1213 RFC1215, RFCs 1441-1452)

NAME

agt.pty – party configuration for SNMPv2 agents

SYNOPSIS

PartyName PartyDiscriminator
TDomain TAddress Port Lifetime MaxMsgSize
partyIndex partyStorageType partyLocal partyAuthClock
AuthPublicSecret | –
AuthPrivateSecret | –
PrivPublicSecret | –
PrivPrivateSecret | –

DESCRIPTION

The configuration file **agt.pty** is one of several configuration files required by the SNMPv2 entities. The default location of **agt.pty** is **/etc/opt/snm/agent** for Solaris 2.x and **/etc/snm/agent** for Solaris 1.x, but can be specified by the environment variable **SR_AGT_CONF_DIR.**

The **agt.pty** is similar to the **mgr.pty**(5) file used by managers; it defines the party table entries for the parties associated with the agent.

Each entry in the file consists of 7 lines:

PartyName PartyDiscriminator
TDomain TAddress Port Lifetime MaxMsgSize
partyIndex partyStorageType partyLocal partyAuthClock
AuthPublicSecret | –
AuthPrivateSecret | –
PrivPublicSecret | –
PrivPrivateSecret | –

where

PartyName
is a unique name for the party. This field is required and must map to an OID in the MIB.

PartyDiscriminator
defines the authentication protocol to be used by this party:

1 or 2 defines this party as noauth / nopriv
3 or 4 defines this party as auth / nopriv
5 or 6 defines this party as auth / priv

This field is an integer and must be present for SNMPv2 parties. If the corresponding *TDomain* is *rfc1157Domain*, this field is ignored. However, a value must be present, and an entry of 1 would be reasonable.

TDomain
defines the transport domain for the party. There are two valid entries for the *TDomain:*

rfc1157Domain — indicates this is a SNMPv1 community entry in the party table.

snmpUDPDomain — indicates this is a SNMPv2 party entry.

Note: A party can be *rfc1157Domain* only if the agent is a bilingual agent, i. e., understands both SNMPv1 and SNMPv2. This file is not consulted when the agent is compiled to be SNMPv1 only.

TAddress

depends on *TDomain* and *partyLocal* for its definition:

When *TDomain* is *rfc1157Domain,* this field, in conjunction with the *Port* field, defines the trap address.

When *TDomain* is *snmpUDPDomain* and *partyLocal* is *false,* this field, in conjunction with the *Port* field, defines either the trap address or the proxy address.

When *TDomain* is *snmpUDPDomain* and *partyLocal* is *true,* this field, and the *Port* field, are ignored.

This field must be present and is an IP address in dotted decimal form; i.e., 12.169.4.9.

Port depends on *TDomain* and *partyLocal* for its definition:

When *TDomain* is *rfc1157Domain,* this field, in conjunction with the *TAddress* field, defines the trap address.

When *TDomain* is *snmpUDPDomain* and *partyLocal* is *false,* this field, in conjunction with the *TAddress* field, defines either the trap address or the proxy address.

When *TDomain* is *snmpUDPDomain* and *partyLocal* is *true,* this field, and the *TAddress* field, are ignored.

This field must be present and is an integer.

partyIndex

used by the ACL table to match a party entry with its access privileges.

Note: This must be a unique value for each party table entry.

This field is an integer in the range of 1 to 65535, inclusive.

partyStorageType

indicates the storage type for this row in the party table. Possible values are:

other
volatile
nonVolatile
permanent

According to RFC1447,

- volatile is lost upon reboot, e. g., in RAM,
- nonVolatile is backed up by stable storage, e. g., in NVRAM,
- permanent cannot be changed or deleted, e. g., in ROM,

and "other" is provided in the unlikely event that someone will find a need for a storage type not covered by the other three.

This field is a case-sensitive string corresponding to one of the above values.

partyLocal

When *TDomain* is *rfc1157Domain,* this field is ignored.

When *TDomain* is *snmpUDPDomain,* indicates whether this party represents the "local" end of a transmission.

This field has two possible values:

true: it is "local,"

or

false: it is not "local,"

Note: When the party is "not local," and the *TDomain* is *snmpUDPDomain* (i. e., SNMPv2), the *TAddress* and *Port* either may be used by a proxy agent to determine which address / port pair should receive proxy requests or to indicate which address / port pair should receive traps.

This field is required (whether representing a SNMPv1 or SNMPv2 party) and is a case-sensitive string corresponding to one of the above values.

Note: When the *TDomain* indicates a SNMPv2 party, the term "local" does not mean local in the sense of location or address, but rather in the sense of origin. This means that on outgoing packets, the source party must be a "local" party–one representing the local entity as the source of the packet; on incoming packets, the source party must be a "remote" party–one representing a valid source party for sending packets to the local entity. In this sense, local means representing the local entity. "Non-local" or "remote" parties are ones that are logically remote to the entity; i. e., the source party of an incoming packet–the party that sent the packet–and the destination party of an outgoing packet–the party that will be receiving the packet at the other end of the transmission.

In other words, packets are received from and sent to "non-local" (or remote) parties, and packets are sent from and received by local parties.

Note: entries that are "shared" with another entity will have *partyLocal* fields that are opposite. For instance, an entry for a local party (*partyLocal = true*) in the agent's party file would be configured as a non-local (*partyLocal = false*) in the manager's party file.

partyAuthClock

defines the current notion of time for the entity.

When *TDomain* is *rfc1157Domain,* this field is unused.

On startup a pre-defined adjustment, *TIME_WARP* in *secure.h,* is added to this value–unless the resulting value would exceed the maximum possible time. This is done to help prevent replay attacks across reboots.

This field must be present and is an integer in the range 0 to 4294967295, inclusive. A perfectly valid initial value is 0; the security software should adjust and "synch" clocks once everything is up and running.

AuthPublicSecret
is unused by the agent at this time. Will represent the "secret" as a series of hexadecimal numbers, each digit representing the corresponding ASCII value for the character in the string. A value of – represents the null string. The string can range from 0 to 16 bytes.

Note: This field should be null (–) for now.

AuthPrivateSecret
represents the "secrets" string for the authentication protocol. The secret is stored as a series of hexadecimal numbers; each digit is the ASCII value for the corresponding character in the string. A value of – represents the null string. The string can range from 0 to 16 bytes, depending on the *TDomain:*

If the party is an *auth/nopriv* or *auth/priv* party, this field must have a length of 16 bytes (i.e., 16 hexadecimal numbers in the configuration file). Both digits of each number must be represented. In other words, *1* would be entered as *01*.

If the corresponding *TDomain* field is equal to *rfc1157Domain,* contains a community string name as an encoded string of characters and can be any length. For example, if using the community name "public" the entry would be "70 75 62 6c 69 63". The command

echo *string* | od –x

is very useful for acquiring the correct encoding.

It would be practical to mirror entries in the **snmpv2d.conf**(5) file, but not necessary.

Note: Valid only when the *PartyDiscriminator* indicates authentication (i. e., *PartyDiscriminator* > 2), or *TDomain* indicates SNMPv1 community.

PrivPublicSecret
is unused by the entity at this time. Will store the "secret" as a series of hexadecimal numbers, each digit representing the ASCII value for the character in the string. A value of – represents the null string. The string can range from 0 to 16 bytes.

Note: This field should be null (–) for now.

PrivPrivateSecret
represents the "secrets" string for *priv* privacy protocol. The secret is stored as a series of hexadecimal numbers; each digit is the ASCII value for the corresponding character in the string. A value of – represents the null string.

If the party is an *auth/priv party*, this field must have a length of 16 bytes (i.e., 16 hexadecimal numbers in the configuration file). Both digits of each number must

be represented. In other words, 1 would be entered as 01.

Note: Valid only when the *PartyDiscriminator* indicates privacy, i. e., *PartyDiscriminator* > 4).

EXAMPLES

The party table entry:

```
initialPartyId.192.147.142.16.2 3
snmpUDPDomain 192.147.142.16 162 300 1458
2 nonVolatile false 0
–
74 68 69 73 74 68 69 73 74 68 69 73 74 68 69 34
–
–
```

defines a party as follows:

The party name is *initialPartyId.192.147.142.16.2*, and this is an *auth/nopriv* party.

The *TDomain* is *snmpUDPDomain*, indicating this is a SNMPv2 party.

Since *partyLocal* is *false* and it is a SNMPv2 party, the *TAddress* and *Port* fields indicate either a proxy address or where traps should be sent.

The *Lifetime* of the message is set to *300* indicating that the message will be valid as long as it is received by the target at a time *greater than* the target's idea of "now" minus 300 seconds.

The message can be no longer than *1458* bytes.

The unique party index is *2*.

Store this party in non-volatile storage (e. g., NVRAM).

This party is *not* local to the SNMPv2 entity, so it is a valid outgoing destination or a valid incoming source.

The initial clock will be *0* plus *TIME_WARP*.

The *AuthPublicSecret* is unused, so there is a null string for its value.

The *AuthPrivateSecret* is initialized to

74 68 69 73 74 68 69 73 74 68 69 73 74 68 69 34

which decodes to *thisthisthisthi4*.

The *PrivPublicSecret* is unused, so there is a null string for its value.

Since the *PartyDiscriminator* indicates *nopriv*, the *PrivPrivateSecret* is unused, and there is a null string for its value.

Example of a Community Entry

The party table entry:

```
# public
initialPartyId.192.147.142.16.31 1
rfc1157Domain 192.147.142.16 162 300 1458
31 nonVolatile true 0
–
```

70 75 62 6c 69 63
–
–

defines a party as follows:

The party name is *initialPartyId.192.147.142.16.31*. The *PartyDiscriminator* is ignored since this is a community, but here must be value in the field; a reasonable entry would be 1 (noauth/nopriv in the SNMPv2 world). This *is* essentially a *noauth/nopriv* since it is a SNMPv1 community, but the actual value of the *partyAuthProtocol* will be *rfc1157noAuth* and *partyPrivProtocol* will be *noPriv*.

The *TDomain* indicates this is a SNMPv1 community entry in the party table.

The *partyLocal* field is unused since this is a community, but a value must be present.

The *TAddress* and *Port* fields indicate that traps should be sent to port *162* at IP address *192.147.142.16*.

The *Lifetime* field is unused since this is a community, but a value must be present.

The message can be no longer than *1458* bytes.

The unique party index is *31*.

Store this party in non-volatile storage, e. g., NVRAM.

The *partyLocal* and *partyAuthClock* fields are unused since this is a community, but values must be present.

The *AuthPublicSecret*, *PrivPublicSecret*, and *PrivPrivateSecret* fields are also unused since this is a community, but the null strings can be used to fill their spots.

The *AuthPrivateSecret* contains the community name:

70 75 62 6c 69 63

which decodes to *public*. This entry may be mirrored in the **snmpv2d.conf**(5) file.

The command **od (1)** is helpful when encoding the community name.

FILES

Additional SNMPv2 Configuration Files

When the entity is compiled with either SNMPv2 or both SNMPv1 and SNMPv2 defined (bilingual), the configuration files **acl.pty, context.pty, snmpv2d.conf,** and **view.pty** are required.

acl.pty Access control privileges for the SNMPv2 parties.

context.pty
Context information for the SNMPv2 parties.

snmpv2d.conf
System initialization and SNMPv1 community/trap community configuration.

view.pty
MIB view information for the SNMPv2 parties.

For Solaris 2.x, the files are located under:

/etc/opt/snm/agent/acl.pty
/etc/opt/snm/agent/agt.pty
/etc/opt/snm/agent/context.pty
/etc/opt/snm/agent/view.pty

For Solaris 1.x, the files are located under:

/etc/snm/agent/acl.pty
/etc/snm/agent/agt.pty
/etc/snm/agent/context.pty
/etc/snm/agent/view.pty

SEE ALSO

v2install(1), **acl.pty**(5), **context.pty**(5), **mgr.cnf**(5), **mgr.pty**(5), **snmpv2d.conf**(5), **view.pty**(5), SNMP RFCs (RFC1155 RFC1157 RFC1212 RFC1213 RFC1215, RFCs 1441-1452)

NAME

build_mdb – build mdb from NIS

SYNOPSIS

build_mdb *mdb*

DESCRIPTION

build_mdb is a script that builds an instance file from the data in the NIS hosts and networks maps. With some modification, it can use standard **/etc/hosts** or DNS (that is, **named**) files.

The script first creates several **awk** scripts and puts them in **/usr/tmp. It then uses ypcat** to create two files, **/tmp/hosts.nnnn /tmp/nets.nnnn,** where *nnnn* is the current process ID. If you are not using NIS, you'll need to modify the script to copy your **/etc/hosts** and **/etc/networks** files (or convert your **named** database) into the two files in **/tmp.** The hosts file in **/tmp** should have the format:

<addr><white space><hostnames>#<space><comment>

while the networks file in **/tmp** should have the format:

<name><white space><addr><white space><aliases>#<space><comment>

where

<name>
is the name of the host or network

<addr>
is the addr of the host or network in A.B.C.D form

<comment>
is an optional comment field.

Once the files are created, extra whitespace is removed and the script looks for duplicate network or host names. If any are found, the script terminates without further processing. You should remove the duplicate names and run the script again.

All network names are prepended with ''Net.''. If a network does not have a name, the script generates an entry of the form Net.<networknumber>.

To limit the number of elements in a view, networks are broken into views at each byte in their address. Records are placed in the view for the next level of network they are in. For example, a host with an address 1.2.3.4 would be placed in view ''Net.1.2.3''. A network 1.2.3 would be placed in view ''Net.1.2'', etc.

OPTIONS

mdb The file where the instance information should be written. The file will contain a record for each host in the NIS database. In addition, it will contain a view instance for each network.

DEFAULTS

The script has some defaults you may wish to change by editing the script (see the top of the file):

default_proxy
The machine to use as the proxy when the instance records are built. As delivered, this is the name of the machine where **build_mdb** is run.

default_color
: The default glyph color. As delivered, glyphs are set to *white*.

default_elem_type
: The default type of element. As delivered, elements are set to the type *component.sun4*.

default_view
: The default type of view when making views (from subnets). As delivered, views are of type *view.subnet*.

NOTES

The script doesn't understand subnetting very well. If you have a netmask set to something that doesn't end on a byte boundary, you probably won't like the way the views are constructed.

The script doesn't try to find out what type of component something is. All components are set to one type. The same is true for networks.

It doesn't figure out which agents are installed. While is is technically possible, it would take too much time to really be practical in large networks.

It doesn't connect anything with buses or connections. It just places elements in a view corresponding to their network.

Duplicate component names should probably be treated as gateways rather than rejected.

NAME

build_oid – build Object Identifier Database

SYNOPSIS

build_oid *[dir1 dir2 ... dirN]*

DESCRIPTION

build_oid builds the Object Identifier Database. The console uses the database to interpret object identifiers.

One of the data types supported by SunNet Manager is an object identifier. Since object identifiers are numeric (i.e., 1.2.3.4.5), a database is provided so the console can display the object identifier using a more meaningful string. The Object Identifier Database provides the information required for this mapping.

OPTIONS

build_oid has no options but may take directory names as arguments. Instead, it uses the environment variable **SNMHOME** to locate the SunNet Manager supplied schema files. If **SNMHOME** is undefined, **/opt/SUNWconn/snm** for Solaris 2.x or **/usr/snm** for Solaris 1.x will be used as the default product directory.

The output file *oid.dbase* is written to the directory specified by the **SNMDBDIR** environment variable. If **SNMDBDIR** is undefined, **/var/opt/snm** for Solaris 2.x or **/var/adm/snm** for Solaris 1.x is used as the default database directory

USAGE

build_oid should be run any time the object indentifier database needs to be built. Examples include at installation of the SUNWsnmct package (when the installation script automatically runs **build_oid)** and whenever new object identifiers need to be added to the database.

Input to the database

Input to the database consists of multiple ASCII files. The files consist of lines containing a descriptive name, followed by its object identifier. If the descriptive name contains special characters, it must be enclosed in double quotes. Comment lines may be included by beginning them with a pound sign (#).

FILES

oid.dbase

This file contains the object identifier database. It is written into the directory specified by the **SNMDBDIR** environment variable. This file is written to **/var/opt/snm** for Solaris 2.x and **/var/adm/snm** for Solaris 1.x if **SNMDBDIR** is not defined.

enterprises.oid

One of the input files supplied with the SunNet Manager product, it contains the mappings for enterprise identifiers. It is normally found in the directory **SNMHOME,** or in **/opt/SUNWconn/snm** for Solaris 2.x and **/usr/snm** for Solaris 1.x if **SNMHOME** is undefined.

snmp.oid

One of the input files supplied with the SunNet Manager product, it contains the mappings for object identifiers for MIB-I of the SNMP protocol. It is normally found in the directory **SNMHOME/agents,** or **/opt/SUNWconn/snm/agents** for Solaris 2.x and **SNMHOME/agents,** or **/usr/snm/agents** for Solaris 1.x if

SNMHOME is undefined.

NAME

build_tt – build textual convention types database

SYNOPSIS

build_tt [mib1 mib2 ... mibN]

DESCRIPTION

build_tt builds the textual convention types database. The **mib2schema**(1) utility uses this database to look for new IMPORT definitions local to a MIB.

When **mib2schema**(1) is parsing a MIB and it runs into an IMPORT clause which it does not understand, it will display a message for the user to run the **build_tt** utility against the MIB file that has the IMPORT definition for the type or object. Whenever **build_tt** is invoked on a MIB file, the internal textual convention type database is updated with the new values.

OPTIONS

build_tt has no options but may take MIB names as arguments.

The output file *tt.dbase* is written to the directory specified by the **SNMDBDIR** environment variable. If **SNMDBDIR** is undefined, **/var/opt/snm** for Solaris 2.x or **/var/adm/snm** for Solaris 1.x is used as the default database directory.

USAGE

Users should run the **build_tt** utility whenever they need to add new textual conventions to the database.

FILES

tt.dbase This file contains the textual conventions and objects referred to when using the IMPORT clause in a MIB. It is written into the directory specified by the **SNMDBDIR** environment variable. This file is written to **/var/opt/snm** for Solaris 2.x and **/var/adm/snm** for Solaris 1.x if **SNMDBDIR** is not defined.

SEE ALSO

mib2schema(1)

NAME

context.pty – context info configuration for SNMPv2 entities

SYNOPSIS

contextId contextIndex contextViewIndex LocalEntity
LocalTime PDst PSrc PContext StorageType Local

DESCRIPTION

The configuration file **context.pty** is one of several configuration files required by the SNMPv2 entities. The default location of **context.pty** is **/etc/opt/snm/agent** or **/etc/opt/snm/manager** for Solaris 2.x and **/etc/snm/agent** or **/etc/snm/manager** for Solaris 1.x, but can be specified by the environment variable **SR_MGR_CONF_DIR** or **SR_AGT_CONF_DIR.**

The **context.pty** file is used by SNMPv2 agents and managers to define either a party's MIB view or a proxy relationship. For more information, please refer to RFC1447.

Each entry in the file consists of 2 lines:

contextId contextIndex contextViewIndex LocalEntity
LocalTime PDst PSrc PContext StorageType Local

where

contextId

is the context identifier which uniquely identifies this context.

This field is an OID.

contextIndex

is the unique index for this SNMPv2 context. This index is used by the ACL table (see **acl.pty**(5)) when determining access control. There are 0 or more entries in the ACL table (*aclResources*) that point to this value in the context table.

This field is an integer in the range of 1 to 65535, inclusive.

contextViewIndex

determines whether this context is local or remote.

A value greater than zero indicates that this is a local context and is, therefore, an index into the view table. There must exist at least one entry in the view table configuration file, *view.pty*, with a *viewIndex* matching this *contextViewIndex*. The entry in the view table will determine the MIB view available in this context.

See **view.pty**(5) and RFC1447 for more information on the *view.pty* configuration file.

A value of zero indicates a proxy relationship, and further information will be provided by *PDst*, *PSrc*, and *PContext*.

This field is an integer in the range of 0 to 65535, inclusive.

LocalEntity

if the corresponding value for *contextViewIndex* is greater than zero, this field identifies the local entity whose management information is in the context's MIB view:

The empty string, represented by .I –, indicates that the MIB view contains the entity's own local management information.

If not –, the string indicates that the MIB view contains management information of some other local entity, such as *Hub1*.

When the corresponding value for *contextViewIndex* is zero, indicating proxy, this field is ignored, but must contain a value; the place-holder string, –, is a perennial favorite.

This field is an octet string.

LocalTime

if the *contextViewIndex* indicates this is a local context, (i. e., *contextViewIndex* > 0), identifies the temporal context of the management information in the MIB view.

Possible values for this field are:

currentTime — refers to management information at the present time.

restartTime — refers to management information upon the next re-initialization of the managed device.

cacheTime.N where *N* is the number of seconds, refers to management information that is in cache and is guaranteed to be at most *N* seconds old.

This field is an OID.

PDst if this is a proxy relationship (i. e., *contextViewIndex* = 0), this field identifies a SNMPv2 party which is the proxy destination of a proxy relationship.

If the value of *contextViewIndex* is greater than zero, this field must have a value of "0.0".

This field is an OID and is the *contextProxyDstParty* field referred to in RFC1447.

PSrc is the *contextProxySrcParty* field. If the corresponding value for *contextViewIndex* is equal to zero, this field identifies a SNMPv2 party which is the proxy source of a proxy relationship. The interpretation of this value depends on the value of the transport domain (see **agt.pty**(5) and RFC1447) associated with the SNMPv2 party used as the proxy destination in this proxy relationship.

If the value of *contextViewIndex* is greater than zero, this field must have a value of "0.0".

This field is an OID.

PContext

is the *contextProxyContext* field. If the corresponding value for *contextViewIndex* is equal to zero, then the value for this field identifies the context of a proxy relationship. Interpretation of an instance of this object depends on the value of the transport domain (see *TDomain* in **agt.pty**(5)) associated with the SNMPv2 party used as the proxy destination in this proxy relationship.

If the value of *contextViewIndex* is greater than zero, this field must have a value

of "0.0".

This field is an OID.

StorageType
indicates the storage type for this row in the party table. Possible values are:

other
volatile
nonVolatile
permanent

According to RFC1447,

- volatile is lost upon reboot, e. g., in RAM,
- nonVolatile is backed up by stable storage, e. g., in NVRAM,
- permanent cannot be changed or deleted, e. g., in ROM,

and "other" is provided in the unlikely event that someone will find a need for a storage type not covered by the other three.

This field is a case-sensitive string corresponding to one of the above values.

Local indicates to an agent whether this context is local to this SNMPv2 agent. Local indicates "residing at the address with the entity."

This field has two possible values:

true – it is local

or

false – it is not local

Local is used in conjunction with *contextViewIndex* and causes different responses depending on its value.

The only time *Local* is allowed to be *false,* is when *contextViewIndex* is 0, indicating this is a proxy context, and the incoming PDU is a proxy get-response. Otherwise, a *Local* of *false* will cause the counter *snmpStatsUnknownContexts* to be incremented, and the packet will be dropped silently.

The field is a case-sensitive string corresponding to one of the above values.

EXAMPLE

The context entry:

initialContextId.192.147.142.16.1 1 1 –
currentTime 0.0 0.0 0.0 nonVolatile true

defines this context as follows:

initialContextId.192.147.142.16.1 is the *contextId* and the *contextIndex* is one.

The *contextViewIndex* has a value of one, indicating this is *not* a proxy and that the MIB view for this context is all subtrees in the view table that have a *viewIndex* of one.

The *LocalEntity* field is the empty string indicating that the information is for the local entity.

The *LocalTime* field has a value of *currentTime,* and the *PDst, PSrc,* and *PContext* fields have a value of "0.0" since this is not a proxy but a local context.

Store this context in nonVolatile storage, such as NVRAM.

Local field is true, indicating this is a local context executing at the local address.

FILES

Additional SNMPv2 Configuration Files

When the agent is compiled with either SNMPv2 or both SNMPv1 and SNMPv2 defined, the configuration files **acl.pty, agt.pty, context.pty, snmpv2d.conf** and **view.pty** are required for the agent side, and **acl.pty, context.pty, mgr.cnf, mgr.pty,** and **view.pty** are required for the manager side.

acl.pty Access control privileges for the SNMPv2 parties.

agt.pty Initial party table information for the agent.

context.pty
Context information for the SNMPv2 parties.

mgr.cnf
Configuration information for the managers.

mgr.pty
Initial party table information for the managers.

snmpv2d.conf
Configuration information for the SNMPv1 entities.

view.pty
MIB view information for the SNMPv2 parties.

For Solaris 2.x, the files are located under:

/etc/opt/snm/{agent,manager}/acl.pty
/etc/opt/snm/agent/agent.pty
/etc/opt/snm/{agent,manager}/context.pty
/etc/opt/snm/manager/manager.cnf
/etc/opt/snm/manager/manager.pty
/etc/opt/snm/agent/snmpv2d.conf
/etc/opt/snm/{agent,manager}/view.pty

For Solaris 1.x, the files are located under:

/etc/snm/{agent,manager}/acl.pty
/etc/snm/agent/agent.pty
/etc/snm/{agent,manager}/context.pty
/etc/snm/manager/manager.cnf
/etc/snm/manager/manager.pty
/etc/snm/agent/snmpv2d.conf
/etc/snm/{agent,manager}/view.pty

SEE ALSO

v2install(1), **acl.pty**(5), **agt.pty**(5), **context.pty**(5), **mgr.cnf**(5), **mgr.pty**(5), **snmpv2d.conf**(5), **view.pty**(5), SNMP RFCs (RFC1155 RFC1157 RFC1212 RFC1213 RFC1215, RFCs 1441-1452)

NAME

discover.conf – configuration file for SunNet Manager Discover Tool

SYNOPSIS

$SNMDISCOVERMAP/discover.conf

DESCRIPTION

The configuration file **discover.conf** contains parameters for certain less commonly used features of the Discover Tool. The file is stored in the directory specified by the value of the environment variable **$SNMDISCOVERMAP .** The **discover.conf** file is located in the directory defined by **SNMDBDIR.** If **SNMDBDIR** is undefined, then **/var/opt/SUNWconn/snm** for Solaris 2.x or **/var/adm/snm** for Solaris 1.x is the default directory.

You need to deal with **discover.conf** only if you want to:

- modify the shape or the color of the glyphs that represent the elements found by the Discover Tool;
- use system descriptions returned by SNMP agents to specify which type of element to create;
- have created customized views that you want to monitor;
- use the Discover Tool to find agents in addition to the agents shipped with SNM.

SYNTAX

The syntax for **discover.conf** is standard for Unix configuration files. The file is an ASCII file, composed of specific types of entries. The file contains individual sets of configurable fields, each set identified by the following uppercase keywords:

OID
COLORS
COMPONENTS
DEFAULTS
MAPPINGS
MONITOR COMPONENTS
AGENTS

In the **discover.conf** file, each of these keywords must be preceded by a hash mark (#). Except when preceding a keyword, a hash mark indicates a comment.

Within **discover.conf,** the order of fields is significant.

OID Section

The OID section maps object identifiers (OIDs) to element types. An element type determines the glyph used for a given element. An entry has the form <*oid*> <*element_type*>, where <*element_type*> is in a schema file that has been loaded into the Console. For example:

```
#OID
#
Sun Microsystems.2.1.1          component.ss10
HP.2.3.2.5                      component.hpsnake
```

When a device is reachable through SNMP, the discover function tries to retrieve the OID for that device. If successful, the discover function maps the OID to the element type specified in **discover.conf.** Using the example above, a machine returning the OID for Sun (which can be in either the textual form of the example or the numeric 1.3.6.1.4.1.42.2.1.1.1) is created as the SNM element type **component.ss10.** Using the same example, if the device returns HP.2.3.2.5 the element type is **component.hpsnake.**

COLORS Section

The COLORS section of **discover.conf** allows you to create your own color names for the two sections, COMPONENTS and DEFAULTS, that follow COLORS. As shipped with SNM, the COLORS section is as follows:

```
#COLORS
#
RED             255 0 0
BLUE            0 0 255
ORANGE          255 0 255
YELLOW          255 255 75
```

See the bottom area of an element's property sheet in the SNM Console for slider bars that show values for different colors.

COMPONENTS Section

The COMPONENTS section of **discover.conf** associates element types with the colors defined in the COLORS section. For example:

```
#COMPONENTS
#
component.ss1           ORANGE
component.server        LIGHT_BLUE
bus.ethernet            RED
view.subnet             BROWN
view.network            DARK_GREEN
component.link          BLACK
```

DEFAULTS Section

The DEFAULTS section of **discover.conf** allows you to assign a specific color for all SNMP devices. For example:

```
#DEFAULTS
#
snmp_color      GREEN
```

With the entry above, glyphs for all SNMP devices will be colored green.

MAPPINGS Section

The MAPPINGS section allows you to map a keyword returned from the SNMP system description to an SNM element type. If a keyword has whitespace, it must be enclosed in quotes. The Discover Tool has the Sun machine types listed below as defaults.

```
"sparcstation 10"       component.ss10
"sparcstation 1"        component.ss1
"sparcstation 2"        component.ss2
"sparcstation 330"      component.ss330
"sparcstation 370"      component.ss370
sun386                  component.sun386
sun3                    component.sun3
sun4                    component.sun3
sun470                  component.sun3
sc2000                                  component.sc2000
sc1000                                  component.sc2000
ipc                     component.ipc
ipx                     component.ipx
lx                      component.lx
```

The mappings in the **discover.conf** file are consulted before the defaults shown above. For example, if the system description returned by the remote SNMP agent is:

SNMP agent ibm pc

and you have the following entries in your **discover.conf** file:

```
#MAPPINGS

"ibm pc"        component.pc
bridge          component.bridge
```

the type for the element created by the Discover Tool will be **component.pc.** Note that the keyword can appear anywhere in the system description returned by the SNMP agent. Using the preceding example, if a system description is "better to build a bridge", the Discover Tool will create a component of the type **component.bridge.**

MONITOR COMPONENTS Section

The MONITOR COMPONENTS section enables the monitor function to monitor views of types other than networks and subnets. An entry in this section has the following form:

<view type>

where *<view type>* identifies the type of view that you want to the monitor function to monitor. The view type and view instance must be successfully loaded in the Console. The view type must have a netmask field; for the view instance, the netmask field must have a non-null value.

The view types shipped with SNM do not qualify for monitoring because they do not have a netmask field. To see the list of view types, invoke Edit>Create in the Console's control area, then click SELECT on the View category in the Create Object window.

AGENTS Section

The AGENTS section enables the Discover Tool to search for agents in addition to those shipped with SNM. An entry has the following form:

<RPC_number> <agent_name> proxy

The fields in this entry are described as follows:

<RPC_number>
The RPC number of the agent as it is specified in the **/etc/services** file or a directory service map or table.

<agent_name>
The agent name as it appears in the properties sheet for an element. Note that this is not **na.** *<name>*, but simply *<name>.<agent_name>* must identify an agent that is in a schema file that is loaded in the Console.

proxy If the agent is a proxy agent, use the word proxy to indicate its proxy status.

AGENT Schema Section

The agent schemas for a specific type of device is configurable. The agent schemas should be listed in this section of the **discover.conf** file. The discover application will automatically activate the agent schemas for all the device types listed in this section along with the existing defaults. The configuration information has to be entered in a specific format. Following is an example of an entry in this section:

```
#
# AGENT SCHEMA MAPPING
#
<snm element type 1>
{
        agentschema1
        agentschema2
        .
        .
        .
        agentschema n
}
<snm element type 2>
{
        agentschema1
        agentschema2
        .
        .
        .
        agentschema n
}
.
.
.
<element type n>:  These types are specified in the #OID section.
<agentschema n>:  Agent schema name.
```

For example:

```
component.ssio
{
        cpustat
        diskinfo
}
```

NAME

getagents – get Site / SunNet / Domain Manager agent environment

SYNOPSIS

host# **getagents**

No command-line parameters are accepted. The program will ask for all needed information.

DESCRIPTION

getagents sets up an agent environment on a local machine (a machine other than the manager station, which already has agents). It supports both SunOS 4.x and 5.x machines.

getagents copies the agent binaries and SNMP proxy schema from the manager station to the local machine. It also:

- creates a directory **/var/opt/SUNWconn/snm** on Solaris 2.x and **/var/adm/snm** on Solaris 1.x for Site / SunNet / Domain Manager log files
- creates symbolic links from the shared library files to **/usr/lib** (**SunOS 4.x only**)
- creates or updates **snm.conf**(5)
- adds agent entries to **rpc**
- adds two SNMP entries to **services (4)**
- creates **/var/opt/SUNWconn/snm/snmp.hosts** on Solaris 2.x and **/var/adm/snm/snmp.hosts** on Solaris 1.x for use by the SNMP proxy agent
- updates **inetd.conf (4)**
- asks **inetd (1M)** to reread its configuration file
- creates a sample database file you can add to your management database on the manager station

NOTES

getagents must be available to the local machine — it's usually copied from the manager station. If the Site / SunNet / Domain Manager distribution is not available to the local machine via an NFS mount, the manager station must have an entry in its **/.rhosts** file for the local machine while **getagents** is running.

getagents must be run as root on the local machine.

There are two sets of agents supplied with Site / SunNet / Domain Manager, one for SunOS 5.x and the other for SunOS 4.x systems. **getagents** will get the appropriate agents, based on the operating system of the machine where **getagents** is run. Agents supplied with Site / SunNet / Domain Manager run only on Sun-4 systems under SunOS 4.0 or later. Proxy agents can get information from devices other than Sun systems, but they must run on a Sun. **getagents** won't prevent you from copying agents to a non-Sun machine, but the agents probably won't run there.

SEE ALSO

inetd.conf (4),
services (4),
snm.conf(5),
inetd (1M)

Site/SunNet/Domain Manager Administration Guide

NAME

linkmap – format of SunNet Manager linkmap file

DESCRIPTION

This file is used by SNM Console to provide link management capability.

This is an ASCII (text editable) file containing four position-dependent fields, separated by whitespace (tabs or spaces) and terminated with a newline.

The first field is required, and at least one of the other fields are required, dependingon the utility using the file. A "-" takes the place of any field not filled in, including the fourth field. Blank lines are allowed. Comments start with "#" and continue through the end of the line.

The location of the file is defined by the environment variable **SNMLINKMAP.** If this is undefined, the default is **/var/opt/snm/linkmap** for Solaris 2.x and **/var/adm/snm/linkmap** for Solaris 1.x.

The contents of the linkmap file in its basic format are as follows:

<*hostname*> <*ifIndex*> <*linkname*> <*mac address*>

where:

<*hostname*>
indicates the name of the proxy host doing the link monitoring. This is a required parameter.

<*ifIndex*>
indicates the if number of the interface to be monitored.

<*linkname*>
indicates the name given to a particular link.

<*mac address*>
the IP address of the interface. At the current time, the MAC address is not used by the SNM 2.3 product.

EXAMPLES:

If router1, interface 1 is connected to router2, interface 5, and the link name is link1, the resulting linkmap file would look like:

```
router1  1  link1  -
router2  5  link1  -
```

NOTES

The links must be connection type objects (connection.link), NOT simple connections.

NAME

mgr.cnf – SNMPv2 manager configuration

SYNOPSIS

```
Clustername
readInformation         | –
writeInformation        | –
noAuthClkSyncInformation | –
md5ClkSyncInformation   | –
adminInformation        | –
```

DESCRIPTION

The configuration file **mgr.cnf** is one of five files required by the SNMPv2 manager routines. The default location of **mgr.cnf** is **/etc/opt/snm/manager** for Solaris 2.x and **/etc/snm/manager** for Solaris 1.x, but can be specified by the environment variable **SR_MGR_CONF_DIR.**

The **mgr.cnf** file defines clusternames to be used as shorthand references to the clusters of information that define the access privileges and authentication protocol used for the request.

Each cluster of information contains six lines:

```
Clustername
readInformation         | –
writeInformation        | –
noAuthClkSyncInformation | –
md5ClkSyncInformation   | –
adminInformation        | –
```

where

Clustername
: is a unique name for the cluster. This field is a string and must be present.

readInformation
: defines the parties and context that should be used when performing gets, getnexts, getones, and getbulks.

  This field is three OIDs. A "–" signifies a NULL record for that particular line.

writeInformation
: defines the parties and context that should be used when performing sets.

  This field is three OIDs. A "–" signifies a NULL record for that particular line.

noAuthClkSync
: and

md5ClkSync
: define the parties and context that should be used for clock synchronization if using the read, write, or admin record and the *authType* of the corresponding parties is md5. These fields are each three OIDs. A "–" signifies a NULL record for the particular line.

adminInformation

defines the parties and context that should be used for system administrator or "root" level activities.

Note: The *adminInformation* record is an unsupported feature in release 12 and is reserved for future use. The value "–" should be used for this field at this time.

EXAMPLE

A read only **mgr.cnf** entry might look like:

```
Test
jmja1 jajm2 initialContextId.128.169.4.16.1
–
–
–
–
```

This clustername can read, but it cannot write, request authentication protocols above the "no-authorization" level, or perform administrator-level functions.

Note: The parties jmja1 and jajm2 would need to be defined in the MIB and the **mgr.pty**(5) file. The initialContextId.128.169.4.16.1 would be defined in the **context.pty**(5) file.

FILES

Additional SNMPv2 Configuration Files

When the agent is compiled with either SNMPv2 or both SNMPv1 and SNMPv2 defined, the configuration files **acl.pty, mgr.pty, context.pty,** and **view.pty** are required.

acl.pty Access control privileges for the SNMPv2 parties.

context.pty
Context information for the SNMPv2 parties.

mgr.pty
Party information for the manager routines.

view.pty
MIB view information for the SNMPv2 parties.

For Solaris 2.x, the files are located under:

/etc/opt/snm/manager/acl.pty
/etc/opt/snm/manager/context.pty
/etc/opt/snm/manager/mgr.cnf
/etc/opt/snm/manager/mgr.pty
/etc/opt/snm/manager/view.pty

For Solaris 1.x, the files are located under:

/etc/snm/manager/acl.pty
/etc/snm/manager/context.pty
/etc/snm/manager/mgr.cnf
/etc/snm/manager/mgr.pty
/etc/snm/manager/view.pty

SEE ALSO

v2install(1), **acl.pty**(5), **agt.pty**(5), **context.pty**(5), **mgr.pty**(5), **snmpv2d.conf**(5), **view.pty**(5),
SNMP RFCs (RFC1155 RFC1157 RFC1212 RFC1213 RFC1215, RFCs 1441-1452)

NAME

mgr.pty – party configuration for SNMPv2 managers

SYNOPSIS

PartyName PartyDiscriminator
TDomain TAddress Port Lifetime MaxMsgSize
partyIndex partyStorageType partyLocal partyAuthClock
AuthPublicSecret | –
AuthPrivateSecret | –
PrivPublicSecret | –
PrivPrivateSecret | –

DESCRIPTION

The configuration file **mgr.pty** is one of several configuration files required by the SNMPv2 entities. The default location of **mgr.pty** is **/etc/opt/snm/manager** for Solaris 2.x and **/etc/snm/manager** for Solaris 1.x, but can be specified by the environment variable **SR_MGR_CONF_DIR.**

The **mgr.pty** file is similar to the **agt.pty**(5) file used by agents; it defines the party table entries for the parties associated with the managers.

Each entry in the file consists of 7 lines:

PartyName PartyDiscriminator
TDomain TAddress Port Lifetime MaxMsgSize
partyIndex partyStorageType partyLocal partyAuthClock
AuthPublicSecret | –
AuthPrivateSecret | –
PrivPublicSecret | –
PrivPrivateSecret | –

where

PartyName
is a unique name for the party.

This field is required and must map to an OID in the MIB.

PartyDiscriminator
defines the authentication protocol to be used by this party:

1 or 2 defines this party as noauth / nopriv
3 or 4 defines this party as auth / nopriv
5 or 6 defines this party as auth / priv

This field is an integer and must be present for SNMPv2 parties. If the corresponding *TDomain* is *rfc1157Domain,* this field is ignored. However, a value must be present, and an entry of 1 would be reasonable.

TDomain
defines the transport domain for the party. There are two valid entries for the *TDomain:*

rfc1157Domain — indicates this is a SNMPv1 community entry in the party table.

snmpUDPDomain — indicates this is a SNMPv2 party entry.

Note: A party can be *rfc1157Domain* only if the entity is a bilingual entity, i. e., understands both SNMPv1 and SNMPv2. This file is not consulted when the entity is compiled to be SNMPv1 only.

TAddress
used in conjunction with the *Port* field to define the destination address.

This field must be present and is an IP address in dotted decimal form; i. e., 128.169.4.4.

Port used in conjunction with the *TAddress* field to define the destination address.

This field must be present and is an integer.

partyIndex
used by the *aclTable* to match a party entry with its access privileges.

Note: This must be a unique value for each party table entry.

This field is an integer in the range of 1 to 65535, inclusive.

partyStorageType
indicates the storage type for this row in the party table. Possible values are:

other
volatile
nonVolatile
permanent

According to RFC1447,

- volatile is lost upon reboot, e. g., in RAM,
- nonVolatile is backed up by stable storage, e. g., in NVRAM,
- permanent cannot be changed or deleted, e. g., in ROM,

and "other" is provided in the unlikely event that someone will find a need for a storage type not covered by the other three.

This field is a case-sensitive string corresponding to one of the above values.

partyLocal
When *TDomain* is *rfc1157Domain*, this field is ignored.

When *TDomain* is *snmpUDPDomain*, indicates whether this party represents the "local" end of a transmission.

This field has two possible values:

true: it is "local,"
or
false: it is not "local,"

Note: When the party is "not local," and the *TDomain* is *snmpUDPDomain* (i. e., SNMPv2), the *TAddress* and *Port* either may be used by a proxy entity to

determine which address / port pair should receive proxy requests or to indicate which address / port pair should receive traps.

This field is required (whether representing a SNMPv1 or SNMPv2 party) and is a case-sensitive string corresponding to one of the above values.

Note: When the *TDomain* indicates a SNMPv2 party, the term "local" does not mean local in the sense of location or address, but rather in the sense of origin. This means that on outgoing packets, the source party must be a "local" party–one representing the local entity as the source of the packet; on incoming packets, the source party must be a "remote" party–one representing a valid source party for sending packets to the local entity. In this sense, local means representing the local entity. "Non-local" or "remote" parties are ones that are logically remote to the entity; i. e., the source party of an incoming packet–the party that sent the packet–and the destination party of an outgoing packet–the party that will be receiving the packet at the other end of the transmission.

In other words, packets are received from and sent to "non-local" (or remote) parties, and packets are sent from and received by local parties.

Note: entries that are "shared" with another entity will have *partyLocal* fields that are opposite. For instance, an entry for a local party (*partyLocal* = *true*) in the agent's party file would be configured as a non-local (*partyLocal* = *false*) in the manager's party file.

partyAuthClock

defines the current notion of time for the entity.

On startup a pre-defined adjustment, *TIME_WARP* in **secure.h,** is added to this value–--unless the resulting value would exceed the maximum possible time. This is done to help prevent replay attacks across reboots.

This field is an integer in the range 0 to 4294967295, inclusive. A perfectly valid initial value is 0; the security software should adjust and "synch" clocks once everything is up and running.

AuthPublicSecret

is unused by the entity at this time. Will represent the "secret" as a series of hexadecimal numbers, each digit representing the corresponding ASCII value for the character in the string. A value of - represents the null string. The string can range from 0 to 16 bytes.

Note: This field should be null (–) for now.

AuthPrivateSecret

represents the "secrets" string for the authentication protocol. The secret is stored as a series of hexadecimal numbers; each digit is the ASCII value for the corresponding character in the string. A value of – represents the null string. The string can range from 0 to 16 bytes, depending on the *TDomain:*

If the party is an *auth/nopriv* or *auth/priv* party, this field must have a

length of 16 bytes (i.e., 16 hexadecimal numbers in the configuration file). Both digits of each number must be represented. In other words, *1* would be entered as *01*.

If the corresponding *TDomain* field is equal to *rfc1157Domain,* contains a community string name as an encoded string of characters and can be any length. For example, if using the community name *public* the entry would be "70 75 62 6c 69 63". The command

echo *string* | od –x

is very useful for acquiring the correct encoding.

It would be practical to mirror entries in the **snmpv2d.conf**(5) file, but not necessary

Note: Valid only when the *PartyDiscriminator* indicates authentication (i. e., *PartyDiscriminator* > 2), or *TDomain* indicates SNMPv1 community.

PrivPublicSecret

is unused by the entity at this time. Will store the "secret" as a series of hexadecimal numbers, each digit representing the ASCII value for the character in the string. A value of – represents the null string. The string can range from 0 to 16 bytes.

Note: This field should be null (–) for now.

PrivPrivateSecret

represents the "secrets" string for the authentication protocol. The secret is stored as a series of hexadecimal numbers; each digit is the ASCII value for the corresponding character in the string. A value of – represents the null string.

If the party is an *auth/priv party*, this field must have a length of 16 bytes (i.e., 16 hexadecimal numbers in the configuration file). Both digits of each number must be represented. In other words, 1 would be entered as 01.

Note: Valid only when the *PartyDiscriminator* indicates privacy, i. e., *PartyDiscriminator* > 4).

EXAMPLES

The party table entry:

```
initialPartyId.192.147.142.16.1 3
snmpUDPDomain 192.147.142.16 161 300 1458
1 nonVolatile false 0
–
74 68 69 73 74 68 69 73 74 68 69 73 74 68 69 34
–
–
```

defines a party as follows:

The name of this party is *initialPartyId.192.147.142.16.1,* and it is an *auth/nopriv* party.

TDomain is *snmpUDPDomain,* indicating this is a SNMPv2 party.

TAddress and *Port* fields indicate traps or proxy messages will be sent to 192.147.142.16 / 161.

The *Lifetime* of the message is set to 300, indicating that the message will be valid as long as it is received by the target at a time *greater than* the target's idea of "now" minus 300 seconds.

The message can be no longer than 1458 bytes.

The unique party index is 1.

Store this party in non-volatile storage (e. g., NVRAM).

This party is *not* local to the SNMPv2 entity.

The initial clock will be 0 plus *TIME_WARP.*

The *AuthPublicSecret* is unused, so there is a null string for its value.

The *AuthPrivateSecret* is initialized to

74 68 69 73 74 68 69 73 74 68 69 73 74 68 69 34

which decodes to *thisthisthisthi4.*

The *PrivPublicSecret* is unused, so there is a null string for its value.

Since the *PartyDiscriminator* indicates *nopriv,* the *PrivPrivateSecret* is unused, and there is a null string for its value.

Example of a Community Entry

The party table entry:

```
# public
initialPartyId.192.147.142.16.31 1
rfc1157Domain 192.147.142.16 162 300 1458
31 nonVolatile true 0
–
70 75 62 6c 69 63
–
–
```

defines a party as follows:

The name of this party is *initialPartyId.192.147.142.16.31,* and the *PartyDiscriminator* is ignored since this is a community.

There must be value in the *PartyDiscriminator* field; a reasonable entry would be 1 (noauth / nopriv in the SNMPv2 world). This *is* essentially a *noauth/nopriv* since it is a SNMPv1 community, but the actual values of the *partyAuthProtocol* will be *rfc1157noAuth* and *partyPrivProtocol* will be *noPriv.*

Since the *TDomain* is *rfc1157Domain,* this is a SNMPv1 community entry in the party table.

The *partyLocal* is field is ignored since this is a community, but a value must be present.

The *TAddress* and *Port* fields indicate the destination address is port *162* at IP address *192.147.142.16*.

The *Lifetime* is unused since this is a community record, but a value must be present.

The message can be no longer than *1458* bytes.

The unique party index is *31*.

Store this party in non-volatile storage (e. g., NVRAM).

The *partyLocal* and initial clock are both ignored since this is a SNMPv1 community, but there must be entries in those fields.

The *AuthPublicSecret*, *PrivPublicSecret*, and *PrivPrivateSecret* fields are also unused, and therefore null strings represent their values.

AuthPrivateSecret contains the community name:

70 75 62 6c 69 63

which decodes to *public*.

This entry may be mirrored in the **snmpv2d.conf**(5) file.

FILES

Additional SNMPv2 Configuration Files

When the entity is compiled with either SNMPv2 or both SNMPv1 and SNMPv2 defined (bilingual), the configuration files **acl.pty, context.pty, mgr.cnf,** and **view.pty** are required.

acl.pty Access control privileges for the SNMPv2 parties.

context.pty
Context information for the SNMPv2 parties.

mgr.cnf
Clustername configurations for the managers.

view.pty
MIB view information for the SNMPv2 parties.

For Solaris 2.x, the files are located under:

/etc/opt/snm/manager/acl.pty
/etc/opt/snm/manager/context.pty
/etc/opt/snm/manager/manager.cnf
/etc/opt/snm/manager/manager.pty
/etc/opt/snm/manager/view.pty

For Solaris 1.x, the files are located under:

/etc/snm/manager/acl.pty
/etc/snm/manager/context.pty
/etc/snm/manager/manager.cnf
/etc/snm/manager/manager.pty
/etc/snm/manager/view.pty

SEE ALSO

v2install(1), **acl.pty**(5), **agt.pty**(5), **context.pty**(5), **mgr.cnf**(5), **snmpv2d.conf**(5), **view.pty**(5),
SNMP RFCs (RFC1155 RFC1157 RFC1212 RFC1213 RFC1215, RFCs 1441-1452)

NAME

mib2schema – translates MIB files to SunNet Manager schema files.

SYNOPSIS

mib2schema *mib_filename [schema_filename]*

DESCRIPTION

mib2schema translates MIB files for SNMP agents to the format required by SunNet Manager's agent schema files.

MIB files must conform strictly to the RFC 1155 SMI ("Structure and Identification of Management Information for TCP / IP-based Internets"). MIB files must also conform to the subset of ASN.1 syntax specified in RFC 1155. MIB files may also include "Concise MIB Definitions" as specified in RFC 1212 and the TRAP-TYPE macro, as defined in "A Convention for Defining Traps for use with the SNMP" (RFC 1215).

mib2schema also generates two other files: an ∗.*oid* file and a ∗.*traps* file. The ∗.*oid* file is an object identifier file that contains a table of object identifiers and names. This file is used by the **build_oid** program. The ∗.*traps* file contains traps definitions; this file may or may not be generated, depending on whether traps were specified in the MIB.

It is important to note that **mib2schema** is not a compiler for MIB specifications. In fact, **mib2schema** expects the MIB specification to conform strictly to the SMI defined in RFC 1155 to work correctly. However, **mib2schema** does perform some primitive error detections. If any syntax error is reported, **mib2schema** aborts immediately. If warnings are detected, **mib2schema** generates warning messages but generates the schema file anyway.

It is now possible for **mib2schema** to look for new IMPORT definitions local to a MIB. Please see the **build_tt**(1) man page for more detailed information.

Please note the following:

1. SNM schemas do not support nested groups and tables; therefore, **mib2schema** does not translate them. If your MIB uses nested groups or tables, you need to rewrite the groups and tables as un-nested groups or tables.
2. SNM schemas do not support ranges defined for data types. Therefore, **mib2schema** ignores ranges defined for data types.
3. **mib2schema** recognizes only the following set of data types as defined in RFC 1155: INTEGER, OCTET STRING, SEQUENCE and SEQUENCE OF.
4. **mib2schema** recognizes only the following set of defined types: NetworkAddress, IpAddress, Counter, Gauge, Timeticks and Opaque, as defined in RFC 1155; and PhysAddress and DisplayString, as defined in RFC 1213 (MIB-II). All defined types must eventually resolve to one of the data types listed in 3.
5. As defined in RFC 1155, you may not specify 0 as an object identifier or as an enumerated value. However, **mib2schema** would still generate the schema file, but warnings would be generated.
6. The key value in the characteristics field for tables cannot be determined in all cases from the INDEX statement in the MIB. In instances when the MIB contains no INDEX clauses or when **mib2schema** cannot determine the key value, a warning is generated.

7. Although **mib2schema** parses DEFVAL clauses, it currently ignores them.
8. **mib2schema** generates only the objects found in the enterprise-specific MIB file into the schema file. If you want to include objects in the standard MIB-I schema file into the enterprise specific schema file, you would have to merge them yourself using your favorite editor.

OPTIONS

If only one file name is specified, **mib2schema** translates the MIB specifications in *mib_filename* to SunNet Manager schema format in *mib_filename.schema.* **mib2schema** also generates *mib_filename.oid* and, optionally, *mib_filename.traps.* If two file names are specified, **mib2schema** translates the MIB specifications in *mib_filename* to SunNet Manager schema format in *schema_filename.*

NOTES

This section describes some common errors that occur when writing MIBs. Reading this section may help to resolve any problems encountered when running **mib2schema.**

Sample MIB Module

Every MIB must have a module name, a BEGIN, an optional IMPORTS statement, the actual MIB body of definitions, and an END. The following is a skeletal structure of a MIB module:

```
FOO-MIB DEFINITIONS ::= BEGIN
-- FOO is the name you would give to the MIB module
IMPORTS
        -- Objects which you import, usually from
        -- RFC1155-SMI, for example:
        experimental, OBJECT-TYPE FROM RFC1155-SMI;
foo OBJECT IDENTIFIER ::= { enterprise 888 }
-- The above statement defines foo as an
-- object identifier with an enterprise number
-- of 888.
-- BODY OF DEFINITIONS
END
```

Note that ASN.1 syntax is case-sensitive, so it is very important to get the case right. For example, object variables and identifiers MUST start with a lowercase letter, while object types MUST start with an uppercase letter. Names cannot include underscore (_) characters. Failure to follow these conventions will result in **mib2schema** generating a syntax error and aborting. Comments are preceded by a pair of hyphens.

Common errors when writing MIBs

1. MIBs must be written in the SMI syntax, which is a subset of ASN.1 syntax. **mib2schema** does not accept MIBs written in ASN.1 syntax outside of the SMI subset.
2. ASN.1 syntax is case-sensitive. Be very careful when typing in names and types. Remember that types must begin with an uppercase letter while names must begin with a lowercase letter. Names cannot include underscores (_).

3. Typographical errors make up the largest number of errors detected by **mib2schema**. Examples of common typos include (those on the left are typos):
   "enterprise" instead of "enterprises"
   "Timeticks" or "timeticks" instead of "TimeTicks"
   "IPADDRESS" instead of "IpAddress"
   "non-accessible" instead of "not-accessible"
4. Tables MUST be made up of entries; they cannot have primitive types as entries. For example, if you defined:

```
foo OBJECT-TYPE
        SYNTAX SEQUENCE OF Foo-Entry
        ACCESS read-only
        STATUS mandatory
        ::= { bar 1 }
```

you must ensure that Foo-Entry is defined as a SEQUENCE constructor like:

```
Foo-Entry ::= SEQUENCE {
        entry1
                INTEGER,
        entry2,
                INTEGER
}
```

You may not define any other type for Foo-Entry.

Warning/Error Messages

This section describes the warning or error messages **mib2schema** generates and the actions you should take to correct your MIB. Warning messages indicate an inconsistency in your MIB, but nothing serious enough to prevent the generation of a schema file. Warning messages are preceded by the word "Warning" and the line number in your MIB that caused the message to be generated. Error messages are usually fatal, and cause **mib2schema** to abort. Error messages are accompanied by messages displaying the last token that was read from the MIB and the line number of that token in the MIB.

Warning Messages

"does not recognize IMPORTED symbol *[symbol-name]*"

mib2schema only recognizes keywords and objects from the SMI, MIB-I, MIB-II, the Concise MIB revisions and the TRAP-TYPE macro. This warning message is generated if *symbol-name* is not an object from any of the aforementioned documents. *symbol-name* will have to be redefined in the MIB.

Run **build_tt** to build the textual convention types database. This allows **mib2schema** to look for new IMPORT definitions local to a MIB. For more information, refer to the **build_tt**(1) man page.

"*[symbol-name]* previously defined"

If *symbol-name* is defined more than twice, this message is generated. When the *symbol-name* happens to be a group name, it is renamed by inserting its parent node name before its own name. This is necessary since the SNM schema format cannot accept duplicate

group names.

"Object Identifier cannot be 0"

As defined in the SMI, you may not use 0 as an object identifier, since 0 is reserved for instances.

"Enumeration Element Value cannot be 0"

As defined in the SMI, you may not use 0 as an enumerated value.

"*[symbol-name]* is not a valid status type"

As defined in the SMI, you may only use the following status types: "deprecated", "mandatory", "obsolete" and "optional". However, since SNM schema ignores the status types, only a warning message is generated.

"The following INDEX entries in *table* not resolved: *name*"

If **mib2schema** is unable to determine the key value for table attributes, it inserts the characters -K ??? into the characteristics field. Users of SunNet Manager 2.x do not need to take any action. If you intend to use the generated schema file with SNM release 1.x, then you should edit the schema and manually change the occurrences of with the appropriate key value.

Error Messages

"Fatal Error: *[symbol-name]* not defined"

This message is generated if *symbol-name* is part of an object's component ID and if it was not previously defined. Typos are the main reason this message is generated.

"Fatal Error: *[symbol-name]* type not resolved"

This message is generated when *symbol-name* is used as a type by an object variable, but is not defined anywhere in the MIB. Since the object's type cannot be resolved, the translation has to be aborted.

"Fatal error: *[name]* unknown enterprise in trap definition"

This message is generated when an unknown enterprise is encountered in trap definitions. *name* is the unknown enterprise name.

BUGS

As mentioned above, **mib2schema** is not a compiler for MIB specifications. Hence, the behavior of **mib2schema** is not guaranteed if the MIB specification is syntactically incorrect. Agents are not required to accept messages that exceed 484 octets. For groups with very long var-bind entries, you may have to split the group into groups with smaller var-bind entries.

SEE ALSO

build_oid(8), **build_tt**(1), **snm_schema**(5), **na.snmp**(8), **na.snmp-trap**(8)

NAME

na.activity – management activity daemon

DESCRIPTION

na.activity maintains a record of currently active data and event reporting requests started from the local host in the activity logfile specified by the *activity-log* keyword listed in the local **snm.conf**(5)**.**

na.activity periodically requests agents to verify they are performing the data or event reporting request recorded in the activity logfile. **na.activity** removes a request from the activity logfile if the agent does not verify it is performing the request.

na.activity is usually started from **inetd (1M).**

ERRORS

na.activity sends an error message to **syslogd (1M)** if it cannot open the activity logfile for appending.

SEE ALSO

snm.conf(5)

NAME

na.cpustat – per cpu statistics (Solaris 2.x)

DESCRIPTION

na.cpustat returns statistics of all the cpus on the system

A reporting interval is specified by the management application. If a reporting interval is not specified, the agent will send a report every 30 seconds.

ATTRIBUTES

na.cpustat has three attribute tables, **load_stats, system_stats** and **vm_stats .**

The **load_stats** table reports load statistics about a particular cpu , using **cpuno** as the key.

cpuno	Cpu Id as represented in kernel (int)
user	time spent in user mode since system boot (unsigned long)
sys	time spent in kernel mode since system boot (unsigned long)
wio	time spent in the in idle waiting for IO(unsigned long)
idle	time spent in the in idle (unsigned long)
madpE	Adaptive mutex enters for this cpu(unsigned long)
traps	no of traps (unsigned long)
intr	no of interrupts (unsigned long)
migr	no of thread migrated to another cpu (unsigned long)

The **system_stats** table reports system statistics about a particular cpu , using **cpuno** as the key.

cpuno	Cpu Id as represented in kernel (int)
syscall	no of system calls (unsigned long)
swtch	no of context switches (unsigned long)
idlet	no of idle thread scheduled to run (unsigned long)
inctx	no of involuntary context switch (unsigned long)
intrb	no of intrrupt thread blocked (unsigned long)
rwrdf	no of reader-writer locks read fails (unsigned long)
rwwrf	no of reader-writer locks write fails (unsigned long)
fork	no of forks + vforks (unsigned long)
exec	no of exec system calls (unsigned long)
read	no of read system calls (unsigned long)
write	no of write system calls (unsigned long)
bread	no of reads from the disk into buffer cache (unsigned long)
bwrit	no of writes to the disk from buffer cache (unsigned long)
lread	no of reads from buffer cache (unsigned long)
lwrit	no of writes to buffer cache (unsigned long)

pread no of raw device reads (unsigned long)
pwrit no of raw device writes (unsigned long)
namei no of name to inode translations (unsigned long)
iget no of ufs inode get operations (unsigned long)
ipg no of inodes found with pages (unsigned long)
inopg no of inodes found without pages (unsigned long)
dblk no of directory block access (unsigned long)
msgs no of IPC:message operations (unsigned long)
sems no of IPC:semaphore operations (unsigned long)
rawch no of raw characters processed (unsigned long)
canch no of canonical characters processed (unsigned long)
outch no of characters output (unsigned long)
rcvi no of serial port rcv interrupts (unsigned long)
xmti no of serial port transmit interrupts (unsigned long)
mdmi no of modem interrupts (unsigned long)

The **vm_stats** table reports per CPU virtual memory statistics , using **cpuno** as the key.

cpuno Cpu Id as represented in kernel (int)
asflt no of as faults (unsigned long)
htflt no of hat faults (unsigned long)
mjflt no of major faults (unsigned long)
cwflt no of copy-on-write faults (unsigned long)
prflt no of protection faults (unsigned long)
ksflt no of kernel as faults (unsigned long)
pgin no of page in's (unsigned long)
pgpgi no of pages paged in (unsigned long)
pgout no of page out's (unsigned long)
pgpgo no of pages paged out (unsigned long)
swpin no of swap ins (unsigned long)
pswpi no of pages swapped in (unsigned long)
swapout
no of swap outs (unsigned long)
pswpo no of pages swappped out (unsigned long)
dfree no of pages freed by daemon swapped out (unsigned long)
dscan no of pages scanned by daemon to be swapped (unsigned long)
zfod no of pages zero filled on demand (unsigned long)

OPTIONS

None.

ERRORS

Fatal Internal Error
An error internal to cpustat occured; contact the support

System Internal Error
An error occured due to system misbehavior ; contact the support

Data build failed
Error occured while building a response ; contact the support.

Too small a frequency; Agent overloaded
The reporting frequency interval specified was too small for the system to cope up with. Try with a bigger one.

NAME

na.diskinfo — report disk space information

DESCRIPTION

This agent reports the total amount of disk space occupied by a file system, the amount of used and available space and the capacity used.

ATTRIBUTES

na.diskinfo has one attribute table, **diskSpace**.

The **diskSpace** table reports the disk space information of file systems. The key is the directory / file name currently mounted (**mountedOn**).

fileSystem - file system name (string[256])

mountedOn - directory / file name currently mounted (string[256])

total - total blocks in file system (long)

used - used blocks in file system (gauge)

avail - available blocks in file system (gauge)

capacity - the percentage used of the file system's capacity (gauge)

iUsed - the number of inodes used (gauge)

iFree - the number of free inodes (gauge)

%iUsed - the percentage of inodes used (gauge)

ERRORS

statfs(2) failed
cannot check the status of the file system.

data build failed
unable to report the table for building report error.

NOTES

This agent only reports the information for file systems of type 4.2, SWAP, and nfs.

NAME

na.etherif – ethernet interface statistics

DESCRIPTION

na.etherif returns statistics for the Lance (le) and the Intel (ie) Ethernet interfaces. By default the agent reads **/vmunix** to get the kernel namelist.

A reporting interval is specified by the management application. If a reporting interval is not specified, the agent will send a report every 30 seconds.

ATTRIBUTES

na.etherif has two attribute tables, **input** and **output .**

The **input** table reports input statistics about a particular ethernet interface, using **ifname** as the key.

ifname Interface name (string[5])

ipkts Number of packets received since system boot (counter)

ierrs Number of input errors since system boot (counter)

If a particular Ethernet interface's error counts are abnormally low while the error counts for other interfaces are high, the interface with the abnormally low error counts is likely to be the one that is malfunctioning.

ierrate Input error rate since system boot (in thousandths of a percent)

This is an agent-supplied calculation of the input errors over input packets. For simplicity in thresholding and graphing, this value is multiplied by 1000 and returned as a gauge.

Watch for an error rate of less than 25 (0.025%). An error rate higher than this may be due to electrical problems with the transceiver or drop cable, or to packets dropped due to lack of receive buffers. Machines that receive slower than the transmitters (for instance, a Sun-3 server with a bunch of Sun-4 clients) and machines with more than 2 interfaces are most likely to run out of receive buffers.

Under SunOS 4.1 , the file **/sys/sunif/ie_conf.c** contains default values for the number of **ie** interface buffers. **le_conf.c** configures the **le** interfaces. The values of interest are:

ie_rbufs receive buffers
ie_rbds receive buffer descriptors
ie_rfds receive frame descriptors

A buffer and its two descriptors occupy about 1600 bytes. The interface configuration file defines both high and low versions of these parameters. The high values (for example, **ie_high_rbufs**) are used when more than one network interface is present. Otherwise, the low values are used. The number of buffers and associated descriptors may be increased, subject to the following constraint:

ie_rfds < ie_rbds < ie_rbufs

The total size of the Ethernet receive buffers and descriptors must not exceed about 125 Kbytes on a Sun 4 / 4x0 machine, so that buffers for off-board Ethernet

interfaces may be I/O cached.

icrc Number of packets received with CRC errors since system boot (counter)

iover Number of receive overruns since system boot (counter)

ibufs (*SunOS 4.x only*) Current number of receive buffers (gauge)

ibufdesc
(*SunOS 4.x only*) Current number of receive buffer descriptors (gauge)

idiscard
(*SunOS 4.x only*) Number of packets missed/discarded since system boot (counter)

iframerr
Number of packets seen with framing/alignment errors since system boot (counter)

iframedesc
(*SunOS 4.x only*) Current number of receive frame descriptors (gauge) — ie only

imiss (*SunOS 5.x only*) Input packets missed

inocanput
(*SunOS 5.x only*) Number of errors trying to send packets upstream

The **output** table reports output statistics about a particular Ethernet interface, using **ifname** as the key.

ifname Interface name (string[5])

opkts Number of packets sent since system boot (counter)

oerrs Number of output errors since system boot (counter)

ocolls Number of collisions for a given packet since system boot (counter)

odefers
Number of deferred transmissions since system boot (counter). The SunOS 4.0 lance (le) driver doesn't keep this information; it will be returned as zero.

oavail%
Percentage of time (since boot) waiting packets could get onto the wire (gauge)

Each time the Ethernet chip goes to send a packet, the following might happen:

1. Chip detects another transmission on the cable, so defers its own.
2. Chip sends the packet, but it collides with another. The chip then retries later.
3. Chip sends the packet okay.

Other things can happen, like network jams and carrier losses, but we don't worry about those for **oavail% .**

The **ovail%** field indicates the percentage of time a packet was able to go out OK: the "availability" of the network since the system was rebooted.

In mathematical terms,

$$\text{ovail\%} = \frac{\text{packets sent}}{\text{packets sent + packets deferred + collisions}}$$

Or, alternatively, "what percentage of ATTEMPTED packets were SUCCESSFUL?"

If **ovail% =** 30, the system has been spending a lot of time trying to put packets onto a very busy network. The two attributes **ocolls** and **odefers** will point you towards why the availability is so low. It doesn't mean the network load is 70 percent, but numbers that low indicate something is wrong with the system. Or, if these numbers correlate across many machines, you've got too much traffic on the network. See the other attributes (for example, **ojams** or **nodcd)** for possible supporting evidence. If you see a high **nodcd,** perhaps your transceiver or drop cable is bad.

You might use 90 percent as a baseline target and worry about values lower than 85 percent.

ojams (*SunOS 4.x only*) Number of network jams on this interface since system boot (counter)

nodcd Number of times carrier lost since system boot (counter)

ounder Number of transmit underruns since system boot (counter)

odrops (*SunOS 4.x only*) Number of output queue packets dropped since system boot (counter)

watchdog
(*SunOS 4.x only*) Number of watchdog timer resets since system boot (counter). The SunOS 4.0 lance (le) driver doesn't keep this information, so it will be returned as zero.

requeues
Number of transmit requeues / retries since system boot (counter)

obufdesc
(*SunOS 4.x only*) Current number of transmit buffer descriptors (gauge)

onobufs
Number of times driver ran out of transmit buffers since system boot (counter)

copies (*SunOS 4.x only*) Number of output copies since system boot (counter)

nocts (*SunOS 4.x only*) Number of times Clear To Send lost since system boot (counter) — ie only

obufs (*SunOS 4.x only*) Current number of transmit buffers (gauge) — ie only

oframedesc
(*SunOS 4.x only*) Current number of transmit frame descriptors (gauge) — ie only

ofreedesc
(*SunOS 4.x only*) Current number of free transmit buffer descriptors (gauge) — ie only

lcoll Number of late collisions since system boot (counter) — le only

obuferr
(*SunOS 4.x only*) Number of le BUFF errors since system boot (counter) — le only

oinits (*SunOS 5.x only*) Number of times the hardware has been initialized

OPTIONS

The *SunOS 4.x* version accepts one option: the name of an alternate file to use to get the kernel namelist.

The *SunOs 5.x* version has no options.

ERRORS

can't read kernel memory
The agent uid does not have permission to read kernel memory, **/dev/kmem .** Run the agent as root (default case if agent started by **inetd (1M)**), or give read permission to the uid the agent is running under.

The error message can also occur if the agent can't find or read the file containing the kernel namelist.

NOTES

Keep in mind that the **ovail%** attribute is NOT an instantaneous value; it's a measure since boot time. The **etherif** agent currently has no way of returning output availability as an instantaneous value. Although it would be nice to see such a number, the manager should do the calculation. (The Console, alas, can't do mathematical operations between attributes yet.)

The way collision count is tallied depends on the Ethernet chip (Lance or Intel). The Intel (ie) chip counts every occurrence, while the Lance (le) chip counts collision on a per-packet basis. If a packet is successfully sent after the first retry, the collision count is one. If a packet is successfully sent after two or more retries, the chip returns a maximum count of two collisions. For example, if packet *a* had four retries and sometime later packet *d* had five retries, the cumulative collision count is four — two for each packet — and not the real count of nine. This means the **ovail%** value will tend to be (artificially) higher for le devices.

The description on **ierrate** was taken from a document written by Hal Stern of Sun's Northeast Area Consulting Group.

NAME

na.etherif2 – ethernet interface statistics (Solaris 2.x)

DESCRIPTION

na.etherif returns statistics for the Lance (le) and the Intel (ie) Ethernet interfaces.

A reporting interval is specified by the management application. If a reporting interval is not specified, the agent will send a report every 30 seconds.

ATTRIBUTES

na.etherif has two attribute tables, **input** and **output .**

The **input** table reports input statistics about a particular ethernet interface, using **ifname** as the key.

ifname Interface name (string[5])

ipkts Number of packets received since system boot (counter)

ierrs Number of input errors since system boot (counter)

If a particular Ethernet interface's error counts are abnormally low while the error counts for other interfaces are high, the interface with the abnormally low error counts is likely to be the one that is malfunctioning.

ierrate Input error rate since system boot (in thousandths of a percent)

This is an agent-supplied calculation of the input errors over input packets. For simplicity in thresholding and graphing, this value is multiplied by 1000 and returned as a gauge.

Watch for an error rate of less than 25 (0.025%). An error rate higher than this may be due to electrical problems with the transceiver or drop cable, or to packets dropped due to lack of receive buffers. Machines that receive slower than the transmitters (for instance, a Sun-3 server with a bunch of Sun-4 clients) and machines with more than 2 interfaces are most likely to run out of receive buffers.

Under SunOS 4.1 , the file **/sys/sunif/ie_conf.c** contains default values for the number of **ie** interface buffers. **le_conf.c** configures the **le** interfaces. The values of interest are:

ie_rbufs — receive buffers
ie_rbds — receive buffer descriptors
ie_rfds — receive frame descriptors

A buffer and its two descriptors occupy about 1600 bytes. The interface configuration file defines both high and low versions of these parameters. The high values (for example, **ie_high_rbufs**) are used when more than one network interface is present. Otherwise, the low values are used. The number of buffers and associated descriptors may be increased, subject to the following constraint:

ie_rfds < ie_rbds < ie_rbufs

The total size of the Ethernet receive buffers and descriptors must not exceed about 125 Kbytes on a Sun 4 / 4x0 machine, so that buffers for off-board Ethernet interfaces may be I / O cached.

icrc Number of packets received with CRC errors since system boot (counter)

iover Number of receive overruns since system boot (counter)

iframerr
Number of packets seen with framing / alignment errors since system boot (counter)

imiss Input packets missed

inocanput
Number of errors trying to send packets upstream

The **output** table reports output statistics about a particular Ethernet interface, using **ifname** as the key.

ifname Interface name (string[5])

opkts Number of packets sent since system boot (counter)

oerrs Number of output errors since system boot (counter)

ocolls Number of collisions for a given packet since system boot (counter)

odefers
Number of deferred transmissions since system boot (counter). The SunOS 4.0 lance (le) driver doesn't keep this information; it will be returned as zero.

oavail%
Percentage of time (since boot) waiting packets could get onto the wire (gauge)

Each time the Ethernet chip goes to send a packet, the following might happen:

1. Chip detects another transmission on the cable, so defers its own.
2. Chip sends the packet, but it collides with another. The chip then retries later.
3. Chip sends the packet okay.

Other things can happen, like network jams and carrier losses, but we don't worry about those for **oavail% .**

The **ovail%** field indicates the percentage of time a packet was able to go out OK: the "availability" of the network since the system was rebooted.

In mathematical terms,

$$\text{ovail\%} = \frac{\text{packets sent}}{\text{packets sent + packets deferred + collisions}}$$

Or, alternatively, "what percentage of ATTEMPTED packets were SUCCESSFUL?"

If **ovail% =** 30, the system has been spending a lot of time trying to put packets onto a very busy network. The two attributes **ocolls** and **odefers** will point you towards why the availability is so low. It doesn't mean the network load is 70 percent, but numbers that low indicate something is wrong with the system. Or, if these numbers correlate across many machines, you've got too much traffic on

the network. See the other attributes (for example, **ojams** or **nodcd)** for possible supporting evidence. If you see a high **nodcd ,** perhaps your transceiver or drop cable is bad.

You might use 90 percent as a baseline target and worry about values lower than 85 percent.

nodcd Number of times carrier lost since system boot (counter)

ounder Number of transmit underruns since system boot (counter) The SunOS 4.0 lance (le) driver doesn't keep this information, so it will be returned as zero.

requeues
Number of transmit requeues / retries since system boot (counter)

onobufs
Number of times driver ran out of transmit buffers since system boot (counter)

lcoll Number of late collisions since system boot (counter) — le only

oinits Number of times the hardware has been initialized

OPTIONS

None.

ERRORS

can't read kernel memory
The agent uid does not have permission to read kernel memory, **/dev/kmem .** Run the agent as root (default case if agent started by **inetd (1M)**), or give read permission to the uid the agent is running under.

NOTES

Keep in mind that the **ovail%** attribute is NOT an instantaneous value; it's a measure since boot time. The **etherif** agent currently has no way of returning output availability as an instantaneous value. Although it would be nice to see such a number, the manager should do the calculation. (The Console, alas, can't do mathematical operations between attributes yet.)

The way collision count is tallied depends on the Ethernet chip (Lance or Intel). The Intel (ie) chip counts every occurrence, while the Lance (le) chip counts collision on a per-packet basis. If a packet is successfully sent after the first retry, the collision count is one. If a packet is successfully sent after two or more retries, the chip returns a maximum count of two collisions. For example, if packet *a* had four retries and sometime later packet *d* had five retries, the cumulative collision count is four — two for each packet — and not the real count of nine. This means the **ovail%** value will tend to be (artificially) higher for le devices.

The description on **ierrate** was taken from a document written by Hal Stern of Sun's Northeast Area Consulting Group.

NAME

na.event– event dispatcher

DESCRIPTION

na.event receives agent event reports and forwards them to all processes that have registered to receive such reports. In addition, **na.event** records each event report it receives in the event logfile specified by the *event-log* keyword listed in the local **snm.conf**(5)**.**

na.event is usually started from **inetd (1M).**

OPTIONS

Starting **na.event** from the shell command line with the **-d1** option displays debugging information.

ERRORS

na.event sends an error message to **syslogd (1M)** if it cannot open the event logfile for appending.

SEE ALSO

netmgt_register_rendez(3N), **netmgt_unregister_rendez**(3N), **snm.logfile**(5)

NAME

na.hostif – interface statistics

DESCRIPTION

na.hostif returns information about the interfaces that send IP packets on the host where the agent runs. By default, the agent reads **/vmunix** to get the kernel namelist.

A reporting interval is specified by the management application. If a reporting interval is not specified, the agent will send a report every 30 seconds.

ATTRIBUTES

na.hostif has one attribute table, **if ,** that reports statistics on a particular interface, using *name* (le0,...) as the key. If no key is specified, statistics on all interfaces are reported.

If the named interface does not support a particular attribute (for example, an X.25 interface does not have an **ipadrr**), **na.hostif** returns no information for that attribute.

name - interface device name (string[32])

etheraddr - MAC-level address (string[20])

mtu - maximum transfer unit (short)

flags - interface information flags (short)

The agent returns flags in decimal, not hexadecimal, notation. Convert the value to hexadecimal and decode it according to the bit assignments shown below.

```
IFF_UP          0x1     /* interface is up */
IFF_BROADCAST   0x2     /* broadcast address valid */
IFF_DEBUG       0x4     /* turn on debugging */
IFF_LOOPBACK    0x8     /* is a loopback net */
IFF_POINTOPOINT 0x10    /* interface is point-to-point link */
IFF_NOTRAILERS  0x20    /* avoid use of trailers */
IFF_RUNNING     0x40    /* resources allocated */
IFF_NOARP       0x80    /* no address resolution protocol */
IFF_PROMISC     0x100   /* receive all packets */
IFF_ALLMULTI    0x200   /* receive all multicast packets */
IFF_PRIVATE     0x8000  /* do not advertise */
```

ipkts - cumulative number of packets received since system boot (counter)

ierrs - cumulative number of input errors since system boot (counter)

opkts - cumulative number of packets sent since system boot (counter)

oerrs - cumulative number of output errors since system boot (counter)

colls - cumulative number of collisions since system boot (counter)

qlen - current output queue length (gauge)

metric - (*SunOS 4.x only*) current hop count or pseudo-cost routing metric (gauge)

ipaddr - local IP address (**inetaddr**)

ipname - local IP name (string[32])

ipdest - IP network or destination address (**inetaddr**)

ipdestname - IP network or destination name (string [32])

netmask - IP subnet mask (**inetaddr**)

bcaddr - IP broadcast address (**inetaddr**)

OPTIONS

The *SunOS 4.x* version accepts one option: the name of an alternate file to use to get the kernel namelist.

The *SunOS 5.x* version has no options.

ERRORS

can't get broadcast, netmask: ioctl SICGIFFLAGS failed

can't get broadcast address: ioctl SIOCGIFBRDADDR failed

can't get netmask: ioctl SIOCGIFNETMASK failed
See **ioctl (2) .**

can't get broadcast, netmask: socket(2) call failed
See **socket (3N) .**

can't read kernel memory
The agent uid does not have permission to read kernel memory, **/dev/kmem**. Run the agent as root (default case if agent started by **inetd (1M)**), or give read permission to the uid the agent is running under.

The error message can also occur if the agent can't find or read the file containing the kernel namelist.

NOTES

The way collision count is tallied depends on the Ethernet chip (Lance or Intel).

The Intel (ie) chip counts every occurrence.

The Lance (le) chip counts collision on a per-packet basis. More specifically, if a packet is successfully sent after the first retry, the collision count is one. If a packet is successfully sent after two or more retries, the chip returns a maximum count of two collisions. For example, if packet*a* had four retries and sometime later packet*d* had five retries, the cumulative collision count is four — two for each packet — and not the real count of nine.

NAME

na.hostmem – network memory buffer pool usage and statistics

DESCRIPTION

na.hostmem reports usage and utilization statistics for the memory buffer pool used by the network routines. These statistics include streams allocation, mbuf usage, etc. By default the agent reads **/vmunix** to get the kernel namelist.

A reporting interval is specified by the management application. If a reporting interval is not specified, the agent sends a report every 30 seconds.

ATTRIBUTES

The *SunOS 4.x* version of **na.hostmem** has three attribute groups, **mbuf, mbuf_uses** and **streams**, and one attribute table, **dblks**.

The *SunOS 5.x* version of **na.hostmem** has one attribute group, **streams**.

The **mbuf** group reports general statistics about mbuf usage.

mbufused - number of mbufs currently in use (gauge)

mbuffree - number of mbufs currently free (gauge)

mbuf% - percentage of mbufs currently in use (gauge)

mbufmiss - number of mbufs missing (gauge)

clused - number of cluster buffers currently in use (gauge)

clfree - number of cluster buffers currently free (gauge)

cl% - percentage of cluster buffers used (gauge)

memused - number of Kbytes allocated to the network currently in use (gauge)

memfree - number of Kbytes allocated to the network currently free (gauge)

mem% - percentage of ‘‘network’’ memory currently being used (gauge)

nomem - number of requests for memory that were denied (counter)

delay - (*SunOS 4.1.x only*) number of requests for memory that were delayed (counter)

drain - (*SunOS 4.1.x only*) number of calls to protocol drain routines (counter)

space - (*SunOS 4.1.x only*) number of interface pages obtained from page pool (counter)

The **mbuf_uses** group reports information about the number of memory buffers used by particular categories of processes.

data - number of mbufs currently allocated to data (gauge)

header - number of mbufs currently allocated to packet headers (gauge)

socket - number of mbufs currently allocated to socket structures (gauge)

pcb - number of mbufs currently allocated to protocol control blocks (gauge)

rtable - number of mbufs currently allocated to routing table entries (gauge)

htable - number of mbufs currently allocated to IMP host table entries (gauge)

atable - number of mbufs currently allocated to address resolution table entries (gauge)

ftable - number of mbufs currently allocated to fragment reassembly queue headers (gauge)

soname - number of mbufs currently allocated to socket names and addresses (gauge)

zombie - number of mbufs currently allocated to zombie process information (gauge)
soopts - number of mbufs currently allocated to socket options (gauge)
rights - number of mbufs currently allocated to access rights (gauge)
ifaddr - number of mbufs currently allocated to interface addresses (gauge)
unknown - number of mbufs currently allocated to unknown mbuf types (gauge)
The **streams** group reports information about streams usage.
strused - number of used streams (gauge)
strfree - number of free streams (gauge)
str% - percentage of used streams (gauge)
strcum - cumulative number of streams allocated (counter)
strfail - number of streams allocation failures (counter)
qused - number of used queues (gauge)
qfree - number of free queues (gauge)
q% - percentage of used queues (gauge)
qcum - cumulative number of queues allocated (counter)
qfail - number of queue allocation failures (counter)
mblkused - (*SunOS 4.x only*) number of used mblks (gauge)
mblkfree - (*SunOS 4.x only*) number of free mblks (gauge)
mblk% - (*SunOS 4.x only*) percentage of used mblks (gauge)
mblkcum - (*SunOS 4.x only*) cumulative number of mblks allocated (counter)
mblkfail - (*SunOS 4.x only*) number of mblk allocation failures (counter)
msgused - (*SunOS 5.x only*) used msgs
msgfree - (*SunOS 5.x only*) free msgs
msg% - (*SunOS 5.x only*) percent used msgs
msgcum - (*SunOS 5.x only*) cumulative msgs allocated
msgfail - (*SunOS 5.x only*) msgs allocation failures
lblkused - (*SunOS 5.x only*) used linkblks
lblkfree - (*SunOS 5.x only*) free linkblks
lblk% - (*SunOS 5.x only*) percent used linkblks
lblkcum - (*SunOS 5.x only*) cumulative linkblks allocated
lblkfail - (*SunOS 5.x only*) linkblks allocation failures
streveused - (*SunOS 5.x only*) used strevents
strevefree - (*SunOS 5.x only*) free strevents
streve% - (*SunOS 5.x only*) percent used strevents
strevecum - (*SunOS 5.x only*) cumulative strevents allocated
strevefail - (*SunOS 5.x only*) strevents allocation failures

The **dblks** table reports statistics on the number of allocation requests for data blocks. The key is on the size of the data block, or zero for all data blocks. Under systems prior to SunOS 4.1, valid keys are 0=total, 4, 16, 64, 128, 256, 512, 1024, 2048 and 4096.

size - dblk size in bytes (gauge)

used - number of used dblks (gauge)

free - number of free dblks (gauge)

used% - percentage of used dblks (gauge)

cum - cumulative number of dblks allocated (counter)

fail - number of dblk allocation failures (counter)

OPTIONS

The *SunOS 4.x* version accepts one option: the name of an alternate file to use to get the kernel namelist.

The *SunOS 5.x* version has no options.

ERRORS

can't read kernel memory

The agent uid does not have permission to read kernel memory, **/dev/kmem**. Run the agent as root (default case if agent started by **inetd (1M)**), or give read permission to the uid the agent is running under.

The error message can also occur if the agent can't find or read the file containing the kernel namelist.

NAME

na.hostmem2 – network memory buffer pool usage and statistics (for Solaris 2.x)

DESCRIPTION

na.hostmem reports usage and utilization statistics for the memory buffer pool used by the network routines. These statistics include streams allocation, mbuf usage, etc.

A reporting interval is specified by the management application. If a reporting interval is not specified, the agent sends a report every 30 seconds.

ATTRIBUTES

The *SunOS 5.x* version of **na.hostmem2** has one attribute group, **streams**.

The **streams** group reports information about streams usage.

strused - number of used streams (gauge)
strfree - number of free streams (gauge)
str% - percentage of used streams (gauge)
strcum - cumulative number of streams allocated (counter)
strfail - number of streams allocation failures (counter)
qused - number of used queues (gauge)
qfree - number of free queues (gauge)
q% - percentage of used queues (gauge)
qcum - cumulative number of queues allocated (counter)
qfail - number of queue allocation failures (counter)
msgused - used msgs
msgfree - free msgs
msg% - percent used msgs
msgcum - cumulative msgs allocated
msgfail - msgs allocation failures
lblkused - used linkblks
lblkfree - free linkblks
lblk% - percent used linkblks
lblkcum - cumulative linkblks allocated
lblkfail - linkblks allocation failures
streveused - used strevents
strevefree - free strevents
streve% - percent used strevents
strevecum - cumulative strevents allocated
strevefail - strevents allocation failures

OPTIONS

None.

ERRORS

can’t read kernel memory

The agent uid does not have permission to read kernel memory, **/dev/kmem**. Run the agent as root (default case if agent started by **inetd (1M)**), or give read permission to the uid the agent is running under.

The error message can also occur if the agent can’t find or read the file containing the kernel namelist.

NAME

na.hostperf – host statistics

DESCRIPTION

na.hostperf is a proxy agent that reports statistics obtained from the kernel on the target system, including CPU, paging / swapping, disk and interface statistics. It gets data by querying the **rpc.rstatd (1M)** daemon on the target host through the **rstat** protocol.

A reporting interval is specified by the management application. If a reporting interval is not specified, the proxy agent sends reports every 30 seconds.

ATTRIBUTES

na.hostperf has one attribute group, **data**, that reports host data, i.e., **perfmeter (1)** information.

cpu% - CPU utilization percentage (gauge). The first time the agent reports this value, it reports the average since boot, that is, the total percentage the CPU was active since it was booted. After the first report, each report is the percentage the CPU was active since the previous report.

intr - number of device interrupts since boot (counter)

cswitch - number of context switches since boot (counter)

avenrun_1 - average number of runnable processes over last minute (float)

avenrun_5 - average number of runnable processes over last five minutes (float)

avenrun_15 - average number of runnable processes over last 15 minutes (float)

disk - number of disk transfers since boot (counter)

uptime - number of hundredths of seconds the host has been up (timeticks)

boottime - time the host was booted (unixtime)

curtime - current time on host (unixtime)

cpubusy - number of hundredths of seconds the CPU has been active since boot (counter)

pgin - number of pages read in from disk since boot (counter)

pgout - number of pages written to disk due to page faults since boot (counter)

pswpin - number of pages swapped in since boot (counter)

pswpout - number of pages swapped out since boot (counter)

ipkts - number of input packets on all interfaces since boot (counter)

opkts - number of output packets on all network interfaces since boot (counter)

ierrs - number of input errors on all network interfaces since boot (counter)

oerrs - number of output errors on all network interfaces since boot (counter)

colls - number of collisions on all interfaces since boot (counter)

ocolls% - percentage of output collisions (gauge). The first time the agent reports this value, it reports the average since boot, that is, the total percentage of output collisions since the system was booted. After the first report, each report is the percentage that the

CPU was active since the previous report.

OPTIONS

No options are accepted.

ERRORS

cannot create new subprocess; would exceed maximum: *maximum*

The proxy agent cannot start a new request because creating a new subprocess would exceed the maximum number of proxy agent subprocesses allowed. Either terminate some requests and then resubmit the request or modify the value associated with the "na.hostperf.max-subprocs" keyword in **/etc/opt/SUNWconn/snm/snm.conf** for Solaris 2.x and in **/etc/snm.conf** for Solaris 1.x. If you change the "na.hostperf.max-subprocs" value, you must terminate all **na.hostperf** requests, kill all **na.hostperf** processes, and then restart the hostperf requests.

cannot get performance data

The **rpc.rstatd (1M)** daemon may not be running on the remote host, or the remote host may not be reachable.

NOTES

The way collision count is tallied depends on the Ethernet chip. The Intel (ie) chip counts every occurrence, while the Lance (le) chip counts collision on a per-packet basis. With the Lance chip, if a packet is successfully sent after the first retry, the collision count is one. If a packet is successfully sent after two or more retries, the chip returns a maximum count of two collisions. For example, if packet *a* had four retries and sometime later packet *b* had five retries, the cumulative collision count is four — two for each packet — not the real count of nine.

It is not possible for **na.hostperf** to report the system statistics per interface since the **rpc.rstatd (1M)** daemon does not break them out. Use **na.hostif**(8) to get per-interface statistics.

NAME

na.iostat – Input / Output statistics

DESCRIPTION

na.iostat returns information about the CPU, disk, and TTY activity on the host where the agent is running. By default, the agent reads **/vmunix** to get the kernel namelist.

A reporting interval is specified by the management application. If a reporting interval is not specified, the agent will send a report every 30 seconds.

ATTRIBUTES

na.iostat has one attribute table, **disk**, and two attribute groups, **cpu** and **tty**.

The **disk** table reports statistics about the system's disks. The key is the name. The attributes in the table are:

name - name of the disk (string)

mbytes - (*SunOS 4.x only*) number of megabytes transferred since boot (counter)

xfers - (*SunOS 4.x only*) number of disk transfers since boot (counter)

reads - number of reads since boot (counter)

writes - number of writes since boot (counter)

seeks - (*SunOS 4.x only*) number of seeks since boot (counter)

activeT - (*SunOS 4.x only*) seconds active since boot (counter)

xferT - (*SunOS 4.x only*) seconds spent transferring data since boot (counter)

seekT - (SunOS 4.x only) seconds spent seeking since boot (counter)

%xferT - (*SunOS 4.x only*) percent of active time spent transferring data (gauge)

avgXferT - (*SunOS 4.x only*) average transfer time, in milliseconds (gauge)

avgSeekT - (*SunOS 4.x only*) average seek time, in milliseconds (gauge)

kbps - (*SunOS 4.x only*) transfer rate, in Kbytes per second (gauge)

kreads - (*SunOS 5.x only*) number of Kbytes read per second (float)

kwrites - (*SunOS 5.x only*) number of Kbytes written per second (float)

svcwait - (*SunOS 5.x only*) average number of transactions waiting for service (float)

acttrans - (*SunOS 5.x only*) average number of transactions actively being serviced (float)

svctime - (*SunOS 5.x only*) average service time in milliseconds (float)

%wait - (*SunOS 5.x only*) percent of time there are transactions waiting for service (float)

%busy - (*SunOS 5.x only*) percent of time the disk is busy (float)

The **cpu** group reports statistics about the CPU utilization.

user - number of ticks spent running user jobs since boot (counter)

nice - number of clock ticks spent in nice mode since boot (counter)

sys - number of clock ticks spent in system mode since boot (counter)

idle - number of clock ticks spent idle since boot (counter)

The **tty** group reports statistics about tty input and output.

tty_in - number of characters read since boot (counter)

tty_out - number of characters written since boot (counter)

OPTIONS

The *SunOS 4.x* version accepts one option: the name of an alternate file to use to get the kernel namelist.

The *SunOS 5.x* version has no options.

ERRORS

can't read kernel memory

The agent uid does not have permission to read kernel memory, **/dev/kmem**. Run the agent as root (default case if agent started by **inetd (1M)**), or give read permission to the uid the agent is running under.

The error message can also occur if the agent can't find or read the file containing the kernel namelist.

NOTES

Milliseconds per average seek is an approximation based on the disk (not the controller) transfer rate. Therefore, the seek time will be overestimated in systems with slower controllers.

Not all controllers report all data. In particular, number of seeks is unavailable for SCSI disks.

The SunOS 4.1.x kernel keeps statistics for up to 32 drives. However, the kernel as shipped only reserves space for statistics for 10 drives. To increase the space, change the constant DK_NDRIVE in **/usr/include/sys/dk.h** to any value up to 32. After you make the change, recompile your kernel (you must have SunOS kernel sources) and reboot with the new kernel. If you do not have kernel sources, then you cannot get statistics for more than 10 drives.

NAME

na.iostat2 – Input / Output statistics

DESCRIPTION

na.iostat2 returns information about the CPU, disk, and TTY activity on the host where the agent is running.

A reporting interval is specified by the management application. If a reporting interval is not specified, the agent will send a report every 30 seconds.

ATTRIBUTES

na.iostat2 has one attribute table, **disk**, and two attribute groups, **cpu** and **tty**.

The **disk** table reports statistics about the system's disks. The key is the name. The attributes in the table are:

name - name of the disk (string)

reads - number of reads per second (float)

writes - number of writes per second (float)

kreads - number of Kbytes read per second (float)

kwrites - number of Kbytes written per second (float)

svcwait - average number of transactions waiting for service (float)

acttrans - average number of transactions actively being serviced (float)

svctime - average service time in milliseconds (float)

%wait - percent of time there are transactions waiting for service (float)

%busy - percent of time the disk is busy (float)

The **cpu** group reports statistics about the CPU utilization.

user - number of ticks spent running user jobs since boot (counter)

nice - number of clock ticks spent in nice mode since boot (counter)

sys - number of clock ticks spent in system mode since boot (counter)

idle - number of clock ticks spent idle since boot (counter)

The **tty** group reports statistics about tty input and output.

tty_in - number of characters read since boot (counter)

tty_out - number of characters written since boot (counter)

OPTIONS

None.

ERRORS

can't read kernel memory

The agent uid does not have permission to read kernel memory, **/dev/kmem**. Run the agent as root (default case if agent started by **inetd (1M)**), or give read permission to the uid the agent is running under.

NOTES

Milliseconds per average seek is an approximation based on the disk (not the controller) transfer rate. Therefore, the seek time will be overestimated in systems with slower controllers.

NAME

na.ippath — print the path that packets take to a network host

DESCRIPTION

The **na.ippath** proxy agent attempts to trace an IP packet's path between the proxy system and the destination host.

The agent initially launches three 38-byte UDP probe packets with ttl (time to live) of 1, listens for an ICMP ''time exceeded'' reply from the nearest gateway, and then measures the round trip time of each probe / reply.

It then increments the ttl and sends three more probes, collecting time exceeded replies and measuring the round trip time. The agent continues to increment the ttl and send out three probes at a time until it receives an ICMP ''port unreachable'' reply, which means the probe reached the destination host, or until the maximum number of hops is reached. The destination port is set to an unlikely value, so that the destination host does not process the UDP probe packets.

The **na.ippath** agent returns a table. Each row in the table shows the ttl (as ''hop''), symbolic and numeric address of the gateway, and average round trip time of the probes. If the probe answers come from different gateways, only the first responding system is printed, but the round trip times are still measured to compute the average.

Each probe will wait up to 3 seconds for a reply. If none of the three probes are answered, the agent prints ''???'' for the system name and address.

This proxy agent should be used primarily for manual fault isolation. Because of the load it could impose on the network, you should use **na.ippath** via one-time (''Quick Dump'') operations only. It is unwise to use **na.ippath** from automated scripts or from ongoing data / event reports. If the proxy agent is instructed to report more than once and you do not specify a reporting interval, it will report every 10 minutes.

ATTRIBUTES

na.ippath has one attribute table, **path .** The attributes in the table are:

hop - number of hops away from proxy system (int)

system - name of the system that is **hop** hops away (string)

address - IP address of the system that is **hop** hops away (string)

avgtrip - average trip time (in milliseconds) from the proxy system to the system that is **hop** hops away (gauge)

note - an optional field that describes potential problems with the system that is **hop** hops away (string)

OPTIONS

The proxy agent operates with some default rules that can be overridden with options. Options accepted are:

–**m** Set the maximum time-to-live (maximum number of hops) used in outgoing probe packets. The default is 30 hops (the same default used for TCP connections).

–**n** Print **system** numerically rather than symbolically. This saves a name server address-to-name lookup for each gateway found on the path.

−p Set the base UDP port number used in probes (default is 33434). **na.ippath** operates under the assumption that nothing is listening on UDP ports *base* to *base+nhops-1* at the destination host. (An ICMP PORT_UNREACHABLE message will be returned to terminate the path tracing.) If something is listening on a port in the default range, this option can be used to pick an unused port range.

−r Bypass the normal routing tables and send directly to a host on an attached network. If the host is not on a directly-attached network, an error is returned. This option can be used to ping a local host through an interface that has no route through it, for example, after the interface was dropped by **route (1M) .**

−s Use the following IP address, which must be given as an IP number, not a host name, as the source address in outgoing probe packets. On hosts with more than one IP address, you can use this option to force the source address to be something other than the IP address of the interface the probe packet is sent on. If the IP address is not one of the proxy system's interface addresses, an error is returned and nothing is sent.

−t Set the *type-of-service* (TOS) in probe packets to the following value (default zero). The value must be a decimal integer in the range 0 to 255. You can use this option to see if different types-of-service result in different paths. (This may be academic since the normal network services like TELNET and **ftp** do not let you control the TOS.) Not all values of TOS are legal or meaningful\msee the IP spec for definitions. Useful values are probably '-t 16' (low delay) and '-t 8' (high throughput).

−w Set the time (in seconds) to wait for a response to a probe (default is three seconds).

OUTPUT

A sample output might be:

Sat Oct 27 21:37:48 1990 [nis.nsf.net] : Quick Dump: ippath.path

hop	system	address	avgtrip
1	helios.ee.lbl.gov	128.3.112.1	12
2	lilac-dmc.Berkeley.EDU	128.32.216.1	26
3	lilac-dmc.Berkeley.EDU	128.32.216.1	26
4	ccngw-ner-cc.Berkeley.EDU	128.32.136.23	39
5	ccn-nerif22.Berkeley.EDU	128.32.168.22	39
6	128.32.197.4	128.32.197.4	52
7	131.119.2.5	131.119.2.5	59
8	129.140.70.13	129.140.70.13	92
9	129.140.71.6	129.140.71.6	232
10	129.140.81.7	129.140.81.7	206
11	nis.nsf.net	35.1.1.48	239

In the output above, hops 2 & 3 are the same, due to a buggy kernel on the second hop system, lbl-csam.arpa. It forwards packets with a zero ttl (as a result of a bug in the distributed version of 4.3BSD). Note that you have to guess what path the packets are taking

cross-country since the NSFNet (129.140) generally doesn't supply address-to-name translations for its NSSes.

A more interesting example is:

```
Sat Oct 27 21:51:01 1990 [ bluejay ] : Quick Dump: ippath.path

hop  system     address              avgtrip

  1  commserv   129.144.44.2          3
  2  snu-ebb    129.144.1.43         6
  3  ???        ???
  4  bluejay    129.144.152.31       10
```

Note that the gateway three hops away either doesn't send ICMP ''time exceeded'' messages or sends them with a ttl too small to reach you.

The silent gateway 3 in the above may be the result of a bug in the 4.[23]BSD network code (and its derivatives): 4.x (x <= 3) sends an unreachable message using whatever ttl remains in the original datagram. Since, for gateways, the remaining ttl is zero, the ICMP ''time exceeded'' is guaranteed to not make it back to you. The behavior of this bug is slightly more interesting when it appears on the destination system, as follows:

```
Sat Oct 27 21:54:20 1990 [ tweedle ] : Quick Dump: ippath.path

hop  system     address              avgtrip  note

  1  commserv   129.144.44.2          3
  2  nico-ebb   129.144.1.91         10
  3  olala-bb   129.144.156.88       10
  4  ???        ???
  5  ???        ???
  6  ???        ???
  7  tweedle    129.144.145.5        3        TTL <= 1
```

In this output there are six ''gateways'' (7 is the final destination). Exactly the last half of them are ''missing.'' Actually, tweedle is using the ttl from the arriving datagram as the ttl in its ICMP reply. The reply will time out on the return path—with no notice sent to anyone since ICMPs aren't sent for ICMPs—until probed with a ttl at least twice the path length. So tweedle is really only four hops away. A reply that returns with a ttl of 1 is a clue that this problem exists. The agent prints a note after the time if the ttl is <= 1. Since there is still a lot of obsolete and non-standard software in the world, expect to see this problem frequently and / or take care picking the target host of your probes.

Other possible annotations after the time include: **Host Unreachable , Net Unreachable , Protocol Unreachable** (got a host, network, or protocol unreachable, respectively), **Source Route Failed ,** or **Frag Needed** (source route failed or fragmentation needed). Neither of the last two annotations should ever occur; the associated gateway is busted if you see one. If almost all the probes result in some kind of unreachable annotation, the agent will give up and exit.

NOTES

The agent collects all the path information before reporting; thus, the report may take a few seconds to show up. The amount of time it takes depends on the number of hops in the path and the amount of time the agent has to wait for each probe response.

The **na.ippath** proxy agent was originally implemented as the command line program **traceroute** by Van Jacobson from a suggestion by Steve Deering. It was debugged by a cast of thousands, with particularly cogent suggestions or fixes from C. Philip Wood, Tim Seaver and Ken Adelman. The conversion to a SunNet Manager agent was done at Sun Microsystems.

SEE ALSO

netstat(1M), **ping**(1M)

NAME

na.iproutes – routing table statistics

DESCRIPTION

na.iproutes reports routing table statistics. By default the agent reads **/vmunix** to get the kernel namelist.

A reporting interval is specified by the management application. If a reporting interval is not specified, the agent will send reports every 30 seconds.

ATTRIBUTES

na.iproutes has one attribute group, **stats**, and one attribute table, **routes**.

The **stats** group reports routing statistics. (The **stats** group supported by OS 4.x only.)

bad - number of bad routing redirects (counter)

dynamic - number of dynamically created routes (counter)

new_gw - number of routes modified due to redirects (counter)

dest_un - number of destinations found unreachable (counter)

wild - number of uses of a wildcard route (counter)

The **routes** table accepts a destination network name, *destname*, as a key and returns the routing table entry for that destination. If no key is specified, the entire routing table is returned. This table may be quite long for machines in large networks.

destname - destination name (string[32])

destaddr - destination address (inetaddr)

gwname - gateway name (string[32])

gwaddr - gateway address (inetaddr)

flags - status of route. (string[10])

U – Up
G – route is to a gateway
H – route is to a host
D – route was dynamically created due to a redirect

uses - number of active uses (gauge)

opkts - number of output packets sent (counter)

ifname - interface name (string[5])

metric - pseudo-cost or hop count routing metric (gauge)

OPTIONS

The *SunOS 4.x* rersion accepts one option: the name of an alternate file to use to get the kernel namelist.

The *SunOS 5.x* version accepts no options.

ERRORS

can't read kernel memory

The agent uid does not have permission to read kernel memory, **/dev/kmem**. Run the agent as root (default case if agent started by **inetd (1M)**), or give read

permission to the uid the agent is running under.

The error message can also occur if the agent can't find or read the file containing the kernel namelist.

NAME

na.layers – network layers statistics **for SunOS 4.1.x**

DESCRIPTION

na.layers returns information about the various network layers on the host where the agent is running. By default, the agent reads **/vmunix** to get the kernel name list.

This agent only runs on 4.x systems. A new agent, **na.layers2**(8)**,** runs on SunOS 5.x systems.

A reporting interval is specified by the management application. If a reporting interval is not specified, the agent will send a report every 30 seconds.

ATTRIBUTES

na.layers has four attribute groups: **udp, ip, icmp** and **tcp**.

The **udp** group reports statistics about the UDP protocol. (See RFC768 for information about UDP.) The group includes:

bad_hdr - number of incomplete headers (counter)

bad_len - number of bad data length fields (counter)

bad_cksum - number of bad checksums (counter)

sock_oflow - number of socket overflows (counter)

The **ip** group reports statistics about the IP protocol. (See RFC791 for information about IP.) The group includes:

in_pkts - number of packets received (counter)

bad_hdr_cksum - number of bad header checksums (counter)

size_min - number of less than minimum size packets (counter)

bad_size - number of packets with less data than shown in the data length field (counter)

hdr_len - number of packets with improper header length (counter)

data_len - number of packets with data length less than header length (counter)

frags - number of fragments received (counter)

frags_drop - number of fragments dropped due to duplication or being out of space (counter)

drop_time - number of fragments dropped after timeout (counter)

forward - number of packets forwarded (counter)

bad_forward - number of packets not forwardable (counter)

redirects - number of redirects sent (counter)

The **icmp** group reports statistics about the ICMP protocol. (See RFC792 for information about ICMP.) The group includes:

icmp_error - number of calls to **icmp_error** (counter)

err_short - number of errors not generated because old message was too short (counter)

err_icmp - number of errors not generated because old message was ICMP (counter)

bad_code - number of times **icmp_code** out of range (counter)

icmp_min_len - number of less than minimum length messages (counter)

icmp_cksum - number of bad checksums (counter)

icmp_bad_len - number of calculated bound mismatches (counter)

out_dst_unreachable - number of ICMP destination unreachables sent (counter)

out_time_exceeded - number of ICMP time exceeded sent (counter)

out_parameter - number of ICMP parameter problems sent (counter)

out_src_quench - number of ICMP source quenches sent (counter)

out_redirect - number of ICMP redirects sent (counter)

out_echo - number of ICMP echo requests sent (counter)

out_echo_reply - number of ICMP echo replies sent (counter)

out_time_stamp - number of ICMP timestamp requests sent (counter)

out_time_stamp_reply - number of ICMP timestamp replies sent (counter)

out_addr_mask_rqst - number of ICMP address mask requests sent (counter)

out_addr_mask_rqst_reply - number of ICMP address mask replies sent (counter)

out_info_rqst - number of ICMP information requests sent (counter)

out_info_rqst_reply - number of ICMP information request replies sent (counter)

in_dst_unreachable - number of ICMP destination unreachables received (counter)

in_time_exceeded - number of ICMP time exceededs received (counter)

in_parameter - number of ICMP parameter problems received (counter)

in_src_quench - number of ICMP source quenches received (counter)

in_redirect - number of ICMP redirects received (counter)

in_echo - number of ICMP echo requests received (counter)

in_echo_reply - number of ICMP echo replies received (counter)

in_time_stamp - number of ICMP timestamp requests received (counter)

in_time_stamp_reply - number of ICMP timestamp replies received (counter)

in_add_mask_rqst - number of ICMP address mask requests received (counter)

in_add_mask_rqst_rply - number of ICMP address mask replies received (counter)

in_info_rqst - number of ICMP information requests received (counter)

in_info_rqst_reply - number of ICMP information request replies received (counter)

out_unknown_type_1 - number of times unknown code #1 in outgoing type field (counter)

out_unknown_type_2 - number of times unknown code #2 in outgoing type field (counter)

out_unknown_type_6 - number of times unknown code #6 in outgoing type field (counter)

out_unknown_type_7 - number of times unknown code #7 in outgoing type field (counter)

out_unknown_type_9 - number of times unknown code #9 in outgoing type field (counter)

out_unknown_type_10 - number of times unknown code #10 in outgoing type field (counter)

in_unknown_type_1 - number of times unknown code #1 in incoming type field (counter)

in_unknown_type_2 - number of times unknown code #2 in incoming type field (counter)

in_unknown_type_6 - number of times unknown code #6 in incoming type field (counter)

in_unknown_type_7 - number of times unknown code #7 in incoming type field (counter)

in_unknown_type_9 - number of times unknown code #9 in incoming type field (counter)

in_unknown_type_10 - number of times unknown code #10 in incoming type field (counter)

The **tcp** group reports statistics about the TCP protocol. (See RFC793 for information about TCP.) The group includes:

pkts_sent - number of packets sent (counter)

snd_data_pkts - number of data packets sent (counter)

snd_data_bytes - number of data bytes sent (counter)

snd_retransmit_pkts - number of data packets retransmitted (counter)

snd_retransmit_bytes - number of data bytes retransmitted (counter)

snd_ack_only - number of acknowledgement–only packets sent (counter)

snd_delayed_ack - number of delayed acknowledgements sent (counter)

snd_URG_only_pkts - number of packets sent with the URGENT flag only (counter)

snd_window_probe_pkts - number of window probes sent (counter)

snd_window_update_pkts - number of window update only packets sent (counter)

snd_control_pkts - number of control — SYN, FIN or RST — packets sent (counter)

rcv_pkts - number of packets received (counter)

rcv_ack - number of acknowledgement packets received (counter)

rcv_ack_bytes - number of bytes acknowledged by received acknowledgement packets (counter)

duplicate_acks - number of duplicate acknowledgements received (counter)

acks_unsent_data - number of acknowledgements received for unsent data (counter)

rcv_pkts_in_seq - number of packets received in sequence (counter)

rcv_bytes_in_seq - number of bytes received in sequence (counter)

rcv_dub_pkts - number of completely duplicate packets received (counter)

rcv_dup_bytes - number of completely duplicate bytes received (counter)

rcv_part_dup_pkts - number of packets with some duplicate data (counter)

rcv_part_dup_bytes - number of duplicate bytes received in packets with some duplicate data (counter)

rcv_out_order_pkts - number of packets received outfrder (counter)

rcv_out_order_bytes - number of bytes received in out of order packets (counter)

rcv_pkts_after_win - number of packets received with data after window (counter)

rcv_bytes_after_win - number of bytes received after window (counter)

rcv_window_probes - number of window probe packets received (counter)

rcv_window_update_pkts - number of window update packets received (counter)

rcv_pkts_after_close - number of packets received after closing connection (counter)

rcv_disc_for_bad_cksums - number of packets received with checksum errors (counter)

rcv_disc_bad_hdr_offset - number of packets received with bad header offset (counter)

rcv_disc_pkts_too_short - number of too short packets received (counter)

conns_requests - number of connections initiated (counter)

conns_accepts - number of connections accepted (counter)

conns_established - number of connections established (counter)

conns_closed - number of connections closed - including drops (counter)

conns_dropped - number of connections dropped (counter)

embryo_conns_dropped - number of embryonic connections dropped (counter)

segments_updated_rtt - number of times successful at getting round trip time (counter)

updated_rtt_attempts - number of times attempted to get round trip time (counter)

retransmit_timeouts - number of retransmission timeouts (counter)

conns_drop_by_rexmit_timeout - number of connections dropped due to retransmission timeout (counter)

persist_timeouts - number of persist timeouts (counter)

keepalive_timeouts - number of keep alive timeouts (counter)

keepalive_probes_sent - number of keep alive probes sent (counter)

conns_drop_by_keepalive - number of connections dropped in keep alive (counter)

OPTIONS

One option is accepted: the name of an alternate file to use to get the kernel name list.

ERRORS

can't read kernel memory

The agent uid does not have permission to read kernel memory, **/dev/kmem**. Run the agent as root—the default when **inetd (1M)** starts the agent—or give read permission to the uid the agent is running under.

The error message can also occur if the agent can't find or read the file containing the kernel name list.

NAME

na.layers2 – network layers statistics for **Solaris 2.x**

DESCRIPTION

na.layers2 returns information about the various network layers on the host where the agent is running.

A reporting interval is specified by the management application. If a reporting interval is not specified, the agent will send a report every 30 seconds.

ATTRIBUTES

na.layers2 has four attribute groups: **udp, ip, icmp**, and **tcp**.

The **udp** group reports statistics about the UDP protocol. (See RFC768 for information about UDP.) The group includes:

udp_indatagrams - number of datagrams received (counter)

udp_outdatagrams - number of datagrams sent (counter)

udp_inerrors - number of errors on input datagrams (counter)

The **ip** group reports statistics about the IP protocol. (See RFC791 for information about IP.) The group includes:

ip_forwarding - determines if the node is a gateway: 1=gateway, 2= NOT gateway (int)

ip_default_ttl - default time-to-live for iph (int)

ip_in_receives - number of input datagrams (counter)

ip_in_hdr_errors - number of discards for header errors (counter)

ip_in_addr_errors - number of discards for bad address (counter)

ip_forw_datagrams - number of datagrams being forwarded (counter)

ip_unknown_protos - number of discards for unknown protocol (counter)

ip_in_discards - number of discards of good datagrams (counter)

ip_in_delivers - number of datagrams sent upstream (counter)

ip_out_requests - number of datagrams received from upstream (counter)

ip_out_discards - number of good outgoing datagrams discarded (counter)

ip_out_no_routes - number of outgoing discards because no route found (counter)

ip_reasm_timout - number of seconds received fragments are held for reassembly (int)

ip_reasm_reqds - number of IP fragments needing reassembly (counter)

ip_frags_ok - number of datagrams fragmented (counter)

ip_frag_fails - number of datagrams discarded for no fragment set (counter)

ip_frag_creates - number of datagram fragments from fragmentation (counter)

ip_addr_entry - (int)

ip_route_entry_size - (int)

ip_net_to_media_size - (int)

ip_routing_discards - (int)

tcp_in_errs - number of segments received with errors (counter)

ip_routing_discards - number of received datagrams not delivered (counter)

The **icmp** group reports statistics about the ICMP protocol. (See RFC792 for information about ICMP.) The group includes:

icmp_in_msgs - number of ICMP messages received (counter)

icmp_in_errors - number of ICMP errors on receive (counter)

icmp_in_dest_unreachs - number of received ''dest unreachable'' messages (counter)

icmp_in_time_excds - number of received ''time exceeded'' messages (counter)

icmp_in_param_probs - number of received ''parameter problem'' messages (counter)

icmp_in_src_quenchs - number of received ''src quench'' messages (counter)

icmp_in_redirects - number of received ''icmp redirect'' messages (counter)

icmp_in_echos - number of received ''echo request'' messages (counter)

icmp_in_echo_reps - number of received ''echo reply'' messages (counter)

icmp_in_timestamps - number of received ''timestamp'' messages (counter)

icmp_in_timestamp_reps - number of received ''timestamp reply'' messages (counter)

icmp_in_addr_masks - number of received ''addr mask request'' messages (counter)

icmp_in_addr_mask_reps - number of received ''addr mask reply'' messages (counter)

icmp_out_msgs- total number of ICMP messages sent (counter)

icmp_out_errors - number of messages not sent because of internal ICMP errors (counter)

icmp_out_dest_unreachs - number of ''dest unreachable'' messages sent (counter)

icmp_out_time_excds - number of ''timeout exceeded'' messages sent (counter)

icmp_out_param_probs - number of ''parameter problems'' messages sent (counter)

icmp_out_src_quench - number of ''src quench'' messages sent (counter)

icmp_out_redirects - number of ''ICMP redirect'' messages sent (counter)

icmp_out_echos - number of ''echo request'' messages sent (counter)

icmp_out_echo_reps - number of ''echo reply'' messages sent (counter)

icmp_out_timestamps - number of ''timestamp request'' messages sent (counter)

icmp_out_timestamp_reps - number of ''timestamp reply'' messages sent (counter)

icmp_out_addr_mask - number of ''addr mask request'' messages sent (counter)

icmp_out_addr_mask_reps - number of ''addr mask reply'' messages sent (counter)

The **tcp** group reports statistics about the TCP protocol. (See RFC793 for information about TCP.) The group includes:

tcp_r_to_algorithm - algorithm used for transmit timeout value (int)

tcp_r_to_min - minimum retransmit timeout (ms) (int)

tcp_r_to_max - maximum retransmit timeout (ms) (int)

tcp_max_conn - maximum number of connections supported (int)

tcp_active_opens - number of direct transitions CLOSED -> SYN-SENT (counter)

tcp_passive_opens - number of direct transitions LISTEN -> SYN_RCVD (counter)

tcp_attempt_fails - number of direct SIN-SENT / RCVD -> CLOSED / LISTEN (counter)

tcp_estab_resets - number of direct ESTABLISHED / CLOSED-WAIT -> CLOSED (counter)

tcp_curr_estab - number of connections ESTABLISHED or CLOSE WAIT (counter)

tcp_in_segs - number of of segments received (counter)

tcp_out_segs - number of segments sent (counter)

tcp_retrans_segs - number of segments retransmitted (counter)

tcp_conn_table_size - table size (int)

tcp_out_rsts - segments sent with RST flag (counter)

OPTIONS

na.layers2 has no options:

ERRORS

can't read kernel memory

The agent uid does not have permission to read kernel statistics, **/dev/kstat.** Run the agent as root—the default when **inetd (1M)** starts the agent—or give read permission to the uid the agent is running under.

NAME

na.logger– management logger

DESCRIPTION

na.logger receives data, event, and error reports from agents and records them in the log file specified by the *monitor-log* keyword listed in the local **snm.conf**(5)**.**

You can start the logger from the command line or from **inetd (1M) .**

OPTIONS

Starting **na.logger** with the **-d1** option displays debugging information.

ERRORS

na.logger sends an error message to **syslogd (1M)** if it cannot open its log file for appending.

SEE ALSO

snm.logfile(5)

NAME

na.lpstat — print the line printer status and information

DESCRIPTION

This proxy agent, originated from the **lpq** program, allows you to examine the printer status and the print scheduler to examine the print job queue.

If the spooling directory is not defined in the **/etc/printcap** file, the default directory is used.

na.lpstat has to be run as root.

ATTRIBUTES

na.lpstat has one attribute group, **status**, and one attribute table, **queue**.

The **status** group reports printer status. The group includes:

status - printer status text string (string[32])

statusCode - indicates printer status: 0 - idle, 1 - waiting, 2 - printing, 3 - printer error. (int)

daemonPresent - (**for SunOS 4.x systems only**) indicates whether the **lp** daemon is present. The value ‘‘true’’ means that the daemon is running, while ‘‘false’’ means the daemon is not running. (enum)

printerQueue - (**for SunOS 4.x systems only**) indicates whether the printer queue is turned on. The value ‘‘on’’ means the queue is turned on. The value‘‘off’’ means the queue is turned off. (enum) **lpScheduler** (**for SunOS 5.x systems only**) indicates whether the lpScheduler program is available. True indicates that the printer is available; False indicates a printer is not available.

The **queue** table reports the job queue of printer. The key is the owner name of the job.

ownerName - owner name of the print job (string[32])

rank - rank of the print job (string[32])

jobNumber - job number of the print job (int)

totalBytes - total number of bytes of the file to be printed (long)

OPTIONS

No options accepted.

ERRORS

cannot open printer description file
cannot open the description file (that is, **/etc/printcap)** for the printer.

unknown printer
printer name is unknown to the proxy agent.

connection is down
connection to the remote printer is down.

lost connection
connection to the remote printer is lost.

cannot chdir to spooling directory
cannot change current directory to spooling directory. The directory might not

exist or give the user access permission.

cannot examine spooling area
cannot examine the spooling directory; this might be related to previous error.

no remote host to connect to
no remote host name defined in the printer description file, or the name is unknown to the proxy agent.

unknown host
remote host name is unknown to the proxy agent.

printer/tcp: unknown service
service unknown to proxy agent.

cannot report status group
proxy agent is unable to report group 'status' for building report error.

cannot report queue table
proxy agent is unable to report table 'queue' for building report error.

cannot open lock file(permission denied)
unable to open the printer lock file, possibly no root permission.

printer is down
printer is down.

no daemon present
warning message, no **lp** daemon running.

NOTES

The proxy agent is dependent upon the **lp** daemon. It examines the status and controls files in the spooling directory that the daemon creates.

The 'statusCode' attribute in the 'status' group is provided for ease in setting up the thresholds for the printer status. The attribute 'status' is an ASCII string. Note that there is no distinction between what kind of printer error may have occurred (for example, out of paper, paper jammed); all errors have a statusCode of 3.

Like the **lpq** program, the proxy depends on the **printcap** file.

NAME

na.ping — machine reachability information

DESCRIPTION

na.ping is a proxy agent that reports machine reachability information. The information is gathered by sending ICMP ECHO request packets to the target machine from the specified proxy and getting the round-trip time of replies.

A reporting interval is specified by the management application. If a reporting interval is not specified, the proxy agent will report every 30 seconds.

ATTRIBUTES

na.ping has two attribute groups, **reach** and **stats**.

The **reach** group reports host reachability information and packet round trip time. The group includes:

hostname - name of the target system (string[32])

size - datagram packet size, in bytes, sent to target system (int). If the datagram passes through a gateway, the packet might be split into fragments. (Fragment size is the maximum packet size (MTU) any given gateway can handle.)

triptime - round trip time, in milliseconds, to send a packet from the proxy agent system to the target system and back (gauge). If the host is unreachable, this value is set to the number of milliseconds before the request timed out.

reachable - an enumerated type indicating whether the host is reachable using ICMP ECHO. The value ''true'' means that the host is reachable while ''false'' indicates that the host is unreachable.

The **stats** group reports statistics for packets sent to a particular machine. These statistics include the average round-trip time, maximum round-trip time, percent of lost packets, and so on. The group includes:

hostname - name of the target system (string[32])

size - datagram packet size, in bytes, sent to target system (int). If the datagram passes through a gateway, the packet might be split into fragments. (Fragment size is the maximum packet size (MTU) that any given gateway can handle.)

sent - number of packets sent (gauge)

dropped - number of packets dropped (gauge)

dups - number of duplicate ECHO REPLY messages received (gauge)

pktloss - packet loss percentage (gauge)

tripmin - minimum round trip time, in milliseconds (gauge). If the host is unreachable, this value is set to the number of milliseconds before the request timed out.

tripmax - maximum round trip time, in milliseconds (gauge). If the host is unreachable, this value is set to the number of milliseconds before the request timed out.

tripavg - average round trip time, in milliseconds (gauge). If the host is unreachable, this value is set to the number of milliseconds before the request timed out.

reachable - an enumerated type indicating whether the host is reachable using ICMP ECHO. The value ''true'' means that the host is reachable, while ''false'' indicates that the host is unreachable.

ERRORS

cannot create new subprocess; would exceed maximum: *maximum*
The proxy agent cannot start a new request because creating a new subprocess would exceed the maximum number of proxy agent subprocesses allowed. Either terminate some requests and then resubmit the request or modify the value associated with the ''na.ping.max-subprocs'' keyword in **/etc/opt/SUNWconn/snm/snm.conf** for Solaris 2.x and in **/etc/snm.conf** for Solaris 1.x. If you change the ''na.ping.max-subprocs'' value, you must terminate all **na.ping** requests, kill all **na.ping** processes, and then restart the ping requests.

send(3n) call failed
The target system may not be reachable due to routing or other network problems. See **send**(3n).

bad option string specified
The option string was not formatted correctly.

socket(3n) call failed
This message could be due to permission problems, system resource limits reached, or a number of other reasons. See **socket**(3n).

NAME

na.rpcnfs – RPC and NFS statistics

DESCRIPTION

na.rpcnfs reports NFS and RPC statistics about a host. By default, the agent reads **/vmunix** to get the namelist for the statistics.

If the reporting interval is not required, the agent will report every 30 seconds.

ATTRIBUTES

The **na.rpcnfs** agent has two attribute groups, **server** and **client**.

The **server** group reports statistics about an NFS server.

rpc_calls - number of RPC calls (counter)

rpc_bad - number of RPC bad calls (counter)

rpc_null - number of RPC null receives (counter)

rpc_short - number of RPC short packets (counter)

rpc_badhdr - number of RPC bad headers (counter)

nfs_calls - number of NFS calls (counter)

nfs_bad - number of NFS bad calls (counter)

nfs_null - number of NFS calls to null (counter)

nfs_getattr - number of NFS calls to **getattr** (counter)

nfs_setattr - number of NFS calls to **setattr** (counter)

nfs_root - number of NFS calls to root (counter)

nfs_lookup - number of NFS calls to **lookup** (counter)

nfs_readlink - number of NFS calls to **readlink** (counter)

nfs_read - number of NFS calls to **read** (counter)

nfs_wrcache - number of NFS calls to **wrcache** (counter)

nfs_write - number of NFS calls to **write** (counter)

nfs_create - number of NFS calls to **create** (counter)

nfs_remove - number of NFS calls to **remove** (counter)

nfs_rename - number of NFS calls to **rename** (counter)

nfs_link - number of NFS calls to **link** (counter)

nfs_symlink - number of NFS calls to **symlink** (counter)

nfs_mkdir - number of NFS calls to **mkdir** (counter)

nfs_rmdir - number of NFS calls to **rmdir** (counter)

nfs_readdir - number of NFS calls to **readdir** (counter)

nfs_fsstat - number of NFS calls to **nfs_fsstat** (counter)

The **client** group reports statistics about an NFS client.

rpc_calls - number of RPC calls (counter)

rpc_bad - number of RPC bad calls (counter)

rpc_rexmt - number of RPC call retransmits (counter)

rpc_xid - number of RPC bad transaction IDs (counter)

rpc_timeout - number of RPC call timeouts (counter)

rpc_wait - number of RPC waits on busy client handles (counter)

rpc_auth - number of RPC authentication information refreshes (counter)

nfs_calls - number of NFS calls (counter)

nfs_bad - number of NFS bad calls (counter)

nfs_get - number of NFS client handle requests received (counter)

nfs_sleep - number of NFS sleeps for handle (counter)

nfs_null - number of NFS calls to null (counter)

nfs_getattr - number of NFS calls to **getattr** (counter)

nfs_setattr - number of NFS calls to **setattr** (counter)

nfs_root - number of NFS calls to root (counter)

nfs_lookup - number of NFS calls to **lookup** (counter)

nfs_readlink - number of NFS calls to **readlink** (counter)

nfs_read - number of NFS calls to **read** (counter)

nfs_wrcache - number of NFS calls to **wrcache** (counter)

nfs_write - number of NFS calls to **write** (counter)

nfs_create - number of NFS calls to **create** (counter)

nfs_remove - number of NFS calls to **remove** (counter)

nfs_rename - number of NFS calls to **rename** (counter)

nfs_link - number of NFS calls to **link** (counter)

nfs_symlink - number of NFS calls to **symlink** (counter)

nfs_mkdir - number of NFS calls to **mkdir** (counter)

nfs_rmdir - number of NFS calls to **rmdir** (counter)

nfs_readdir - number of NFS calls to **readdir** (counter)

nfs_fsstat - number of NFS calls to **fsstat** (counter)

OPTIONS

The *SunOS 4.x* version accepts one option, a single string specifying the alternate kernel filename to read to obtain the kernel namelist.

The *SunOS 5.x* version has no options.

ERRORS

kernel not configured with server nfs and rpc code

kernel not configured with client nfs and rpc code

The kernel must be reconfigured to contain kernel code to support server or client RPC or NFS.

can’t read kernel memory

The agent uid does not have permission to read kernel memory, **/dev/kmem**. Run the agent as root (the default case if agent was started by **inetd (1M)**) or give read permission to the uid the agent is running under.

This message can also occur if the agent can’t find or read the file containing the kernel namelist.

NAME

na.snmp-trap – SNMP trap daemon

SYNOPSIS

na.snmp-trap -p *UDP_port_number*

DESCRIPTION

na.snmp-trap receives SNMP traps, translates them to SunNet Manager traps, and forwards them to the event dispatcher on one or more management stations. **na.snmp-trap** is usually started from **inetd** (1M).

OPTIONS

na.snmp-trap uses the following entries in **snm.conf**(5)**:**

na.snmp.hostfile
: is the name of the SNMP host file specifying system-specific schema and trap files.

na.snmp.default-schema
: is the name of the default SNMP schema file. This file is used if you do not specify a system-specific schema file.

na.snmp-trap.default-trapfile
: is the name of the default SNMP trap file. This file is used if you do not specify a system-specific trap file.

na.snmp-trap.rendez
: are the names of one or more manager systems that should receive the traps. Separate multiple host names with a colon (:). If this entry is not found, traps will be forwarded to the local machine.

na.snmp-trap.raw
: specifies whether the SNMP trap daemon returns raw OIDs and their values without translation for attributes in trap reports, in addition to the translated attribute names and values.

DESCRIPTION

When the SNMP trap daemon receives a trap, it attempts to match the IP address and the community string name in the trap with the system (or proxy) address and read community string specified by the entries in the SNMP host file. If a match is found, the system name, schema, and trap file specified by that entry are used. If no match is found, the default schema and trap file are used and the system name is set by looking up the name of the IP address in the trap. If this fails, the system name is set to the IP address in dot format.

When the trap daemon starts up, it loads the schema files specified by the keywords ‘‘na.snmp.schemas’’ and ‘‘na.snmp.default-schema’’ in the **/etc/opt/SUNWconn/snm/snm.conf** file for Solaris 2.x and in the **/etc/snm.conf** file for Solaris 1.x. The SNMP trap daemon uses the schema file to determine the name of attributes in the trap’s var-bind list and to map enumerated types to a string. If this fails, attribute names are the object identifier in dot format, and enumerated types are returned as numeric strings. If any changes are made to the schema file, you need to restart the trap daemon.

If a trap file has been specified, the SNMP trap daemon will use it to decide whether the trap should be forwarded or not. It is also used to generate the name of the trap. If no trap file has been specified or the enterprise trap entry cannot be found in the trap file, the trap is forwarded. If it is an enterprise-specific trap, its name will be ‘‘enterprise X,’’ where X is the enterprise-specific trap type number.

OPTIONS

–**p** *UDP_port_number*
UDP port number to use for na.snmp-trap daemon. Default is 162.

SEE ALSO

na.snmp.trapfile(5), **na.snmp.hostfile**(5), **snm.conf**(5)

NOTES

na.snmp-trap must not be running when **inetd** restarts because **inetd** will respond, ‘‘address already in use.’’

NAME

na.snmp – Simple Network Management Protocol proxy agent

SYNOPSIS

na.snmp -p *UDP_port_number*

DESCRIPTION

The SNMP proxy agent allows a SunNet Manager user to manage SNMP devices. **na.snmp** can be used with the following SNMP schemas:

- Sun-supplied schema file. (See ''FILES'' section.)
- Schema file supplied by a vendor or other source.
- Schema file that you create from an enterprise MIB. (See **mib2schema**(1) for more information.)

If no reporting interval is specified for requests, the proxy agent will report every 30 seconds.

OPTIONS

–**p** *UDP_port_number*

UDP port number to use for na.snmp agent. Default is 161.

Management applications can specify a range of SNMP-related options when making a request to the SNMP proxy agent. For example, the schema name, read and write community strings, and request timeout can be sent with the request.

For requests that are sent from the SNM Console, the read and write community strings and request timeout are specified in the Properties window of the target device. In addition, the Options fields in the Data or Event Request Properties window and in the Set Tool window allow you to specify a community string for the request. The schema you choose for the request is also sent to the SNMP proxy as an optional argument.

For other management applications, optional request arguments are sent to the SNMP proxy with the **netmgt_set_argument**(3N) function. The SNMP proxy agent accepts the following request argument names:

na.snmp-read-community
na.snmp-write-community
na.snmp-schema
na.snmp-proxy-device
na.snmp-get-list
na.snmp-get-next-list
na.snmp-timeout

All these request arguments except for **na.snmp-timeout** are NETMGT_STRING argument type codes. **na.snmp-timeout** is a NETMGT_LONG argument type code. See the **netmgt_set_argument**(3N) man page for a description of how to set these arguments. For examples of how to set optional arguments, see Section 4.2, ''Optional Arguments,'' in Chapter 4, ''Requesting Data and Event Reports,'' in the *Site/SunNet/Domain Manager Application and Agent Development Guide.*

FILES

The proxy agent uses keywords in the **/etc/opt/SUNWconn/snm/snm.conf** file for Solaris 2.x or **/etc/snm.conf** file for Solaris 1.x on the host running the proxy agent. **/etc/opt/SUNWconn/snm/snm.conf** for Solaris 2.x or **/etc/snm.conf** for Solaris 1.x is required on the host running the proxy agent. See the **snm.conf**(5) man page.

snmp.schema(5) describes MIB I, as defined by RFC 1156. **snmp-mibII.schema**(5) describes MIB II, as defined by RFC 1213. **sun-snmp.schema** describes the MIB associated with the Sun-supplied SNMP agent **snmpd**(8) for SPARC systems. This MIB is based on MIB II with SunOS-specific extensions.

NOTES

na.snmp sends all SNMP requests to port 161.

na.snmp usually accepts and displays table key components separated with a period (.). For example, an SNMP atTable key would look like ''1.1.129.144.40.245.'' The exception is where the key for a table in a schema is defined as type MAC_ADDRESS (that is, ''-K MAC_ADDRESS''). The key is then accepted and displayed in the standard, colon-separated hexadecimal format (for example, AA:00:04:00:63:20). Keys can be formatted in either MAC address format or standard dot notation, but not both at the same time. The proxy agent is able to handle a key with any number of components as long as the total key length is less than 128 octets.

ERRORS

attribute unavailable for set operation
The set could not be completed because the attribute was not available for set operations.

bad attribute type
An object attribute type received from the SNMP agent does not match the attribute type specified by the proxy agent schema. The rest of the message indicates the expected type and received type.

bad variable list: no object identifiers
The request's optional argument ''na.snmp.get-list'' or ''na.snmp.get-next-list'' does not contain any object identifiers.

cannot create new subprocess; would exceed maximum: *maximum*
The proxy agent cannot start a new request because creating a new subprocess would exceed the maximum number of proxy agent subprocesses allowed. Either terminate some requests and then resubmit the request, or modify the value associated with the ''na.snmp.max-subprocs'' keyword in **/etc/opt/SUNWconn/snm/snm.conf** for Solaris 2.x or **/etc/snm.conf** for Solaris 1.x. If you change the ''na.snmp.max-subprocs'' value, you must terminate all **na.snmp** requests, kill all **na.snmp** processes, and then restart the SNMP requests.

bad variable value
The requested specified an incorrect syntax or value for a set operation.

cannot dispatch request
The proxy cannot dispatch the request. The rest of the message indicates the cause of the failure.

cannot find default schema: "*schema*" **in search path:** "*path*"
The proxy agent could not find the default schema in the schema search path specified by the ‘‘na.snmp.schemas’’ keyword in the **/etc/opt/SUNWconn/snm/snm.conf** file for Solaris 2.x or in the **/etc/snm.conf** file for Solaris 1.x. Make sure that the default schema is in one of the directories in the schema search path.

cannot find group: "*group*" **in schema**: "*schema*"
The proxy agent could not find the requested group in the specified schema. Make sure that the schemas used by the management application match the schemas used by the proxy agent.

cannot find group: "*group*" **in default schema**: "*schema*"
The proxy agent could not find the requested group in the default schema. Make sure that the schemas used by the management application match the schemas used by the proxy agent.

cannot find schema: "*schema*" **in search path**: "*path*"
The proxy agent could not find the specified schema in the schema search path specified by the ‘‘na.snmp.schemas’’ keyword in the **/etc/opt/SUNWconn/snm/snm.conf** file for Solaris 2.x or in the **/etc/snm.conf** file for Solaris 1.x. Make sure the schema is in one of the directories in the schema search path.

cannot get sysUpTime
The proxy agent cannot get the variable *sysUpTime* from the SNMP agent.

cannot make request PDU
An error occurred building a request PDU.

cannot make request varbind list
An error occurred building a request variable binding list.

cannot open host file
An error occurred opening the file associated with the "na.snmp.hostfile" keyword in **/etc/opt/SUNWconn/snm/snm.conf** for Solaris 2.x and **/etc/snm.conf** for Solaris 1.x.

cannot open schema file
An error occurred opening the proxy agent schema file.

cannot parse host file
The proxy agent was unable to parse the file associated with the ‘‘na.snmp.hostfile’’ keyword in **/etc/opt/SUNWconn/snm/snm.conf** for Solaris 2.x and **/etc/snm.conf** for Solaris 1.x.

cannot parse response PDU
An error occurred parsing a response PDU.

cannot parse schema file
The proxy agent could not parse the proxy agent schema file.

general error
A general error was received.

invalid table key: "*key*"; key must be in dot notation
The requested key must be in standard dot notation, that is, decimal numbers separated by periods.

invalid table key: "*key*"; key must be in MAC address format
The requested key must be in standard MAC address format, that is, hexadecimal numbers separated by colons. This error message only occurs for tables whose key is defined as type MAC_ADDRESS in the associated schema file.

missing attribute
an attribute is missing from the requested group.

no response from system
the SNMP agent on the target system does not respond to SNMP requests. This error might indicate that the SNMP agent is not running on the target system, the target system is down, or the network containing the target system is unreachable.

recvfrom(3N) failed
A **recvfrom (3N)** system call failed. The rest of the message indicates the cause of the failure.

request ID - response ID mismatch
The response ID does not match the request ID.

response too big
The agent could not fit the results of an operation into a single SNMP message. Split large groups or tables into smaller groups.

select(3C) failed
A **select (3C)** system call failed. The rest of the message indicates the cause of the failure.

string contains non-displayable characters
A string contains non-displayable characters.

sysUpTime type bad
The variable *sysUpTime* received from the SNMP agent has the wrong data type.

unknown SNMP error
An unknown SNMP error was received.

variable is read only
The SNMP agent did not perform the set request because a variable to set may not be written.

NAME

na.snmp.hostfile – Site / SunNet / Domain Manager SNMP target configuration file

DESCRIPTION

na.snmp.hostfile is a keyword in **snm.conf**(5) that points to the file the SNMP proxy agent and SNMP trap proxy use to obtain target-specific information.

SYNTAX

Entries are listed one line per system. Each field in the entry is separated by white space. The format is:

host-name read-community write-community request-timeout schema-file [*trap-file proxy-name*]

where

host-name
is the name of the device manageable by the SNMP proxy agent *na.snmp*. The name is not case-sensitive.

read-community
is the SNMP community name (default is ‘‘public’’) the proxy agent uses when reading attribute values from *host-name*.

write-community
is the SNMP community name (default is ‘‘public’’) the proxy agent uses when writing attribute values to *host-name*.

request-timeout
is the time, in seconds (default is 5) that the proxy agent waits for a response from *host-name*.

schema-file
is a colon-separated list of the names of the agent schema files that contain the group and attribute definitions for *host-name*. Note that each name must be an absolute path name that begins with a slash (/). Each schema file must reside in a directory specified by the "na.snmp.schemas" keyword in the **/etc/opt/SUNWconn/snm/snm.conf** file for Solaris 2.x and in the **/etc/snm.conf** file for Solaris 1.x.

The schema for *host-name* is only used on the machine where the agent runs, unlike other agent schema used only by the manager station.

If you add groups (that is, enterprise information) for *host-name* to be managed by the proxy, put the information in this schema file, and add the groups to the schema used by the manager.

trap-file is the name of the trap file that contains the trap definitions for *host-name*. Note that this must be an absolute path name that begins with a slash (/). Default is the file associated with the ‘‘na.snmp-trap.default-trapfile’’ keyword in **snm.conf**(5)**.**

If you wish to use the default trap file, specify "-" as *trap-path*.

vendor-proxy
is an optional field containing the host to send an SNMP request if the SNMP proxy agent cannot send a request directly to an agent on *host-name*. It is

assumed the SNMP agent on *vendor-proxy* can send a request to an agent on *host-name*, possibly using another protocol than the SNMP.

If an entry is not listed for a given host, the SNMP proxy agent uses the groups from the default schema and trap files (specified by the ''na.snmp.default-schema'' and ''na.snmp-trap.default-trapfile'' keywords in **snm.conf**(5)), uses ''public'' for *read-community* and *write-community*, and a *request-timeout* of five seconds.

SEE ALSO

snm.conf(5), **na.snmp**(8), **na.snmp-trap**(8)

NOTES

na.snmp.hostfile is not a file; it is the name of a keyword in **snm.conf**(5)**.** Since you can change the name of the file, it is listed by its keyword.

NAME

na.snmp-trap.default-trapfile – SunNet Manager SNMP trap file

DESCRIPTION

The SNMP trap daemon **na.snmp-trap**(8) translates SNMP traps to SunNet Manager traps. It uses the SNMP trap file to map trap types to names and to determine which trap types should be ignored.

The default file name is defined by the value of this keyword in the **/etc/opt/SUNWconn/snm/snm.conf** file for Solaris 2.x and in the **/etc/snm.conf** file for Solaris 1.x on the system where the trap daemon is running. At installation, this file name is **/var/opt/SUNWconn/snm/snmp.traps** for Solaris 2.x and **/var/adm/snmp.traps** for Solaris 1.x. If you wish to specify a different trap file for a specific device, identify the trap file in an SNMP host file entry for the device.

SYNTAX

The SNMP trap file consists of one or more entries. Each entry defines the traps for one enterprise.

The first line in an entry is of the following format:

enterprise *id*

where

id is the enterprise identifier in dot notation. Set id to "0" to specify the trap names for generic traps.

The remaining lines in an entry are of the following format:

number name **discard**

where

number is the type number of the trap.

name is the name of the trap.

discard
is an optional field that, if set to the string *discard*, indicates that this trap type should be discarded.

EXAMPLE

The following is an example of a trap file entry:

```
enterprise 1.3.6.1.4.1.42
        1       CPU_Failure
        2       Power_Supply_Failure
        3       Network_Connection_Failure
        4       Over_Heating                    discard
        5       RealTimeClock_Failure           discard
```

SEE ALSO

snm.conf(5), **na.snmp-trap**(8)

NOTES

na.snmp-trap.default-trapfile is not a file. It is the name of a keyword in **snm.conf**(5)**.** Since you can change the name of the file, it is listed by its keyword.

NAME

na.snmp.traplog – SunNet Manager SNMP trap logfile

DESCRIPTION

The SNMP trap logger **na.snmp-trap**(8) writes SNMP trap reports to the file specified by the *na.snmp.trap-logfile* keyword listed in the local **snm.conf**(5)**.**

Note this information only pertains to the log files written by the SNMP trap logger. Log files written by other utilities are not covered here.

SYNTAX

Each entry is a newline-terminated ASCII string. Fields in the entry are *name/value* pairs separated by whitespace. Each entry contains the following *names* and their associated *values.*

1. ‘‘sequence’’ - *trap sequence number* (0 is first trap received)
2. ‘‘receive-time’’ - *time trap received* (seconds since UNIX epoch)
3. ‘‘version’’ - *SNMP version*
4. ‘‘community’’ - *SNMP community*
5. ‘‘enterprise’’ - *type of object generating the trap*
6. ‘‘source-address’’ - *address of object generating the trap*
7. ‘‘source-time’’ - *time elapsed between the last (re)initialization of the network entity and the generation of the trap*
8. ‘‘trap-type’’ - *trap type* (either generic trap type or specific trap code)
9. *variable bindings list* containing additional information about the trap with the format:
    1. *variable name*
    2. *variable value*

FILES

snm.conf(5)

NAME

na.snmpv2 – Simple Network Management Protocol - Version 2 proxy agent

DESCRIPTION

The SNMPv2 proxy agent allows a SunNet Manager user to manage SNMPv1 and SNMPv2 devices. **na.snmpv2** can be used with the following SNMP schemas:

- Sun-supplied schema file. (See FILES)
- Schema file supplied by a vendor or other source.
- Schema file that you create from an enterprise MIB. (See **mib2schema**(1) for more information.)

If no reporting interval is specified for requests, the proxy agent will report every 30 seconds.

OPTIONS

Management applications can specify a range of SNMP-related options when making a request to the SNMP proxy agent. For example, the schema name, read and write community strings, and request timeout can be sent with the request.

For requests that are sent from the SNM Console, the read and write community strings and request timeout are specified in the Properties window of the target device. In addition, the Options fields in the Data or Event Request Properties window and in the Set Tool window allow you to specify a community string for the request. The schema you choose for the request is also sent to the SNMP proxy as an optional argument.

For other management applications, optional request arguments are sent to the SNMP proxy with the **netmgt_set_argument**(3N) function. The SNMP proxy agent accepts the following request argument names:

na.snmpv2-read-community
na.snmpv2-write-community
na.snmpv2-schema
na.snmpv2-proxy-device
na.snmpv2-get-list
na.snmpv2-get-next-list
na.snmpv2-timeout

All these request arguments except for **na.snmpv2-timeout** are NETMGT_STRING argument type codes. **na.snmpv2-timeout** is a NETMGT_LONG argument type code. See the **netmgt_set_argument**(3N) man page for a description of how to set these arguments. For examples of how to set optional arguments, see Section 4.2, ''Optional Arguments,'' in Chapter 4, ''Requesting Data and Event Reports,'' in the *Site/SunNet/Domain Manager Application and Agent Development Guide.*

FILES

The proxy agent uses keywords in the **/etc/opt/snm/snm.conf** file for Solaris 2.x and in the **/etc/snm.conf** file for Solaris 1.x on the host running the proxy agent. **/etc/opt/snm/snm.conf** for Solaris 2.x or **/etc/snm.conf** for Solaris 1.x is required on the host running the proxy agent. See the **snm.conf**(5) man page.

snmp.schema(5) describes MIB I, as defined by RFC 1156. **snmp-mibII.schema**(5) describes MIB II, as defined by RFC 1213. **sun-snmp.schema** describes the MIB associated with the Sun-supplied SNMP agent and **snmpv2d**(8) for SPARC systems. This MIB is based on MIB II with SunOS-specific extensions.

NOTES

na.snmpv2 sends all SNMP requests to port 161.

na.snmpv2 usually accepts and displays table key components separated with a period (.). For example, an SNMP atTable key would look like ''1.1.129.144.40.245.'' The exception is where the key for a table in a schema is defined as type MAC_ADDRESS (that is, ''-K MAC_ADDRESS''). The key is then accepted and displayed in the standard, colon-separated hexadecimal format (for example, AA:00:04:00:63:20). Keys can be formatted in either MAC address format or standard dot notation, but not both at the same time. The proxy agent is able to handle a key with any number of components as long as the total key length is less than 128 octets.

ERRORS

attribute unavailable for set operation
: The set could not be completed because the attribute was not available for set operations.

bad attribute type
: An object attribute type received from the SNMP agent does not match the attribute type specified by the proxy agent schema. The rest of the message indicates the expected type and received type.

bad variable list: no object identifiers
: The request's optional argument ''na.snmpv2.get-list'' or ''na.snmpv2.get-next-list'' does not contain any object identifiers.

cannot create new subprocess; would exceed maximum: *maximum*
: The proxy agent cannot start a new request because creating a new subprocess would exceed the maximum number of proxy agent subprocesses allowed. Either terminate some requests and then resubmit the request, or modify the value associated with the ''na.snmpv2.max-subprocs'' keyword in **/etc/opt/snm/snm.conf** for Solaris 2.x and in **/etc/snm.conf** for Solaris 1.x. If you change the ''na.snmpv2.max-subprocs'' value, you must terminate all **na.snmpv2** requests, kill all **na.snmpv2** processes, and then restart the SNMP requests.

bad variable value
: The requested specified an incorrect syntax or value for a set operation.

cannot dispatch request
: The proxy cannot dispatch the request. The rest of the message indicates the cause of the failure.

cannot find default schema: "*schema"* **in search path:** "*path"*
: The proxy agent could not find the default schema in the schema search path specified by the ''na.snmpv2.schemas'' keyword in the **/etc/opt/snm/snm.conf** file for Solaris 2.x and in the **/etc/snm.conf** file for Solaris 1.x. Make sure that the default schema is in one of the directories in the schema search path.

cannot find group: "*group*" **in schema**: "*schema"*
The proxy agent could not find the requested group in the specified schema. Make sure that the schemas used by the management application match the schemas used by the proxy agent.

cannot find group: "*group"* **in default schema**: "*schema"*
The proxy agent could not find the requested group in the default schema. Make sure that the schemas used by the management application match the schemas used by the proxy agent.

cannot find schema: "*schema"* **in search path**: "*path"*
The proxy agent could not find the specified schema in the schema search path specified by the ''na.snmpv2.schemas'' keyword in the **/etc/opt/snm/snm.conf** file for Solaris 2.x and in the **/etc/snm.conf** file for Solaris 1.x. Make sure the schema is in one of the directories in the schema search path.

cannot get sysUpTime
The proxy agent cannot get the variable *sysUpTime* from the SNMP agent.

cannot make request PDU
An error occurred building a request PDU.

cannot make request varbind list
An error occurred building a request variable binding list.

cannot open host file
An error occurred opening the file associated with the "na.snmpv2.hostfile" keyword in **/etc/opt/snm/snm.conf** for Solaris 2.x and **/etc/snm.conf** for Solaris 1.x.

cannot open schema file
An error occurred opening the proxy agent schema file.

cannot parse host file
The proxy agent was unable to parse the file associated with the ''na.snmpv2.hostfile'' keyword in **/etc/opt/snm/snm.conf** for Solaris 2.x and in **/etc/snm.conf** for Solaris 1.x.

cannot parse response PDU
An error occurred parsing a response PDU.

cannot parse schema file
The proxy agent could not parse the proxy agent schema file.

general error
A general error was received.

invalid table key: "*key";* **key must be in dot notation**
The requested key must be in standard dot notation, that is, decimal numbers separated by periods.

invalid table key: "*key";* **key must be in MAC address format**
The requested key must be in standard MAC address format, that is, hexadecimal numbers separated by colons. This error message only occurs for tables whose key is defined as type MAC_ADDRESS in the associated schema file.

missing attribute
an attribute is missing from the requested group.

no response from system
the SNMP agent on the target system does not respond to SNMP requests. This error might indicate that the SNMP agent is not running on the target system, the target system is down, or the network containing the target system is unreachable.

recvfrom(3N) failed
A **recvfrom(3N)** system call failed. The rest of the message indicates the cause of the failure.

request ID - response ID mismatch
The response ID does not match the request ID.

response too big
The agent could not fit the results of an operation into a single SNMP message. Split large groups or tables into smaller groups.

select(3C) failed
A **select(3C)** system call failed. The rest of the message indicates the cause of the failure.

string contains non-displayable characters
A string contains non-displayable characters.

sysUpTime type bad
The variable *sysUpTime* received from the SNMP agent has the wrong data type.

unknown SNMP error
An unknown SNMP error was received.

variable is read only
The SNMP agent did not perform the set request because a variable to set may not be written.

NAME

na.sync – synchronous interface statistics

DESCRIPTION

na.sync returns a table of information about the synchronous interfaces installed in the host running the agent.

ATTRIBUTES

na.sync has two attribute tables, **mode** and **data**.

The **mode** table gathers mode statistics about a particular synchronous interface, using *ifname* as the key. If no key is supplied, the agent returns information about every synchronous interface installed in the machine.

ifname - interface name (string[32])

txclock - transmit clock source (enum)

0 - incoming transmit clock (txc)
1 - incoming receive clock (rxc)
2 - baud rate generator (baud)
3 - phase-lock loop output

rxclock - receive clock source (enum)

0 - incoming receive clock
1 - incoming transmit clock
2 - baud rate generator
3 - phase-lock loop output

Note: The baud rate value is assigned by the system administrator. The actual clock rate is often set by an external modem and can be different.

loopback - do internal loopback (enum)

0 - false
1 - true

baudrate - interface baud rate (unsigned int)

The **data** table gathers I / O statistics for a particular synchronous interface, using *ifname* as the key. If no key is supplied, the agent returns information about every synchronous interface installed in the machine.

ifname - interface name (string[32]).

ipkts - cumulative number of packets received since system boot. (counter)

opkts - cumulative number of packets sent since system boot. (counter)

ibytes - cumulative number of bytes received since system boot. (counter)

obytes - cumulative number of bytes sent since system boot. (counter)

iutil% - current percentage of input utilization. (gauge)

outil% - current percentage of output utilization. (gauge)

aborts - cumulative number of aborts since system boot. (counter)

crcs - cumulative number of CRC errors since system boot. (counter)

over - cumulative number of receiver overruns since system boot. (counter)

under - cumulative number of transmitter underruns since system boot. (counter)

OPTIONS

No options are accepted.

ERRORS

No synchronous interfaces on this host
None of the available interfaces is a synchronous interface.

This key not a synchronous interface
The key (a named interface) is not a synchronous interface.

Error in SIOCGIFCONG ioctl system call
An **ioctl**(2) failed, usually due to an incompatibility at the kernel level.

NAME

na.traffic – network traffic statistics

DESCRIPTION

na.traffic gathers statistics about network traffic (number of packets in a given interval) and reports the information at the end of the interval.

The agent reports Ethernet activity based on information contained in the packet headers, for example, source and destination IP addresses, Ethernet addresses, and port numbers. Each table accepts a key to narrow the information reported, for example, a particular address, port number, packet length, or protocol.

If no reporting interval is given, the agent will use a default reporting interval of 10 seconds.

ATTRIBUTES

na.traffic has one attribute group, **AllPackets**, and nine attribute tables: **SrcIp, DstIP, SrcEth, DstEth, BetweenIp, SrcPort, DstPort, Length** and **Proto**.

The **AllPackets** group returns the total number of Ethernet packets observed during the interval.

The **SrcIp** table contains the number of packets transmitted from a particular IP address, using *srcipaddr* as the key.

Note: An IP address is a string of four decimal numbers (range of 0-255) separated by dots (.) e.g., 129.9.200.3

srcipaddr - source IP address (string[15])

pkts - number of packets from the source IP address per interval (gauge)

The **DstIp** table contains the number of packets sent to a particular IP address, using *dstipaddr* as the key.

dstipaddr - destination IP address (string[15])

pkts - number of packets to the destination IP address per interval (gauge)

The **SrcEth** table contains the number of packets sent from a particular Ethernet address, using *srcethaddr* as the key.

Note: An Ethernet address is a string of six hexadecimal numbers (range is 0-FF) separated by colons (:) e.g., 8:0:20:6:41:0:d7.

srcethaddr - source Ethernet address (string[17])

pkts - number of packets from the source Ethernet address per interval (gauge)

The **DstEth** table contains the number of packets sent to a particular Ethernet address, using *ethaddr* as the key.

ethaddr - destination Ethernet address (string[17])

pkts - number of packets to the destination Ethernet address per interval (gauge)

The **BetweenIp** table contains the number of packets transferred between two given IP addresses, using *ipaddresses* as the key.

ipaddresses - pair of IP addresses (string[30])

pkts - number of packets between the two IP addresses per interval (gauge)

The **SrcPort** table contains the number of packets that originated from a particular port number, using *srcport* as the key.

srcport - port number of source IP address (short)

pkts - number of packets from source port number per interval (gauge)

The **DstPort** table contains the number of packets destined for a particular port number, using *dstport* as the key.

dstport - port number of destination IP address (short)

pkts - number of packets to destination port number per interval (gauge)

The **Length** table contains the number of packets transferred of a particular length or that match a given set of conditions, using *length_exp* as the key.

length_exp - packet length or range of lengths (string[9]). The **length_exp** has the format:

[relational operator] <white space> <packet length>

where:

relational operator
specifies one of the following:

EQ — equal to
GE — greater than or equal to
GT — greater than
LE — less than or equal to
LT — less than.

If you do not supply a relational operator, the agent returns a table showing all packets that fall within the specified range.

white space
single space delimiter separates the relational operator from the packet length.

packet length
specifies a particular packet size range. The defined ranges of packet size (in bytes) are:

60-241
242-423
424-605
606-787
788-969
970-1151
1152-1333
1334-1514.

pkts - number of packets of specified length per interval (gauge)

The **Proto** table contains statistics about a particular protocol, using *protoname* as the key.

protoname - protocol name (string[32]). Currently, **na.traffic** recognizes the following protocols: UDP, TCP, ICMP, ARP, DECnet and AppleTalk.

pkts - number of packets of selected protocol per interval (gauge)

OPTIONS

na.traffic accepts an option in the request.

ifname specifies a particular interface when your machine has more than one (for example, ie0 and ie1). Use this option when the particular interface you are interested in differs from the default name. Pass this option in the form: **-i** *ifname,* where *ifname* is ie0, ie1, and so on.

ERRORS

can't open NIT device for ethernet interface
(**For 4.x systems only.**) The agent could not open the NIT device, **/dev/nit .** NIT may not be present in this kernel, or the device entry is missing from the **/dev** directory.

If your kernel does not support NIT, run **na.traffic** in a system that supports NIT and is on the same Ethernet segment as your machine. Alternatively, you can reconfigure your kernel (see **config**(8)) to support NIT and boot the system.

cannot put ethernet interface into promiscuous mode
(**For 4.x systems only.**) The agent could not set the interface in promiscuous mode. The agent requires the interface to be in promiscuous mode in order to sample all traffic on the net. The most likely cause is a kernel bug.

NIT already in promiscuous mode
(**For 4.x systems only.**) The agent already set the interface into promiscuous mode. The most likely cause is an internal problem with the agent.

can't push a packet filter
(**For 4.x systems only.**) An **ioctl**(2) call to set a message filter failed. The most likely cause is a kernel bug.

bad ethernet address specified
The Ethernet address supplied as a key to either the **SrcEth** or **DstEth** groups has an incorrect format.

bad internet address specified
The Internet address supplied as a key to either the **SrcIp** or **DstIp** groups has an incorrect format.

bad length expression specified
The length expressions specified as a key has an incorrect format.

bad internet address pair specified
The key specification of a pair of Internet addresses has a syntax error. The expected syntax is two Internet addresses, separated by a single space character.

error during sampling
Indicates the occurrence of some undefined system error during the interval the agent sampled the traffic. The agent will cease sampling and return an error

message because recovery may not be possible.

out of memory
Failed to allocate memory for storing network traffic statistics. To alleviate the problem, try reducing the sampling interval or reducing the amount of memory used by other parts of the system.

protocol not yet handled by group Proto
The specified protocol is not known by agent.

bad ethernet interface name specified
The optional specification of the Ethernet interface supplied in the options string has a syntax error. The expected syntax is **-i** *ifname*, where *ifname* is the interface name, such as ie0, le0, ie1.

specify ethernet interface name; unable to determine default
Under some system configurations the agent cannot determine the default interface for sampling Ethernet traffic. Pass the agent the interface name with the **-i** flag in the options field.

can't get interface flags

can't get interface list
Indicate internal errors, usually due to kernel incompatibilities.

NOTES

The statistics returned by **na.traffic** do not include counts for packets sent by the system on which the agent is running.

na.traffic is not intended as a replacement for a protocol analyzer. In fact, the data returned by the traffic agent may differ from statistics obtained by using a protocol analyzer. This is because **na.traffic** places the Ethernet interface on the system into promiscuous mode in order to gather packet statistics. If other processes are running on the system, packets may be lost.

NAME

na.x25 – virtual circuit statistics

DESCRIPTION

na.x25 returns a table of information about an X.25 virtual circuit.

If no default reporting interval is sent, the agent will report every 30 seconds.

ATTRIBUTES

na.x25 has one attribute table, **circuit**, that provides statistics about a particular virtual circuit, using *lcn* as the key.

lcn - logical channel number (short)

linkid - link identifier (int)

in/out - direction of circuit, initiate connection or receive connection (string[5])

hdlc_state - HDLC link state, up or down (string[5])

pkt_lev - packet level, up or down (string[5])

state - X.25 state (string[9])

opkts - number of packets sent (counter). This attribute is valid only if the connection is up. This attribute value is cumulative, and reset to zero when the connection is established.

ipkts - number of packets received (counter). This attribute is valid only if the connection is up. This attribute value is cumulative, and reset to zero when the connection is established.

address - remote address (string[32])

OPTIONS

A non-negative integer corresponding to the interface identifier (also known as "link id") can be specified as an option. In this case, only information pertaining to the specified interface will be returned. This option will work only with version 7.0 of SunNet X.25 (version 6.0 will ignore it).

ERRORS

this link not active
The link specified by the key is not active.

X25 ioctl failed

ioctl (rd_link_stats) fail

ioctl (rd_lcn_info) fail

ioctl (sicgiflags) fail

ioctl (getaddr) fail
See **ioctl**(2).

NAME

netmgt_alloc_manager_id, netmgt_free_manager_id, netmgt_set_manager_id, netmgt_get_manager_id – manipulate the manager application ID

SYNOPSIS

```
u_int   netmgt_alloc_manager_id()

bool_t  netmgt_free_manager_id(manager_id)
u_int   manager_id;

bool_t  netmgt_set_manager_id(manager_id)
u_int   manager_id;

u_int   netmgt_get_manager_id()
```

DESCRIPTION

A manager ID is an unsigned integer associated with a individual manager application. It is transported in SunNet Manager request and report messages and used to distinguish among requests started by a number of manager applications running on the same workstation.

netmgt_alloc_manager_id() allocates a unique ID to the calling manager application. This ID is the inode number of a temporary file in the SunNet Manager directory **/etc/opt/SUNWconn/snm/snm.id** for Solaris 2.x and **/etc/snm/snm.id** for Solaris 1.x. This function may only be called by applications that send data, event, or set requests. Manager applications should call **netmgt_free_manager_id()** to deallocate the manager ID when they are finished.

netmgt_set_manager_id() sets the manager ID field in data, event, and set requests to *manager_id* . This function may only be called by applications that send data, event, or set requests. It should only be called after a call to **netmgt_set_instance**(3N) and before a call to **netmgt_request_data**(3N), **netmgt_request_event**(3N), or **netmgt_request_set**(3N)**.**

netmgt_get_manager_id() returns the value of the manager ID field in data, event, set, trap and error reports. This function may only be called by applications that receive such reports.

netmgt_free_manager_id() deallocates a manager ID previously allocated by a call to **netmgt_alloc_manager_id() .** This function unlinks the temporary file in the SunNet Manager directory **/etc/opt/SUNWconn/snm/snm.id** for Solaris 2.x and **/etc/snm/snm.id** for Solaris 1.x. This function may only be called by applications that send data, event, or set requests.

RETURN VALUES

netmgt_alloc_manager_id() returns a positive integer if successful; otherwise, it returns zero.

netmgt_set_manager_id() and **netmgt_free_manager_id()** return TRUE if successful; otherwise, they return FALSE.

FILES

/etc/opt/SUNWconn/snm/snm.id (Solaris 2.x)

/etc/snm/snm.id (Solaris 1.x)

SEE ALSO

netmgt_kill_request2(3N), **netmgt_set_instance**(3N)

NOTES

These functions only work if the SunNet Manager activity daemon **na.activity** is installed on the local system since it acts as a manager ID server.

netmgt_alloc_manager_id() creates a temporary file in **/etc/opt/SUNWconn/snm/snm.id** on Solaris 2.x and in **/etc/snm/snm.id** on Solaris 1.x for each call. If applications do not call **netmgt_free_manager_id()** to deallocate manager IDs, unused temporary files will accumulate in **/etc/opt/SUNWconn/snm/snm.id** for Solaris 2.x and in **/etc/snm/snm.id** for Solaris 1.x which must then be removed manually.

NAME

netmgt_build_report – add a statistic to a report

SYNOPSIS

```
bool_t  netmgt_build_report (
        Netmgt_data     *data,
        bool_t          *event);
```

DESCRIPTION

netmgt_build_report gets the current attribute value from the data statistic *data* and adds it to a report. If an event has occurred for this attribute, *event* will be set upon return.

You should call this function once for each attribute in the group / table. Once all statistics are added (via repeated calls to this function), the agent should call **netmgt_send_report**(3N) to send the report to the rendezvous process.

There are two methods available for reporting, each with its own strengths. The recommended method is to call **netmgt_send_report**(3N) only after all the attributes have been checked — the entire report has been created. This is the most efficient use of network resources. However, it will fail if the total size of the report is larger than 6KB, which can happen for large tables. The other method is to call **netmgt_send_report**(3N) after each row has been checked. This second method will almost always work, but is not an efficient use of network resources and is discouraged unless the agent returns large reports. This restriction applies only to event reports.

INPUT ARGUMENTS

Pointer to a structure that gets filled in with the following fields:

name attribute name.

type attribute type code, as defined in **netmgt_arglist.h**.

length attribute value length, in bytes.

value pointer to the attribute value buffer. The value's internal representation is described by *type*.

event event status buffer. If an event has occurred for this attribute, this flag will be set to TRUE; otherwise, it will be set to FALSE.

RETURN VALUE

TRUE if successful, otherwise FALSE. Additionally, *event* is set to indicate if an event has occurred.

DIAGNOSTICS

If FALSE is returned, global variable *netmgt_error* indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N).

FILES

netmgt_arglist.h

SEE ALSO

netmgt_fetch_error(3N), **netmgt_send_report**(3N)

NAME

netmgt_dbg – print debugging output

SYNOPSIS

```
#include <stdio.h>

NETMGT_DBG1(format, [, arg] ...)
        char    *format;
NETMGT_DBG2(format, [, arg] ...)
        char    *format;
NETMGT_DBG3(format, [, arg] ...)
        char    *format;
```

DESCRIPTION

These three macros are used for printing application execution tracing information to the standard output stream *stdout*. The arguments are identical to those of **printf**(1).

NETMGT_DBG1
: writes its argument(s) if the debug level is at least one.

NETMGT_DBG2
: writes its argument(s) if the debug level is at least two.

NETMGT_DBG3
: writes its argument(s) if the debug level is at least three.

Use NETMGT_DBG1 for minimal tracing, NETMGT_DBG2 for average tracing, and NETMGT_DBG3 for verbose tracing.

The debug level is set via a call to **netmgt_set_debug**(3N)**.**

INPUT ARGUMENTS

See **printf**(1).

DIAGNOSTICS

None.

SEE ALSO

netmgt_set_debug(3N)

NAME

netmgt_fetch_argument – fetch a request argument

SYNOPSIS

```
bool_t
netmgt_fetch_argument(argname, arg)
        char            *argname;
        Netmgt_arg      *arg;
```

DESCRIPTION

netmgt_fetch_argument gets the optional request argument that corresponds with *argname*. The information is returned in the buffer pointed to by *arg*. If no argument with name *argname* exists in the options list, an error is returned.

INPUT ARGUMENTS

argname
: argument name. The two supplied manager applications **snm**(1) and **snm_cmd**(1) tag their arguments with an *argname* of NETMGT_OPTSTRING (defined in **netmgt_arglist.h**). The agent should use this value.

arg
: pointer to a structure that gets filled in with the following fields:

name
: argument name. The agent can use this value to verify it got what it requested.

type
: argument type code, as defined in the header file **netmgt_arglist.h**.

length
: length (in bytes) of the argument value.

value
: pointer to the argument value buffer. The value's internal representation is described by *type*.

RETURN VALUE

TRUE if successful, otherwise **FALSE.**

DIAGNOSTICS

If FALSE is returned, global variable *netmgt_error* indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N).

FILES

netmgt_arglist.h

SEE ALSO

netmgt_fetch_error(3N)

NAME

netmgt_fetch_data – get data statistics

SYNOPSIS

```
bool_t
netmgt_fetch_data(data)
        Netmgt_data    *data;
```

DESCRIPTION

netmgt_fetch_data gets data statistics from a data report and places them into a local buffer *data*. When a rendezvous receives a data report, the first call to this function gets the first data statistic in the report. Successive calls get successive statistics.

A returned *name* string of NETMGT_ENDOFROW marks the end of a row of tabular data. The string NETMGT_ENDOFARGS marks the end of the report. These strings are defined in **netmgt_arglist.h**. The caller can repeatedly call this function until NETMGT_ENDOFARGS is seen.

INPUT ARGUMENTS

pointer to a buffer where individual data stats are placed. It has the following fields:

name attribute name.

type attribute type code, as defined in **netmgt_arglist.h**.

length length of the attribute value, in bytes.

value attribute value. The value's internal representation is described by *type*.

RETURN VALUE

TRUE if successful, otherwise **FALSE.**

DIAGNOSTICS

If FALSE is returned, global variable *netmgt_error* indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N).

FILES

netmgt_arglist.h

SEE ALSO

netmgt_fetch_error(3N)

NAME

netmgt_fetch_error – fetch a network management error buffer

SYNOPSIS

```
bool_t
netmgt_fetch_error(error)
        Netmgt_error    *error;
```

DESCRIPTION

netmgt_fetch_error copies the relevant portions of the global *netmgt_error* structure into the user buffer pointed to by *error*. The caller should free the memory pointed by *error* upon return.

Two kinds of errors are possible: generic errors, defined in include file **netmgt_errno.h**, and errors specific to the agent, defined in the agent schema *agentErrors* record.

Agent-specific errors can be warnings or fatal errors. Generic errors are always fatal.

INPUT ARGUMENTS

error pointer to a local error structure that gets filled in with the following:

service_error
generic error code from **Netmgt_stat**, listed in the include file **netmgt_errno.h**. Non-generic (agent-specific) errors will use **NETMGT_WARNING** or **NETMGT_FATAL** and list the agent error code in the *agent_error* parameter.

NETMGT_WARNING indicates an agent-specific non-fatal error. The agent will continue to process the request.

NETMGT_FATAL indicates an agent-specific fatal error. The agent will terminate the request.

agent_error
agent-specific error code corresponding to the *agentErrors* record in the agent schema. For generic errors, this field should be ignored.

message optional null-terminated string containing additional information about the error. This field is used for information that cannot be included in the agent schema such as the device name involved in the error.

RETURN VALUE

TRUE if successful, else **FALSE.**

DIAGNOSTICS

FALSE is returned when a NULL *error* buffer was passed. The old value of *netmgt_error* is clobbered; it is replaced by the error code from **netmgt_fetch_error**.

FILES

netmgt_errno.h

SEE ALSO

netmgt_send_error(3N)

NAME

netmgt_fetch_event – fetch event report statistics

SYNOPSIS

```
bool_t
netmgt_fetch_event(event)
        Netmgt_event  *event;
```

DESCRIPTION

netmgt_fetch_event gets event statistics out of an event report and places them into a local buffer *event*. When a rendezvous receives an event report, the first call to this function fetches the first event statistic in the report. Successive calls get successive statistics.

A returned *name* string of NETMGT_ENDOFROW marks the end of a row of tabular data. The string NETMGT_ENDOFARGS marks the end of the report. These strings are defined in **netmgt_arglist.h**. The caller can repeatedly call this function until NETMGT_ENDOFARGS is seen.

INPUT ARGUMENTS

event pointer to a buffer where individual event statistics are placed. It has the following fields:

- *name* attribute name.
- *type* attribute type code, as defined in **netmgt_arglist.h**.
- *length* attribute value length, in bytes.
- *value* attribute value. The value's internal representation is described by *type*.
- *relop* relational operator, as defined in **netmgt_arglist.h**, indicating how the threshold was exceeded. If this attribute didn't trigger the event report, this field will be set to NETMGT_NOP.
- *thresh_val* threshold value. The threshold's internal representation is described by *type*.
- *priority* priority of the event. Possible values are NETMGT_LOW_PRIORITY, NETMGT_MEDIUM_PRIORITY or NETMGT_HIGH_PRIORITY.

RETURN VALUE

TRUE if successful, otherwise **FALSE.**

DIAGNOSTICS

If FALSE is returned, global variable *netmgt_errno* indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N).

FILES

netmgt_arglist.h

SEE ALSO

netmgt_fetch_error(3N)

NOTES

To find out which attributes triggered the event report, cycle through the event statistics via repeated calls to this function. Look for the returned *relop* value to be something besides NETMGT_NOP.

NAME

netmgt_fetch_msginfo – fetch received message information

SYNOPSIS

```
bool_t
netmgt_fetch_msginfo(msginfo)
        Netmgt_msginfo      *msginfo;
```

DESCRIPTION

netmgt_fetch_msginfo obtains additional message information from a data or event report and places it into a local buffer *msginfo.*

INPUT ARGUMENTS

msginfo buffer where the message information is placed. It has the following fields:

manager_addr
: IP address of the manager that requested the event report.

request_time
: time when the agent started the request.

report_time
: time when the agent sent the report. Note that the value comes from the system clock where the agent is running, which may not be synchronized with the clock where the caller is running.

delta_time
: a relative timestamp the that agent returns. This value should be larger than the value returned in the last report from this agent running this request. If the timestamp is smaller, the agent or managed device has probably been reset.

agent_addr
: IP address where the agent is running.

agent_prog
: RPC program number of the agent that sent the report.

agent_vers
: RPC version number of the agent that sent the report.

status error code from **Netmgt_stat** (defined in **netmgt_errno.h**) indicating any service error.

RETURN VALUE

TRUE if successful, otherwise **FALSE.**

DIAGNOSTICS

FALSE is returned when a NULL *msginfo* buffer is passed in. Global variable *netmgt_error* indicates the error. An error buffer is available by calling **netmgt_fetch_error**(3N)**.**

FILES

netmgt_errno.h

SEE ALSO

netmgt_fetch_error(3N)

NAME

netmgt_fetch_setval – fetch set request argument

SYNOPSIS

```
bool_t
netmgt_fetch_setval(setval)
        Netmgt_setval  *setval;
```

DESCRIPTION

netmgt_fetch_setval gets the next set request argument from a set request and places it into a local buffer pointed to by *setval*. When an agent receives a set request, the first call to this function gets the first set request argument in the request. Successive calls get successive arguments.

The string NETMGT_ENDOFARGS marks the end of the request arguments. This string is defined in **netmgt_arglist.h**. The caller can repeatedly call this function until NETMGT_ENDOFARGS is seen.

INPUT ARGUMENTS

setval pointer to a buffer where individual set request arguments are placed. It has the following fields:

- *group* name of the group or table containing the attribute to set.
- *key* table key. A key is only required when setting an attribute in one row of a table.
- *name* attribute name.
- *type* argument type code, as defined in **netmgt_arglist.h**.
- *length* length of the argument value, in bytes.
- *value* argument value buffer. The value's internal representation is described by *type*.

RETURN VALUE

TRUE if successful, otherwise **FALSE.**

DIAGNOSTICS

If FALSE is returned, global variable *netmgt_error* indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N).

FILES

netmgt_arglist.h

SEE ALSO

netmgt_set_value(3N), **netmgt_request_set**(3N), **netmgt_fetch_error**(3N)

NAME

netmgt_alloc_manager_id, netmgt_free_manager_id, netmgt_set_manager_id, netmgt_get_manager_id – manipulate the manager application ID

SYNOPSIS

```
u_int   netmgt_alloc_manager_id()

bool_t  netmgt_free_manager_id(manager_id)
u_int   manager_id;

bool_t  netmgt_set_manager_id(manager_id)
u_int   manager_id;

u_int   netmgt_get_manager_id()
```

DESCRIPTION

A manager ID is an unsigned integer associated with a individual manager application. It is transported in SunNet Manager request and report messages and used to distinguish among requests started by a number of manager applications running on the same workstation.

netmgt_alloc_manager_id() allocates a unique ID to the calling manager application. This ID is the inode number of a temporary file in the SunNet Manager directory **/etc/opt/SUNWconn/snm/snm.id** for Solaris 2.x and **/etc/snm/snm.id** for Solaris 1.x. This function may only be called by applications that send data, event, or set requests. Manager applications should call **netmgt_free_manager_id()** to deallocate the manager ID when they are finished.

netmgt_set_manager_id() sets the manager ID field in data, event, and set requests to *manager_id* . This function may only be called by applications that send data, event, or set requests. It should only be called after a call to **netmgt_set_instance**(3N) and before a call to **netmgt_request_data**(3N), **netmgt_request_event**(3N), or **netmgt_request_set**(3N)**.**

netmgt_get_manager_id() returns the value of the manager ID field in data, event, set, trap and error reports. This function may only be called by applications that receive such reports.

netmgt_free_manager_id() deallocates a manager ID previously allocated by a call to **netmgt_alloc_manager_id() .** This function unlinks the temporary file in the SunNet Manager directory **/etc/opt/SUNWconn/snm/snm.id** for Solaris 2.x and **/etc/snm/snm.id** for Solaris 1.x. This function may only be called by applications that send data, event, or set requests.

RETURN VALUES

netmgt_alloc_manager_id() returns a positive integer if successful; otherwise, it returns zero.

netmgt_set_manager_id() and **netmgt_free_manager_id()** return TRUE if successful; otherwise, they return FALSE.

FILES

/etc/opt/SUNWconn/snm/snm.id (Solaris 2.x)

/etc/snm/snm.id (Solaris 1.x)

SEE ALSO

netmgt_kill_request2(3N), **netmgt_set_instance**(3N)

NOTES

These functions only work if the SunNet Manager activity daemon **na.activity** is installed on the local system since it acts as a manager ID server.

netmgt_alloc_manager_id() creates a temporary file in **/etc/opt/SUNWconn/snm/snm.id** on Solaris 2.x and in **/etc/snm/snm.id** on Solaris 1.x for each call. If applications do not call **netmgt_free_manager_id()** to deallocate manager IDs, unused temporary files will accumulate in **/etc/opt/SUNWconn/snm/snm.id** for Solaris 2.x and in **/etc/snm/snm.id** for Solaris 1.x which must then be removed manually.

NAME

netmgt_alloc_manager_id, netmgt_free_manager_id, netmgt_set_manager_id, netmgt_get_manager_id – manipulate the manager application ID

SYNOPSIS

```
u_int   netmgt_alloc_manager_id()

bool_t  netmgt_free_manager_id(manager_id)
u_int   manager_id;

bool_t  netmgt_set_manager_id(manager_id)
u_int   manager_id;

u_int   netmgt_get_manager_id()
```

DESCRIPTION

A manager ID is an unsigned integer associated with a individual manager application. It is transported in SunNet Manager request and report messages and used to distinguish among requests started by a number of manager applications running on the same workstation.

netmgt_alloc_manager_id() allocates a unique ID to the calling manager application. This ID is the inode number of a temporary file in the SunNet Manager directory **/etc/opt/SUNWconn/snm/snm.id** for Solaris 2.x and **/etc/snm/snm.id** for Solaris 1.x. This function may only be called by applications that send data, event, or set requests. Manager applications should call **netmgt_free_manager_id()** to deallocate the manager ID when they are finished.

netmgt_set_manager_id() sets the manager ID field in data, event, and set requests to *manager_id* . This function may only be called by applications that send data, event, or set requests. It should only be called after a call to **netmgt_set_instance**(3N) and before a call to **netmgt_request_data**(3N), **netmgt_request_event**(3N), or **netmgt_request_set**(3N)**.**

netmgt_get_manager_id() returns the value of the manager ID field in data, event, set, trap and error reports. This function may only be called by applications that receive such reports.

netmgt_free_manager_id() deallocates a manager ID previously allocated by a call to **netmgt_alloc_manager_id() .** This function unlinks the temporary file in the SunNet Manager directory **/etc/opt/SUNWconn/snm/snm.id** for Solaris 2.x and **/etc/snm/snm.id** for Solaris 1.x. This function may only be called by applications that send data, event, or set requests.

RETURN VALUES

netmgt_alloc_manager_id() returns a positive integer if successful; otherwise, it returns zero.

netmgt_set_manager_id() and **netmgt_free_manager_id()** return TRUE if successful; otherwise, they return FALSE.

FILES

/etc/opt/SUNWconn/snm/snm.id (Solaris 2.x)

/etc/snm/snm.id (Solaris 1.x)

SEE ALSO

netmgt_kill_request2(3N), **netmgt_set_instance**(3N)

NOTES

These functions only work if the SunNet Manager activity daemon **na.activity** is installed on the local system since it acts as a manager ID server.

netmgt_alloc_manager_id() creates a temporary file in **/etc/opt/SUNWconn/snm/snm.id** on Solaris 2.x and in **/etc/snm/snm.id** on Solaris 1.x for each call. If applications do not call **netmgt_free_manager_id()** to deallocate manager IDs, unused temporary files will accumulate in **/etc/opt/SUNWconn/snm/snm.id** for Solaris 2.x and in **/etc/snm/snm.id** for Solaris 1.x which must then be removed manually.

NAME

netmgt_init_rpc_agent – initialize an RPC-based agent

SYNOPSIS

```
bool_t
netmgt_init_rpc_agent(name, serial program, version,
                      proto, timeout, reserved, flags,
                      verify, dispatch, reap, shutdown)
        char   *name;
        u_int   serial;
        u_long  program;
        u_long  version;
        u_long  proto;
        struct  timeval timeout;
        u_int   reserved;
        u_int   flags;
        bool_t  (* verify)();
        void    (* dispatch)();
        void    (* reap)();
        void    (* shutdown)();
```

DESCRIPTION

netmgt_init_rpc_agent initializes the Agent Services library and registers the agent with the RPC system to receive requests via RPC from manager processes.

INPUT ARGUMENTS

name agent name as it appears in the **rpc** database.

serial agent serial number, beginning with 1 and increasing for each release of the agent. This is not the RPC version number. It is a method for (among other things) ensuring the agent and its schema file are in sync. In the current release the **snm**(1) and **snm_cmd**(1) managers don't look at this information but future releases will. Therefore, you should include it.

program
RPC program number.

version RPC version number. For the current release, use **NETMGT_VERS**.

proto transport service. **IPROTO_UDP** is the only option available in the current release.

timeout maximum time to wait for a reply from the rendezvous when sending a data or event report.

reserved
currently unused. For compatibility with future releases, set this to zero.

flags startup options. Two options are available to the agent: **NETMGT_DONT_FORK** and **NETMGT_DONT_EXIT**.

If **NETMGT_DONT_FORK** is set, the Agent Services library won't **fork**(2) an agent subprocess to handle the request. Instead, the agent *dispatch* routine is run in the context of the original process.

NETMGT_DONT_FORK is useful for debugging agents (stepping through the *dispatch* function, setting breakpoints, and the like) with debuggers that don't handle child processes well, such as **dbx**(1). This option should not be used in production code.

Two notes about this option: First, the manager request will probably time out because it won't get a confirmation message. Second, the activity daemon won't know about any requests sent to the agent.

If **NETMGT_DONT_EXIT** is set, an agent will not exit when it has no child processes handling requests. Similarly, a command such as **snm_cmd -A -a ping** will not cause the **ping** parent process to exit, although any children will be terminated.

Discretionary use of this option is advised. Usually only special agents like the event dispatcher and activity daemon need to use it.

verify function called to verify the request parameters.

When a request message is received, the *verify* function is called before the agent *dispatch* function is called. If any of the request parameters are invalid, the *verify* function should send an error with **netmgt_send_error**(3N) indicating the problem, and then return FALSE. If the verification succeeds, the *verify* function should return TRUE.

If the *verify* function returns FALSE, the *dispatch* function is not called.

dispatch
function called after a request message has been verified as okay for processing. It is called in the context of the child process, unless **NETMGT_DONT_FORK** has been set in the *flags* argument. It should return rather than **exit**(2) when finished with a request, to allow the Agent Services library to regain control and do final cleanup.

reap optional function used for agent-specific cleanup, called before an agent child process exits.

shutdown
optional function used for agent-specific cleanup, called when an agent parent process exits. If this value is NULL, the Agent Services library will call **netmgt_shutdown_agent**(3N)**.** If the value is not null, the **shutdown** function should call **netmgt_shutdown_agent**(3N) as the last thing it does.

RETURN VALUE

TRUE if successful, otherwise **FALSE.**

DIAGNOSTICS

If FALSE is returned, global variable *netmgt_error* indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N)**.**

SEE ALSO

snm_cmd(1), **netmgt_fetch_error**(3N), **netmgt_shutdown_agent**(3N)
Site/SunNet/Domain Manager Application and Agent Development Guide

NOTES

The Agent Services library uses the following signals in the parent: SIGINT, SIGQUIT, SIGTERM, and SIGCHLD. If the parent wishes to have control passed to its own functions when one of these signals occurs, it should specify the functions in the *shutdown* (for SIGINT, SIGQUIT, and SIGTERM) and *reap* (for SIGCHLD) parameters. One signal is used in the child process: SIGUSR1. Agents should avoid using this signal.

NAME

netmgt_kill_request – terminate a request

SYNOPSIS

```
bool_t
netmgt_kill_request(agent_host, agent_prog, agent_vers,
                    manager_host, timestamp, timeout)
        char    *agent_host;
        u_long  agent_prog;
        u_long  agent_vers;
        char    *manager_host;
        struct  timeval timestamp;
        struct  timeval timeout;
```

DESCRIPTION

netmgt_kill_request terminates the request running on *agent_host* for the agent with RPC program number *agent_prog* and RPC version number *agent_vers* started by a manager process on *manager_host* at time *timestamp.*

INPUT ARGUMENTS

agent_host
: name of the system where the request is running.

agent_prog
: agent's RPC program number.

agent_vers
: agent's RPC version number.

manager_host
: name of the system where the request was started. If *manager_host* is NULL, requests started by any manager will be terminated.

timestamp
: request timestamp. If *timestamp* contains zero seconds and zero microseconds, requests with any timestamp that meets the selection criteria in the other arguments will be terminated.

timeout maximum amount of time to wait for a response from the remote system.

RETURN VALUE

TRUE if successful, otherwise **FALSE.**

DIAGNOSTICS

If FALSE is returned, global variable *netmgt_error* indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N).

SEE ALSO

netmgt_fetch_error(3N)

NAME

netmgt_kill_request2 – terminate requests started by one manager

SYNOPSIS

```
bool_t
netmgt_kill_request2(agent_host, agent_prog, agent_vers,
                     manager_host, manager_id, timeout)
        char     *agent_host;
        u_long   agent_prog;
        u_long   agent_vers;
        char     *manager_host;
        u_int    manager_id;
        struct   timeval timeout;
```

DESCRIPTION

netmgt_kill_request2 terminates all requests running on *agent_host* for the agent with RPC program number *agent_prog* and RPC version number *agent_vers*, started by a manager process on *manager_host* with manager ID *manager_id*.

INPUT ARGUMENTS

agent_host
: name of the system where the request is running.

agent_prog
: agent's RPC program number.

agent_vers
: agent's RPC version number.

manager_host
: name of the system where the request was started.

manager_id
: manager ID indicating which manager application started the request.

timeout maximum amount of time to wait for a response from the remote system.

RETURN VALUE

TRUE if successful, otherwise **FALSE.**

DIAGNOSTICS

If FALSE is returned, global variable *netmgt_error* indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N)**.**

SEE ALSO

netmgt_fetch_error(3N) **netmgt_set_manager_id**(3N)

NAME

netmgt_mark_end_of_row – set table end-of-row

SYNOPSIS

void
netmgt_mark_end_of_row()

DESCRIPTION

netmgt_mark_end_of_row adds the end-of-row string **NETMGT_ENDOFROW** (defined in the header file **netmgt_arglist.h**) to a report. This string allows the rendezvous to determine the end of each row of a table. It should be called after a complete row has been processed, even for single-row reports.

INPUT ARGUMENTS

None.

RETURN VALUE

None.

DIAGNOSTICS

None.

FILES

netmgt_arglist.h

NAME

netmgt_oid2string – convert an object identifier to a printable string

SYNOPSIS

```
char *
netmgt_oid2string(length, value)
    u_int    length;
    caddr_t  value;
```

DESCRIPTION

netmgt_oid2string converts the SNMP object identifier in *value*, stored as an array of unsigned long integers, to a printable character string in dot notation.

INPUT ARGUMENTS

length length of object identifier, in octets.

value pointer to an array of unsigned long integers containing the object identifier value.

RETURN VALUE

Pointer to dot notation string, if successful. Otherwise NULL.

DIAGNOSTICS

If NULL is returned, global variable *netmgt_error* indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N)**.**

SEE ALSO

netmgt_fetch_error(3N)
RFC 1157, A Simple Network Management Protocol

NAME

netmgt_register_callback – register a transient RPC callback function

SYNOPSIS

```
u_long
netmgt_register_callback(callback, udp_sock, tcp_sock, vers, proto)
        void    (* callback)();
        int     *udp_sock;
        int     *tcp_sock;
        u_long  vers;
        u_long  proto;
```

DESCRIPTION

netmgt_register_callback allocates a transient RPC callback program number, creates RPC service transport handles associated with *udp_sock* and / or *tcp_sock*, and associates the callback RPC program number with the service dispatch function *callback* on the local host. This is done to establish a rendezvous for receiving data or event reports.

When a report is received on the local host, *callback* is called with the following arguments:

u_int type
: report type — either NETMGT_DATA_REPORT, NETMGT_EVENT_REPORT, NETMGT_TRAP_REPORT, or NETMGT_ERROR_REPORT.

*char *target*
: name of the target system (where the managed element resides).

*char *group*
: name of the group whose attribute values are to be retrieved.

*char *key*
: optional row identifier for tables. An instance of the *group* on the managed *system*. This is used if the group is a table with more than one possible row.

u_int count
: the reporting count.

timeval interval
: the reporting interval.

u_int flags
: report options.

INPUT ARGUMENTS

callback function that will handle reports as they arrive.

udp_sock
: socket descriptor associated with a UDP transport RPC callback function. This should be set to RPC_ANYSOCK if you don't want to specify a socket descriptor.

tcp_sock
: socket descriptor associated with a TCP transport RPC callback function. This should be set to RPC_ANYSOCK if you don't want to specify a socket descriptor.

vers RPC callback function version number.

proto transport protocol of the callback function. Specify IPROTO_UDP|IPPROTO_TCP. The result of this setting is that UDP transport is used for messages of 6144 bytes or less; TCP transport is used for messages longer than 6144 bytes. Because of this feature, if you registered, for example, only with UDP, you would not receive any large reports. If there is a possibility that you will receive reports larger than 6144 bytes, you should register with both TCP and UDP. This argument creates two sockets that must be closed just before calling **netmgt_unregister_callback**(3N)**.** (See ''NOTES'' below.)

RETURN VALUE

A transient RPC program number, if successful, otherwise NULL.

DIAGNOSTICS

If NULL is returned, global variable *netmgt_error* indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N)**.**

SEE ALSO

netmgt_fetch_error(3N), **netmgt_unregister_callback**(3N),

NOTES

The caller must call **netmgt_unregister_callback**(3N) to unregister the transient RPC program number before exiting. Otherwise, the non-existent program number will continue to be registered with the local portmapper. The calling program should close both UDP and TCP sockets. These sockets should be closed just before calling **netmgt_unregister_callback**(3N)**,** or your program may run out of file descriptors.

NAME

netmgt_register_rendez – register with the event dispatcher

SYNOPSIS

```
bool_t
netmgt_register_rendez(event_host, rendez_host, rendez_prog, rendez_vers,
                       agent_prog, priority, timeout)
        char        *event_host;
        char        *rendez_host;
        u_long      rendez_prog;
        u_long      rendez_vers;
        u_long      agent_prog;
        u_int       priority;
        struct timeval timeout;
```

DESCRIPTION

netmgt_register_rendez instructs the event dispatcher on *event_host* to start sending event reports with priority *priority* and higher from agents with RPC program number *agent_prog* to the rendezvous with RPC program number *rendez_prog* and RPC version number *rendez_vers* on *rendez_host*.

INPUT ARGUMENTS

event_host
: name of the system where the event dispatcher is running.

rendez_host
: name of the system where the rendezvous is running.

rendez_prog
: RPC program number of the rendezvous.

rendez_vers
: RPC version number of the rendezvous.

agent_prog
: RPC program number of the agent of interest. If *agent_prog* is NETMGT_ANY_AGENT, event reports from all agents will be sent. If *agent_prog* is NETMGT_DBMGR_PROG, only trap reports that indicate the addition, modification, or deletion of elements in the runtime database will be sent.

event_priority
: lowest priority of event reports the rendezvous process wants to receive. *priority* can be one of three values: NETMGT_LOW_PRIORITY, NETMGT_MEDIUM_PRIORITY or NETMGT_HIGH_PRIORITY.

timeout maximum time to wait for confirmation from the event dispatcher.

RETURN VALUE

TRUE if successful, otherwise FALSE.

DIAGNOSTICS

If FALSE is returned, global variable *netmgt_error* indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N)**.**

SEE ALSO **netmgt_fetch_error**(3N), **netmgt_unregister_rendez**(3N), **na.event**(8)

NOTES The caller must call **netmgt_unregister_rendez**(3N) before it exits or when it is no longer interested in getting event reports. Otherwise, a copy of the event dispatcher will continue to run.

NAME

netmgt_request_agent_ID – request identification from an agent

SYNOPSIS

```
Netmgt_ident *
netmgt_request_agent_ID(agent_host, agent_prog, agent_vers, timeout)
        char     *agent_host;
        u_long   agent_prog;
        u_long   agent_vers;
        struct   timeval timeout;
```

DESCRIPTION

netmgt_request_agent_ID gets information identifying an agent.

INPUT ARGUMENTS

agent_name
system where the agent is installed.

agent_prog
agent's RPC program number.

agent_vers
agent's RPC version number.

timeout maximum time to wait for a response from the agent before failing with an error.

RETURN VALUE

Pointer to a structure (as defined in **netmgt_agent.h**) containing the following fields:

char name[NETMGT_NAMESIZ]
agent name as it appears in the **rpc** database.

u_int serial
agent serial number. This serial number changes with the agent schema; they should match.

char arch[NETMGT_NAMESIZ]
host ID (not architecture) of the machine where the agent is installed.

If an error is detected, the function returns NULL.

DIAGNOSTICS

If NULL is returned, global variable *netmgt_error* indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N).

FILES

netmgt_agent.h

SEE ALSO

hostid(1B), **netmgt_fetch_error**(3N)

NAME

netmgt_request_data – request data reports from an agent

SYNOPSIS

```
struct timeval *
netmgt_request_data(agent_host, agent_prog, agent_vers,
                    rendez_host, rendez_prog, rendez_vers,
                    count, interval, timeout, flags)
        char        *agent_host;
        u_long       agent_prog;
        u_long       agent_vers;
        char        *rendez_host;
        u_long       rendez_prog;
        u_long       rendez_vers;
        u_int       count;
        struct timeval interval;
        struct timeval timeout;
        u_int       flags;
```

DESCRIPTION

netmgt_request_data sends a request to an agent for a specific group's attribute values. The caller must have previously called **netmgt_set_instance**(3N) to define the system, group, and optional key of the request. If any agent-specific options are to be set, the caller must have previously called **netmgt_set_argument**(3N)**.**

The request is sent to the agent process on *agent_host* with RPC program number *agent_prog* and RPC version number *agent_vers.* The agent will report its collected data to the rendezvous process on *rendez_host* with RPC program number *rendez_prog* and RPC version number *rendez_vers.*

INPUT ARGUMENTS

agent_host
: host where the agent runs.

agent_prog
: agent's RPC program number.

agent_vers
: agent's RPC version number.

rendez_host
: host where the rendezvous runs.

rendez_prog
: rendezvous' RPC program number.

rendez_vers
: rendezvous' RPC version number.

count maximum number of reports wanted. If *count* is zero, the agent will send reports until it is told to stop.

interval reporting interval. If *interval* is zero, the agent will use its own default interval.

timeout maximum time to wait for confirmation from the agent before the call fails.

flags request option flags. Two flags are currently defined:

NETMGT_RESTART

Restart the request if the agent abnormally terminates and is restarted. Otherwise, when the agent restarts, this request is forgotten.

This flag is advisory; request restart is not guaranteed. The restart is not attempted until the agent parent is started (asked to start another request, or asked what requests it is working on) when all requests marked for restart will be restarted, if possible.

By default, this flag is not set.

NETMGT_DO_DEFERRED

Have the agent collect the data but *don't send it just yet*.

In many cases, an agent collects useful information for debugging problems, but the information isn't useful under normal conditions. If the manager started the request only after the error condition started, it would have been started after the fact and valuable data would have been lost. On the other hand, if the request was started and the data reports were continually streaming back to the manager before the error condition occurred, an unnecessary traffic and CPU load would be caused from lots of uninteresting data coming back.

With this flag, data reports can be held on the agent's system until the manager is ready (if ever) to ask for them. When the manager asks via a call to **netmgt_request_deferred**(3N) for the collected reports, the ''old'' reports will be sent.

This option (as well as **netmgt_request_deferred**(3N)) is handled transparently for the agent by the Agent Services library. The library keeps only the last 32 reports the agent ''sent,'' memory permitting.

By default, this flag is not set.

RETURN VALUE

Timestamp of the request, if successful, otherwise NULL.

DIAGNOSTICS

If NULL is returned, global variable *netmgt_error* indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N)**.**

SEE ALSO

netmgt_fetch_error(3N), **netmgt_request_deferred**(3N), **netmgt_set_argument**(3N), **netmgt_set_instance**(3N)

NAME

netmgt_request_deferred – get any available deferred data reports

SYNOPSIS

```
bool_t
netmgt_request_deferred(agent_host, agent_prog, agent_vers,
                        manager_host, timestamp, timeout)
        char        *agent_host;
        u_long       agent_prog;
        u_long       agent_vers;
        char        *manager_host;
        struct timeval timestamp;
        struct timeval timeout;
```

DESCRIPTION

netmgt_request_deferred instructs the agent running on *agent_host* with the RPC program number *agent_prog* and RPC version number *agent_vers*, to return any deferred data reports to the rendezvous. The manager process on host *manager_host* has previously requested the agent to defer sending. *timestamp* identifies the request.

Upon return of the function, the queue of deferred reports on the agent side is emptied, and the agent continues deferring data reports.

INPUT ARGUMENTS

agent_host
: name of host where the agent is executing.

agent_prog
: agent RPC program number.

agent_vers
: agent RPC version number.

manager_host
: host name of original request.

timestamp
: when the original request was started.

timeout maximum amount of time to wait for request confirmation.

RETURN VALUE

TRUE is successful, otherwise **FALSE.**

DIAGNOSTICS

If FALSE is returned, global variable *netmgt_error* indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N).

SEE ALSO

netmgt_fetch_error(3N), **netmgt_request_data**(3N)

NAME

netmgt_request_events – request event reports from an agent

SYNOPSIS

```
struct timeval *
netmgt_request_events(agent_host, agent_prog, agent_vers,
                      rendez_host, rendez_prog, agent_vers,
                      count, interval, timeout, flags)
        char        *agent_host;
        u_long       agent_prog;
        u_long       agent_vers;
        char        *rendez_host;
        u_long       rendez_prog;
        u_long       rendez_vers;
        u_int        count;
        struct timeval  interval;
        struct timeval  timeout;
        u_int        flags;
```

DESCRIPTION

netmgt_request_events sends a request to an agent to watch for interesting events on specific attributes. The caller must have previously called **netmgt_set_instance**(3N) to define the system, group, and optional key of the request, and **netmgt_set_threshold**(3N) to define the threshold and relation parameters of the request. If any agent-specific options are to be set, the caller must have previously called **netmgt_set_argument**(3N)**.**

The request is sent to the agent process on *agent_host* with RPC program number *agent_prog* and RPC version number *agent_vers.*

An event report is sent when the infix expression

<*attribute value*> <*relational operator*> <*threshold value*>

is TRUE.

If an event occurs, an event report will be sent to the rendezvous on *rendez_host* with RPC program number *rendez_prog* and RPC version number *rendez_vers.*

INPUT ARGUMENTS

agent_host
host where the agent runs.

agent_prog
agent's RPC program number.

agent_vers
agent's RPC version number.

rendez_host
host where the rendezvous runs.

rendez_prog
rendezvous' RPC program number.

rendez_vers
rendezvous' RPC version number.

count maximum number of data reports wanted. An event report may be sent up to *count* times, once per *interval*, but only if an event occurs. Otherwise no report is sent.

If *count* is zero, event reporting will continue until the agent is told to stop.

interval reporting interval. If an event has not occurred when the report is scheduled to be sent, no report will be sent.

If *interval* is zero, the agent will use its own default interval.

timeout maximum time to wait for confirmation from the agent before the call fails.

flags request options flags. Three flags are currently defined:

NETMGT_SEND_ONCE

Have the agent terminate after one event report has been sent. If this flag is not set, the agent continues to conditionally send event reports until otherwise directed.

By default, this flag is not set.

NETMGT_RESTART

Restart the request if the agent abnormally terminates and is restarted. Otherwise, when the agent restarts, this request is forgotten.

This flag is advisory; request restart is not guaranteed. The restart is not attempted until the agent parent is started (asked to start another request, or asked what requests it is working on), when all requests marked for restart will be restarted, if possible.

By default, this flag is not set.

NETMGT_DO_DEFERRED

Have the agent collect the event but *don't send it just yet*.

In many cases, an agent collects useful information for debugging problems, but the information isn't useful under normal conditions. If the manager started the request only after the error condition started, it would have been started after the fact, and valuable data would have been lost. On the other hand, if the request was started and the event reports were continually streaming back to the manager before the error condition occurred, an unnecessary traffic and CPU load would be caused from lots of uninteresting event reports coming back.

With this flag, data reports can be held on the agent's system until the manager is ready (if ever) to ask for them. When the manager asks via a call to **netmgt_request_deferred**(3N) for the collected reports, the ''old'' reports will be sent.

This option (as well as **netmgt_request_deferred**(3N)) is handled transparently for the agent by the Agent Services library. The library keeps only the last 32 reports the agent ''sent,'' memory permitting.

By default, this flag is not set.

RETURN VALUE

Timestamp of the request, if successful, otherwise NULL.

DIAGNOSTICS

If NULL is returned, global variable *netmgt_error* indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N)**.**

SEE ALSO

netmgt_fetch_error(3N), **netmgt_request_deferred**(3N), **netmgt_set_argument**(3N), **netmgt_set_instance**(3N), **netmgt_set_threshold**(3N)

NOTES

Deferring event reports is a dubious activity.

NAME

netmgt_request_set– send a set request to an agent

SYNOPSIS

```
bool_t
netmgt_request_set(agent_host, agent_prog, agent_vers,
                   rendez_host, rendez_prog, rendez_vers,
                   timeout, flags)
        char        *agent_host;
        u_long       agent_prog;
        u_long       agent_vers;
        char        *rendez_host;
        u_long       rendez_prog;
        u_long       rendez_vers;
        struct timeval timeout;
        u_int        flags;
```

DESCRIPTION

netmgt_request_set sends a request to an agent to set the values of one or more group or table attributes. The caller must have previously called **netmgt_set_instance**(3N) to define the system containing the attributes. In addition, the caller must have previously called **netmgt_set_value**(3N) to specify the values to set the attributes. If any agent-specific options are to be set, the caller must have previously called **netmgt_set_argument**(3N)**.**

The request is sent to the agent process on *agent_host* with RPC program number *agent_prog* and RPC version number *agent_vers*. The agent will report the results of the set operation to the rendezvous process on *rendez_host* with RPC program number *rendez_prog* and RPC version number *rendez_vers*.

INPUT ARGUMENTS

agent_host
: host where the agent runs.

agent_prog
: agent's RPC program number.

agent_vers
: agent's RPC version number.

rendez_host
: host where the rendezvous runs.

rendez_prog
: rendezvous' RPC program number.

rendez_vers
: rendezvous' RPC version number.

timeout maximum time to wait for confirmation from the agent before the call fails.

flags request option flags. One flag is currently defined:

NETMGT_RESTART
: Restart the request if the agent abnormally terminates and is restarted.

Otherwise, when the agent restarts, this request is forgotten.

This flag is advisory; request restart is not guaranteed. The restart is not attempted until the agent parent is started (asked to start another request, or asked what requests it is working on), when all requests marked for restart will be restarted, if possible.

RETURN VALUE

TRUE if the agent verified it will attempt to perform the request, otherwise FALSE. The caller must run as a callback RPC server to receive a report indicating whether the agent succeeded in performing the set request.

DIAGNOSTICS

If FALSE is returned, global variable *netmgt_error* indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N)**.**

SEE ALSO

netmgt_fetch_error(3N), **netmgt_set_value**(3N), **netmgt_set_argument**(3N), **netmgt_set_instance**(3N)

NAME

netmgt_request_set2– send a set request to an agent

SYNOPSIS

```
struct timeval *
netmgt_request_set2(agent_host, agent_prog, agent_vers,
                    rendez_host, rendez_prog, rendez_vers,
                    timeout, flags)
        char        *agent_host;
        u_long       agent_prog;
        u_long       agent_vers;
        char        *rendez_host;
        u_long       rendez_prog;
        u_long       rendez_vers;
        struct timeval timeout;
        u_int        flags;
```

DESCRIPTION

netmgt_request_set2 sends a request to an agent to set the values of one or more group or table attributes. The caller must have previously called **netmgt_set_instance**(3N) to define the system containing the attributes. In addition, the caller must have previously called **netmgt_set_value**(3N) to specify the values to set the attributes. If any agent-specific options are to be set, the caller must have previously called **netmgt_set_argument**(3N).

The request is sent to the agent process on *agent_host* with RPC program number *agent_prog* and RPC version number *agent_vers*. The agent will report the results of the set operation to the rendezvous process on *rendez_host* with RPC program number *rendez_prog* and RPC version number *rendez_vers*.

INPUT ARGUMENTS

agent_host
: host where the agent runs.

agent_prog
: agent's RPC program number.

agent_vers
: agent's RPC version number.

rendez_host
: host where the rendezvous runs.

rendez_prog
: rendezvous' RPC program number.

rendez_vers
: rendezvous' RPC version number.

timeout maximum time to wait for confirmation from the agent before the call fails.

flags request option flags. One flag is currently defined:

NETMGT_RESTART
: Restart the request if the agent abnormally terminates and is restarted.

Otherwise, when the agent restarts, this request is forgotten.

This flag is advisory; request restart is not guaranteed. The restart is not attempted until the agent parent is started (asked to start another request, or asked what requests it is working on), when all requests marked for restart will be restarted, if possible.

RETURN VALUE

timestamp of the request, if successful. Otherwise NULL. The caller must run as a call-back RPC server to receive a report indicating whether the agent succeeded in performing the set request.

NOTES

netmgt_request_set2 performs the same function as **netmgt_request_set. netmgt_request_set2** returns a request timestamp like **netmgt_request_data** or **netmgt_request_event** instead of a Boolean value.

DIAGNOSTICS

If NULL is returned, global variable *netmgt_error* indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N)**.**

SEE ALSO

netmgt_fetch_error(3N), **netmgt_set_value**(3N), **netmgt_set_argument**(3N), **netmgt_set_instance**(3N), **netmgt_request_set**(3N)

NAME

netmgt_save_argument, netmgt_restore_argument – save and restore optional argument

SYNOPSIS

```
bool_t
netmgt_save_argument (name, argument)
        char *name,
        Netmgt_argument *argument;
bool_t
netmgt_restore_argument (argument)
        Netmgt_argument *argument;
```

DESCRIPTION

netmgt_save_argument copies information about an optional argument into the buffer pointed to by *argument* .

netmgt_restore_argument sets a argument from the contents of the buffer pointed to by *argument* .

```
typedef struct
{
 char name [NETMGT_NAMESIZ] ;          /* argument name */
 u_int type ;                          /* argument type */
 u_int length ;                        /* argument length */
 u_char value [NETMGT_VALUESIZ] ;      /* argument value buffer */
}    Netmgt_argument ;
```

The members of this structure are:

name Optional argument name

type Argument data type

length Argument value length

value Argument value buffer

DESCRIPTION

TRUE if successful, otherwise **FALSE.**

DIAGNOSTICS

If false is returned, global variable netmgt_errno indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N)**.**

FILES

netmgt_arglist.h, netmgt_msg.h, netmgt_request.h

SEE ALSO

netmgt_fetch_error(3N), **netmgt_save_request**(3N), **netmgt_save_threshold**(3N)

NOTES

Only agents may call this function.

NAME

netmgt_save_request, netmgt_restore_request – save and restore current request state

SYNOPSIS

```
bool_t
netmgt_save_request (request_info)
        Netmgt_request *request_info;
bool_t
netmgt_restore_request (request_info)
        Netmgt_request *request_info;
```

DESCRIPTION

netmgt_save_request copies information about the current request state into the buffer pointed to by 'request_info'.

netmgt_restore_request sets the current request state from the contents of the buffer pointed to by 'request_info'.

```
typedef struct
{
 struct in_addr manager_addr ;          /* manager IP address */
 struct timeval request_time ;          /* request timestamp */
 u_int type ;                           /* request type */
 u_int manager_id ;                     /* manager ID */
 u_int flags ;                          /* request flags */
 u_int priority ;                       /* request priority */
 struct in_addr rendez_addr ;           /* rendezvous IP address */
 u_long rendez_prog ;                   /* rendezvous RPC program number */
 u_long rendez_vers ;                   /* rendezvous RPC version number */
 u_long proto;                          /* transport protocol */
 struct timeval interval ;              /* reporting interval */
 u_int count ;                          /* reporting count */
 char system [NETMGT_NAMESIZ] ;         /* system */
 char group [NETMGT_NAMESIZ] ;          /* group */
 char key [NETMGT_NAMESIZ] ;            /* key */
 u_int num_arguments ;                  /* number of optional arguments */
 u_int num_thresholds ;                 /* number of event thresholds
                                                or data attributes */
 u_int num_setvals ;                    /* number of set arguments */
}       Netmgt_request ;
```

The members of this structure are:

manager_addr
IP address of manager that sent the request

request_time
Time when the manager sent the request

type Request type. See **netmgt_arglist.h**

manager_id

ID of the manager that sent the request

flags Request flags. See **netmgt_msg.h** and **netmgt_request.h**

priority
Request priority (currently unused)

rendez_addr
IP address of report rendezvous

rendez_prog
RPC program number of report rendezvous

rendez_vers
RPC version number of report rendezvous

proto Transport protocol used to send the request (either IPPROTO_UDP or IPPROTO_TCP)

interval
Report interval

count Report count

system Target system name

group Schema group if a data or event request

key Table key if a data or event request for tabular group

num_arguments
Number of optional arguments

num_thresholds
Number of thresholds if an event request or number of attributes in case of a data request

num_setvals
Number of set arguments if a set request

RETURN VALUE

TRUE if successful, otherwise **FALSE.**

DIAGNOSTICS

If false is returned, global variable **netmgt_errno** indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N)**.**

FILES

netmgt_arglist.h, netmgt_msg.h, netmgt_request.h

SEE ALSO

netmgt_fetch_error(3N), **netmgt_save_argument**(3N), **netmgt_save_threshold**(3N)

NOTES

Only agents may call this function.

NAME

netmgt_save_threshold, netmgt_restore_threshold – save and restore event threshold

SYNOPSIS

```
bool_t
netmgt_save_threshold (name, threshold)
        char *name,
        Netmgt_threshold *threshold;
bool_t
netmgt_restore_threshold (threshold)
        Netmgt_threshold *threshold;
```

DESCRIPTION

netmgt_save_threshold copies information about an event threshold into the buffer pointed to by 'threshold'.

netmgt_restore_threshold sets a threshold from the contents of the buffer pointed to by 'threshold'.

```
typedef struct
{
 char name [NETMGT_NAMESIZ] ;                  /* attribute name */
 u_int type ;                                  /* attribute type */
 u_int relop ;                                 /* relational operator */
 u_int thresh_len ;                            /* threshold value length */
 u_char thresh_val [NETMGT_VALUESIZ] ;         /* threshold value buffer */
 u_int prev_len ;                              /* previous attribute value length */
 u_char prev_val [NETMGT_VALUESIZ];            /* previous attribute value buffer */
 u_int priority ;                              /* event priority */
}    Netmgt_threshold ;
```

The members of this structure are:

name Name of the attribute on which the threshold is set

type Attribute data type

relopi Threshold relational operator

thresh_len
Threshold value length

thresh_val
Threshold value buffer

prev_len
Previous attribute value length

prev_val
Previous attribute value

priority
Event priority

RETURN VALUE **TRUE** if successful, otherwise **FALSE.**

DIAGNOSTICS If false is returned, global variable netmgt_errno indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N)**.**

FILES **netmgt_arglist.h, netmgt_msg.h, netmgt_request.h**

SEE ALSO **netmgt_fetch_error**(3N), **netmgt_save_request**(3N), **netmgt_save_argument**(3N)

NOTES Only agents may call this function.

NAME

netmgt_save_argument, netmgt_restore_argument – save and restore optional argument

SYNOPSIS

```
bool_t
netmgt_save_argument (name, argument)
        char *name,
        Netmgt_argument *argument;
bool_t
netmgt_restore_argument (argument)
        Netmgt_argument *argument;
```

DESCRIPTION

netmgt_save_argument copies information about an optional argument into the buffer pointed to by *argument* .

netmgt_restore_argument sets a argument from the contents of the buffer pointed to by *argument* .

```
typedef struct
{
 char name [NETMGT_NAMESIZ] ;          /* argument name */
 u_int type ;                          /* argument type */
 u_int length ;                        /* argument length */
 u_char value [NETMGT_VALUESIZ] ;      /* argument value buffer */
}    Netmgt_argument ;
```

The members of this structure are:

name Optional argument name

type Argument data type

length Argument value length

value Argument value buffer

DESCRIPTION

TRUE if successful, otherwise **FALSE.**

DIAGNOSTICS

If false is returned, global variable netmgt_errno indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N)**.**

FILES

netmgt_arglist.h, netmgt_msg.h, netmgt_request.h

SEE ALSO

netmgt_fetch_error(3N), **netmgt_save_request**(3N), **netmgt_save_threshold**(3N)

NOTES

Only agents may call this function.

NAME

netmgt_save_attribute - save a data attribute

SYNOPSIS

```
bool_t  netmgt_save_attribute (
            Netmgt_attr    *attr);
```

DESCRIPTION

netmgt_save_attribute copies information about a data attribute into the buffer pointed to by *attr*.

attr is a pointer to an object with the following structure:

```
typedef struct {
        char    name [NETMGT_NAMESIZ];   /* attribute name */
        u_int   type;                    /* attribute type */
} Netmgt_attr ;
```

Parameters

name name of the attribute

type attribute data type

RETURN VALUE

TRUE if successful, otherwise **FALSE.**

DIAGNOSTICS

If **FALSE** is returned, global variable **netmgt_errno** indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N).

FILES

netmgt_arglist.h, netmgt_msg.h, netmgt_request.h

SEE ALSO

netmgt_fetch_error(3N),
netmgt_save_request(3N),
netmgt_save_argument(3N)

NOTE

Only agents may call this function.

NAME

netmgt_save_request, netmgt_restore_request – save and restore current request state

SYNOPSIS

```
bool_t
netmgt_save_request (request_info)
        Netmgt_request *request_info;
bool_t
netmgt_restore_request (request_info)
        Netmgt_request *request_info;
```

DESCRIPTION

netmgt_save_request copies information about the current request state into the buffer pointed to by 'request_info'.

netmgt_restore_request sets the current request state from the contents of the buffer pointed to by 'request_info'.

```
typedef struct
{
 struct in_addr manager_addr ;          /* manager IP address */
 struct timeval request_time ;          /* request timestamp */
 u_int type ;                           /* request type */
 u_int manager_id ;                     /* manager ID */
 u_int flags ;                          /* request flags */
 u_int priority ;                       /* request priority */
 struct in_addr rendez_addr ;           /* rendezvous IP address */
 u_long rendez_prog ;                   /* rendezvous RPC program number */
 u_long rendez_vers ;                   /* rendezvous RPC version number */
 u_long proto;                          /* transport protocol */
 struct timeval interval ;              /* reporting interval */
 u_int count ;                          /* reporting count */
 char system [NETMGT_NAMESIZ] ;         /* system */
 char group [NETMGT_NAMESIZ] ;          /* group */
 char key [NETMGT_NAMESIZ] ;            /* key */
 u_int num_arguments ;                  /* number of optional arguments */
 u_int num_thresholds ;                 /* number of event thresholds
                                                or data attributes */
 u_int num_setvals ;                    /* number of set arguments */
}       Netmgt_request ;
```

The members of this structure are:

manager_addr
IP address of manager that sent the request

request_time
Time when the manager sent the request

type Request type. See **netmgt_arglist.h**

manager_id

ID of the manager that sent the request

flags Request flags. See **netmgt_msg.h** and **netmgt_request.h**

priority
Request priority (currently unused)

rendez_addr
IP address of report rendezvous

rendez_prog
RPC program number of report rendezvous

rendez_vers
RPC version number of report rendezvous

proto Transport protocol used to send the request (either IPPROTO_UDP or IPPROTO_TCP)

interval
Report interval

count Report count

system Target system name

group Schema group if a data or event request

key Table key if a data or event request for tabular group

num_arguments
Number of optional arguments

num_thresholds
Number of thresholds if an event request or number of attributes in case of a data request

num_setvals
Number of set arguments if a set request

RETURN VALUE

TRUE if successful, otherwise **FALSE.**

DIAGNOSTICS

If false is returned, global variable **netmgt_errno** indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N)**.**

FILES

netmgt_arglist.h, netmgt_msg.h, netmgt_request.h

SEE ALSO

netmgt_fetch_error(3N), **netmgt_save_argument**(3N), **netmgt_save_threshold**(3N)

NOTES

Only agents may call this function.

NAME

netmgt_save_threshold, netmgt_restore_threshold – save and restore event threshold

SYNOPSIS

```
bool_t
netmgt_save_threshold (name, threshold)
        char *name,
        Netmgt_threshold *threshold;
bool_t
netmgt_restore_threshold (threshold)
        Netmgt_threshold *threshold;
```

DESCRIPTION

netmgt_save_threshold copies information about an event threshold into the buffer pointed to by 'threshold'.

netmgt_restore_threshold sets a threshold from the contents of the buffer pointed to by 'threshold'.

```
typedef struct
{
 char name [NETMGT_NAMESIZ] ;                  /* attribute name */
 u_int type ;                                  /* attribute type */
 u_int relop ;                                 /* relational operator */
 u_int thresh_len ;                            /* threshold value length */
 u_char thresh_val [NETMGT_VALUESIZ] ;         /* threshold value buffer */
 u_int prev_len ;                              /* previous attribute value length */
 u_char prev_val [NETMGT_VALUESIZ];            /* previous attribute value buffer */
 u_int priority ;                              /* event priority */
}   Netmgt_threshold ;
```

The members of this structure are:

name Name of the attribute on which the threshold is set

type Attribute data type

relopi Threshold relational operator

thresh_len
Threshold value length

thresh_val
Threshold value buffer

prev_len
Previous attribute value length

prev_val
Previous attribute value

priority
Event priority

RETURN VALUE **TRUE** if successful, otherwise **FALSE.**

DIAGNOSTICS If false is returned, global variable netmgt_errno indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N)**.**

FILES **netmgt_arglist.h, netmgt_msg.h, netmgt_request.h**

SEE ALSO **netmgt_fetch_error**(3N), **netmgt_save_request**(3N), **netmgt_save_argument**(3N)

NOTES Only agents may call this function.

NAME

netmgt_send_error – send an error report

SYNOPSIS

```
bool_t
netmgt_send_error(error)
        Netmgt_error    *error;
```

DESCRIPTION

netmgt_send_error sends an error report to the rendezvous, indicating that an error occurred while handling a request. Two kinds of errors can be sent: generic errors, defined in include file **netmgt_errno.h**, and errors specific to the agent, defined in the agent schema *agentErrors* record.

Agent-specific errors can be warnings or fatal errors. Generic errors are always fatal.

INPUT ARGUMENTS

error pointer to a structure with the following fields:

service_error
generic error code from **Netmgt_stat**, listed in the include file **netmgt_errno.h**. Non-generic (agent-specific) errors should use **NETMGT_WARNING** or **NETMGT_FATAL** and list the agent error code in the *agent_error* parameter.

NETMGT_WARNING indicates an agent-specific non-fatal error. The agent will continue to process the request.

NETMGT_FATAL indicates an agent-specific fatal error. The agent will terminate the request.

agent_error
agent-specific error code corresponding to the *agentErrors* record in the agent schema. For generic errors, set this field to zero.

message optional, null-terminated string containing additional information about the error. Use this field for information that cannot be included in the agent schema, such as the device name involved in the error.

RETURN VALUE

TRUE if successful, otherwise **FALSE.**

DIAGNOSTICS

If FALSE is returned, global variable *netmgt_error* indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N)**.**

FILES

netmgt_errno.h

SEE ALSO

netmgt_fetch_error(3N)

NAME

netmgt_send_report – send a data or event report

SYNOPSIS

```
bool_t
netmgt_send_report(delta_time, status, flags)
        struct timeval  delta_time;
        Netmgt_stat     status;
        u_int           flags;
```

DESCRIPTION

netmgt_send_report sends a data or event report to a rendezvous process. The caller should have previously built a report argument list with calls to **netmgt_build_report**(3N)**.**

INPUT ARGUMENTS

delta_time

relative timestamp of the report. A manager process can use the value of *delta_time* to check the continuity of the agent reports. For each report, if the value of this field is less than the value in the previous report, the manager will know something happened to reset the agent or the managed device.

There is no hard and fast rule about what value should be here. One recommendation is that agents return the difference between the current time and some agent-defined starting time. Some agents might return the amount of time since the managed system has been initialized. Whatever the agent returns should be documented, so that users can interpret the results correctly.

Don't put an absolute timestamp here, as it would not be useful.

status status code **NETMGT_SUCCESS** or **NETMGT_WARNING**. Use **NETMGT_WARNING** when a non-fatal error occurred while collecting the statistics. If **NETMGT_WARNING** is specified, the caller must have previously sent an error report with **netmgt_send_error**(3N)**.**

flags report flag. Use **NETMGT_LAST** if no more reports will be sent to the rendezvous process from this agent; otherwise, send a NULL.

RETURN VALUE

TRUE if successful, otherwise **FALSE.**

DIAGNOSTICS

If FALSE is returned, global variable *netmgt_error* indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N)**.**

SEE ALSO

netmgt_build_report(3N), **netmgt_fetch_error**(3N), **netmgt_send_error**(3N)

NAME

netmgt_set_argument – append an argument to the request

SYNOPSIS

```
bool_t
netmgt_set_argument(arg)
        Netmgt_arg      *arg;
```

DESCRIPTION

netmgt_set_argument specifies optional request arguments to be sent to the agent. *arg* points to the buffer that contains the request arguments.

These optional request arguments are agent-specific, for directing the agent to perform actions not accounted for by the standard request mechanisms. For example, the **na.ping**(8) agent uses them to set packet size, time to wait for echo replies, etc.

INPUT ARGUMENTS

arg pointer to a structure with the following fields:

- *name* argument name. Since **snm**(1) and **snm_cmd**(1) use the argument string NETMGT_OPTSTRING (as defined in **netmgt_arglist.h**) and agents look for that string, other manager applications should use it as well, unless selected managers and agents have agreed to use different (or additional) argument strings.
- *type* argument type code, as defined in **netmgt_arglist.h**. Again, to be compatible with **snm**(1), **snm_cmd**(1) and the supplied agents, you should use NETMGT_STRING.
- *length* length of the argument value, in bytes.
- *value* argument value buffer. The value's internal representation is described by *type*.

RETURN VALUE

TRUE if successful, otherwise **FALSE.**

DIAGNOSTICS

If FALSE is returned, global variable *netmgt_error* indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N)**.**

FILES

netmgt_arglist.h

SEE ALSO

netmgt_fetch_error(3N)

NAME

netmgt_set_attribute – request an attribute in a data report

SYNOPSIS

```
bool_t  netmgt_set_attribute (
        char  *attr_name);
```

DESCRIPTION

netmgt_set_attribute requests an attribute in a data report. *attr_name* is a pointer to a string containing the name of the attribute.

Parameters

attr_name
pointer to a valid attribute name in a group

RETURN VALUE

TRUE if successful, otherwise **FALSE.**

DIAGNOSTICS

If **FALSE** is returned, global variable **netmgt_error** indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N)**.**

FILES

netmgt_arglist.h

SEE ALSO

netmgt_fetch_error(3N)

NAME

netmgt_set_debug – set the debugging level

SYNOPSIS

```
void
netmgt_set_debug(debug_level)
        u_int    debug_level;
```

DESCRIPTION

netmgt_set_debug sets the debugging level for application execution tracing. It is used with the three macros **NETMGT_DBG1, NETMGT_DBG2**, and **NETMGT_DBG3.**

INPUT ARGUMENTS

debug_level

debug level. Four debugging levels are supported. Debugging level zero disables tracing. Level 1 enables minimal tracing. Level 2 enables intermediate tracing. Level 3 enables full tracing.

In order to get debugging output on the screen, the application must set *debug_level* greater than zero.

RETURN VALUE

None.

DIAGNOSTICS

None.

SEE ALSO

netmgt_dbg(3N)

NAME

netmgt_set_instance – initialize a request

SYNOPSIS

```
bool_t
netmgt_set_instance(target, group, key)
        char    *target;
        char    *group;
        char    *key;
```

DESCRIPTION

netmgt_set_instance specifies the *target*, *group*, and optional *key* an agent should use when performing a request.

INPUT ARGUMENTS

target name of the system containing the element whose attributes are to be reported.

group the group name whose attributes values are to be reported.

key value of one or more attributes that uniquely distinguish an instance of the element. This is only required when the group is a table, and you are requesting a subset of the table.

RETURN VALUE

TRUE if successful, otherwise **FALSE.**

DIAGNOSTICS

If FALSE is returned, global variable *netmgt_error* indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N)**.**

SEE ALSO

netmgt_fetch_error(3N)

NAME

netmgt_alloc_manager_id, netmgt_free_manager_id, netmgt_set_manager_id, netmgt_get_manager_id – manipulate the manager application ID

SYNOPSIS

```
u_int   netmgt_alloc_manager_id()

bool_t  netmgt_free_manager_id(manager_id)
u_int   manager_id;

bool_t  netmgt_set_manager_id(manager_id)
u_int   manager_id;

u_int   netmgt_get_manager_id()
```

DESCRIPTION

A manager ID is an unsigned integer associated with a individual manager application. It is transported in SunNet Manager request and report messages and used to distinguish among requests started by a number of manager applications running on the same workstation.

netmgt_alloc_manager_id() allocates a unique ID to the calling manager application. This ID is the inode number of a temporary file in the SunNet Manager directory **/etc/opt/SUNWconn/snm/snm.id** for Solaris 2.x and **/etc/snm/snm.id** for Solaris 1.x. This function may only be called by applications that send data, event, or set requests. Manager applications should call **netmgt_free_manager_id()** to deallocate the manager ID when they are finished.

netmgt_set_manager_id() sets the manager ID field in data, event, and set requests to *manager_id* . This function may only be called by applications that send data, event, or set requests. It should only be called after a call to **netmgt_set_instance**(3N) and before a call to **netmgt_request_data**(3N), **netmgt_request_event**(3N), or **netmgt_request_set**(3N)**.**

netmgt_get_manager_id() returns the value of the manager ID field in data, event, set, trap and error reports. This function may only be called by applications that receive such reports.

netmgt_free_manager_id() deallocates a manager ID previously allocated by a call to **netmgt_alloc_manager_id() .** This function unlinks the temporary file in the SunNet Manager directory **/etc/opt/SUNWconn/snm/snm.id** for Solaris 2.x and **/etc/snm/snm.id** for Solaris 1.x. This function may only be called by applications that send data, event, or set requests.

RETURN VALUES

netmgt_alloc_manager_id() returns a positive integer if successful; otherwise, it returns zero.

netmgt_set_manager_id() and **netmgt_free_manager_id()** return TRUE if successful; otherwise, they return FALSE.

FILES

/etc/opt/SUNWconn/snm/snm.id (Solaris 2.x)

/etc/snm/snm.id (Solaris 1.x)

SEE ALSO

netmgt_kill_request2(3N), **netmgt_set_instance**(3N)

NOTES

These functions only work if the SunNet Manager activity daemon **na.activity** is installed on the local system since it acts as a manager ID server.

netmgt_alloc_manager_id() creates a temporary file in **/etc/opt/SUNWconn/snm/snm.id** on Solaris 2.x and in **/etc/snm/snm.id** on Solaris 1.x for each call. If applications do not call **netmgt_free_manager_id()** to deallocate manager IDs, unused temporary files will accumulate in **/etc/opt/SUNWconn/snm/snm.id** for Solaris 2.x and in **/etc/snm/snm.id** for Solaris 1.x which must then be removed manually.

NAME

netmgt_set_threshold – set the event report threshold for an attribute

SYNOPSIS

```
bool_t
netmgt_set_threshold(thresh)
        Netmgt_thresh *thresh;
```

DESCRIPTION

netmgt_set_threshold sets the threshold associated with an attribute value. *thresh* is a pointer to a structure containing threshold information.

INPUT ARGUMENTS

threshold

pointer to a structure with the following fields:

name attribute name.

type attribute (and threshold) type code, as defined in **netmgt_arglist.h**.

relop relational operator to be used with the threshold to determine if an event has occurred. If the infix expression

<attribute value> <relational operator> <threshold value>

is TRUE, the agent will send an event report to the rendezvous. The following relational operators are defined:

NETMGT_EQ — attribute equal to threshold.

NETMGT_NE — attribute not equal to threshold.

NETMGT_LT — attribute less than threshold.

NETMGT_LE — attribute less than or equal to threshold.

NETMGT_GT — attribute greater than threshold.

NETMGT_GE — attribute greater than or equal to threshold.

NETMGT_CHANGED — attribute has changed (no threshold used).

NETMGT_INCRBY — attribute increased by threshold.

NETMGT_DECRBY — attribute decreased by threshold.

NETMGT_INCRBYMORE — attribute increased by more than threshold.

NETMGT_INCRBYLESS — attribute increased by less than threshold.

NETMGT_DECRBYMORE — attribute decreased by more than threshold.

NETMGT_DECRBYLESS — attribute decreased by less than threshold.

thresh_len

threshold and attribute value length, in bytes. For strings use **strlen(***thresh_val***)**.

thresh_val

pointer to the threshold value buffer. The value's internal representation is described by *type*.

prev_val
pointer to the previous attribute value buffer. This field is used only by the Agent Services library; you do not need to set it.

priority priority level the agent should use when sending an event report. *priority* can be one of three values: NETMGT_LOW_PRIORITY, NETMGT_MEDIUM_PRIORITY, or NETMGT_HIGH_PRIORITY.

RETURN VALUE

TRUE if successful, otherwise **FALSE.**

DIAGNOSTICS

If FALSE is returned, global variable *netmgt_error* indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N)**.**

FILES

netmgt_arglist.h

SEE ALSO

netmgt_fetch_error(3N)

NAME

netmgt_set_value – append a set argument to the request

SYNOPSIS

```
bool_t
netmgt_set_value(setval)
        Netmgt_setval *setval;
```

DESCRIPTION

netmgt_set_value specifies a set request argument to be sent to the agent. *setval* points to the buffer that contains the set request argument. This function should be called once for each attribute to be set.

INPUT ARGUMENTS

setval pointer to a structure with the following fields:

- *group* the group or table containing the attribute to set.
- *key* table key. A key is only required when setting an attribute in one row of a table.
- *name* attribute name.
- *type* argument type code, as defined in **netmgt_arglist.h**.
- *length* length of the argument value, in bytes.
- *value* argument value buffer. The value's internal representation is described by *type*.

RETURN VALUE

TRUE if successful, otherwise **FALSE.**

DIAGNOSTICS

If FALSE is returned, global variable *netmgt_error* indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N)**.**

FILES

netmgt_arglist.h

SEE ALSO

netmgt_request_set(3N), **netmgt_fetch_setval**(3N), **netmgt_fetch_error**(3N)

NAME

netmgt_shutdown_agent – unregister and terminate agent

SYNOPSIS

void
netmgt_shutdown_agent()

DESCRIPTION

netmgt_shutdown_agent unregisters the calling agent from the RPC system, and then shuts it down. It calls **exit**(2) on behalf of the agent with an exit value of zero.

If the agent did not define a *shutdown* routine when it was initialized, this function will be called automatically when the agent needs to be terminated. Otherwise, agents should call this function after they are finished and are about to exit — whether due to normal termination or a fatal error.

INPUT ARGUMENTS

None.

RETURN VALUE

Does not return a value.

DIAGNOSTICS

None.

SEE ALSO

netmgt_init_rpc_agent(3N)

NAME

netmgt_sperror – get a description of an error

SYNOPSIS

char *
netmgt_sperror()

DESCRIPTION

netmgt_sperror returns the error description string associated with the last error contained in the global variable *netmgt_error*.

INPUT ARGUMENTS

None.

RETURN VALUE

pointer to a null-terminated string containing the description of the current error condition.

DIAGNOSTICS

None.

NAME

netmgt_start_agent – start the agent

SYNOPSIS

void
netmgt_start_agent()

DESCRIPTION

netmgt_start_agent allows the agent to receive requests. It suspends the agent to wait for incoming requests, which are passed to the agent's *verify* routine. If the verification is successful, the request is then passed to the agent's *dispatch* routine to dispatch and execute the request.

INPUT ARGUMENTS

None.

RETURN VALUE

Never returns. Control is given back to the agent as callbacks to the *verify*, *dispatch*, *reap*, and *shutdown* routines defined when the agent is initialized.

DIAGNOSTICS

None.

SEE ALSO

netmgt_init_rpc_agent(3N)

NAME

netmgt_start_trap – set context for sending traps

SYNOPSIS

```
bool_t
netmgt_start_trap(system, agent_prog, agent_vers, group, rendez_host,
rendez_prog, rendez_vers, priority, timeout)
        char *system;
        u_int agent_prog;
        u_int agent_vers;
        char *group;
        char *rendez_host;
        u_int rendez_prog;
        u_int rendez_vers;
        u_int priority;
        struct timeval timeout;
```

DESCRIPTION

netmgt_start_trap sets the context for sending one or more trap reports. Agents must call this function once before calling **netmgt_build_report**(3N) and **netmgt_send_report**(3N) to send a trap. Agents should not call this function again to send another trap unless they need to change trap parameters such as *system* or *group.*

INPUT ARGUMENTS

system name of the system associated with the trap.

agent_prog
RPC program number of the agent sending the trap. It may be set to zero if unknown.

agent_vers
RPC program version number of the agent sending the trap. It is set to zero if *agent_prog* has been set to zero.

group group within the agent schema to which the trap attributes belong. It may be set to ‘‘trap’’ if the attributes are not defined in any agent schema.

rendez_host
name of the system where the trap should be sent.

rendez_prog
rendezvous for the trap. This should be set to specify the event dispatcher (**NETMGT_EVENT_PROG**).

rendez_vers
version number of the rendezvous for the trap. This should be set to specify the event dispatcher’s version (**NETMGT_EVENT_VERS**).

priority priority of the trap. The value must be one of **NETMGT_LOW_PRIORITY**, **NETMGT_MEDIUM_PRIORITY**, or **NETMGT_HIGH_PRIORITY**.

timeout maximum time (in seconds) that **netmgt_send_report**(3N) is to wait for a reply from the rendezvous when sending a trap report.

RETURN VALUE

TRUE if successful, otherwise **FALSE.**

DIAGNOSTICS

If FALSE is returned, global variable *netmgt_error* indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N)**.**

SEE ALSO

netmgt_build_report(3N), **netmgt_send_report**(3N)

NAME

netmgt_unregister_callback – unregister a transient RPC callback function

SYNOPSIS

```
bool_t
netmgt_unregister_callback(prog, vers)
        u_long prog;
        u_long vers;
```

DESCRIPTION

netmgt_unregister_callback unregisters the transient callback function with RPC program number *prog* and version number *vers.*

INPUT ARGUMENTS

prog RPC program number of the callback function.

vers RPC version number of the callback function.

RETURN VALUE

Always TRUE.

DIAGNOSTICS

None.

SEE ALSO

netmgt_register_callback(3N)

NAME

netmgt_unregister_rendez – unregister from the event dispatcher

SYNOPSIS

```
bool_t
netmgt_unregister_rendez(event_host, rendez_host, rendez_prog, rendez_vers,
                         agent_prog, event_priority, timeout)
        char     *event_host;
        char     *rendez_host;
        u_long   rendez_prog;
        u_long   rendez_vers;
        u_long   agent_prog;
        u_int    event_priority;
        struct   timeval timeout;
```

DESCRIPTION

netmgt_unregister_rendez asks the event dispatcher on *event_host* to stop sending event reports with priority *event_priority* from agents with RPC program number *agent_prog* to the rendezvous process with RPC program number *rendez_prog* and RPC version number *rendez_vers* on *rendez_host*.

INPUT ARGUMENTS

event_host
: name of the system where the event dispatcher is running.

rendez_host
: name of the system where the rendezvous process is running.

rendez_prog
: RPC program number of the rendezvous process.

rendez_vers
: RPC version number of the rendezvous process.

agent_prog
: RPC program number of the agent who was sending the reports.

event_priority
: event report priority, which can be one of three values: NETMGT_LOW_PRIORITY, NETMGT_MEDIUM_PRIORITY, or NETMGT_HIGH_PRIORITY.

timeout maximum time to wait for confirmation from the event dispatcher.

RETURN VALUE

TRUE if successful, otherwise FALSE.

DIAGNOSTICS

If FALSE is returned, global variable *netmgt_error* indicates the error. The caller can get the reason for the error with **netmgt_fetch_error**(3N)**.**

SEE ALSO

netmgt_fetch_error(3N)

NAME

snm – Site / SunNet / Domain Manager Console

SYNOPSIS

snm [-i] [-q] [-v1] [-v2] *MDB-files ...*

DESCRIPTION

snm is the Site / SunNet / Domain Manager Console. **snm** is used primarily for configuration, performance, and fault management [1]. It enables you to model and visualize an arbitrary collection of network elements and to coordinate agent activity to gather network information.

snm is an an OPEN LOOK™ application [2], [3]. The paradigm is the same as **filemgr**(1) — elements are represented as glyphs in a hierarchical, graphical display. You handle objects through direct manipulation, glyph menu operations, and element properties.

The Console Properties window allows you to define preferences for your instance of the Console.

On-line help is available for all Console windows and tools. For information on using the Console, see the *Site/SunNet/Domain Manager Administration Guide* and the *Site/SunNet/Domain Manager Application and Agent Development Guide.*

OPTIONS

Each option is recognized as a separate argument.

-i Initialize the run-time management database (MDB) from structure and instance files (both implied by the configuration file **/etc/opt/SUNWconn/snm/snm.conf** for Solaris 2.x or **/etc/snm.conf** for Solaris 1.x and explicitly specified on the command line). Otherwise, **snm** will use the run-time database from the last run.

-q Suppress the QuickStart popup.

-v1 Converts an MDB file saved in SNM 1.x format to SNM 2.0 format.

-v2 Denotes an MDB file in 2.0 format. Use this argument only when loading additional files that are in SNM 1.x format. If all the files you are loading are in SNM 2.0 format, you do not need this argument.

MDB-files

A list of structure and instance schema files to be read into the run-time database. If the run-time database is not being initialized, these files are appended to it.

EXAMPLES

The following command:

snm -i -v1 old_file1 old_file2

converts the files *old_file1* and *old_file2* from SNM 1.x format into SNM 2.0 format and loads them into the Console.

The following command:

snm -i -v1 old_file1 old_file2 -v2 new_file1 new_file2

converts the files *old_file1* and *old_file2* from SNM 1.x format into SNM 2.0 format and loads them, along with *new_file1* and *new_file2* , into the Console.

SEE ALSO

[1] *Information Processing Systems Open Systems Interconnection Systems Management Overview*, ISO / DP 10040

[2] *OPEN LOOK Graphical User Interface Style Guide, Release 1.0*

[3] *OPEN LOOK Graphical User Interface Functional Specification, Release 1.0*

NAME

snm.conf – Site/SunNet/Domain Manager configuration file for agents and daemons

DESCRIPTION

/etc/opt/SUNWconn/snm/snm.conf for Solaris 2.x or **/etc/snm.conf** for Solaris 1.x is an ASCII text file that acts as a central place for defining Site/SunNet/Domain Manager agent and daemon resources and operations.

/etc/opt/SUNWconn/snm/snm.conf for Solaris 2.x or **/etc/snm.conf** for Solaris 1.x also specifies per-host security levels for agents and may be used for third-party Site/SunNet/Domain Manager applications.

SYNTAX

Blank lines are allowed. Entries cannot span lines. Comments start with a ''#'' as the first non-white space character on the line and continue through the end of the line.

Entries have the format:

keyword<whitespace>*string_value*

keyword is an identifier defined by the agent or daemon. The agent or daemon looks for *keyword*, skips the white space (TABS and spaces), and uses *string_value* as the associated value. You should not change the *keyword*s because the agent or daemon won't find them.

Each agent has an entry of the form **na.name ,** just like their file names. The Agent Services library uses these entries to set agent security levels when an agent is started.

The interpretation of *string_value* is up to the agent using it. It can be a file, a directory, a list of directories, a number or anything else that will fit on a line. *string_value* may contain embedded white space if the agent supports it.

You can always change the value of *string_value* to satisfy local site requirements. Ensure the value is appropriate for the way the agent uses *keyword*. If writing an agent, ensure that the agent's usage of the keyword's value is well-documented.

ENTRIES

These are the keywords used by Site/SunNet/Domain Manager agents and daemons:

na.snmp.schemas

list of directories where SNMP schema files reside. Separate each directory with a colon.

The keyword's supplied value is **/opt/SUNWconn/snm/agents** for Solaris 2.x and **/usr/snm/agents** for Solaris 1.x.

na.snmp.default-schema

file name where the SNMP proxy agent can find its default schema. The default schema file is used if no schema file is defined in the optional arguments for a request.

This file must exist (and be identical) on both the manager station and the machine where the agent runs. On the manager station, the agent schema is treated like any other; that is, it defines groups, tables, attributes, and so on, for the user interface. The SNMP proxy agent uses the schema to map information returned by the target host into Site/SunNet/Domain Manager attributes.

The keyword's supplied value is **/opt/SUNWconn/snm/agents/snmp-mibII.schema** for Solaris 2.x and **/usr/snm/agents/snmp-mibII.schema** for Solaris 1.x if Site/SunNet/Domain Manager was installed in **/opt/SUNWconn/snm** for Solaris 2.x and **/usr/snm** for Solaris 1.x.

na.snmp.hostfile
the file name where the SNMP proxy agent gets host-specific information for SNMP requests if certain information is not included in the manager request. The file is also where the SNMP trap daemon gets the name of the trap file to be used for specified devices.

The keyword's supplied value is **/var/opt/SUNWconn/snm/snmp.hosts** for Solaris 2.x and **/var/adm/snm/snmp.hosts** for Solaris 1.x.

na.snmp.request_timeout
the number of seconds the SNMP proxy agent will wait for an SNMP response from the target device. If the round trip time between the SNMP proxy agent and the managed devices is longer than 5 seconds (due to a slow link or the amount of traffic), you may want to increase this number. If the packet round trip time is predictably faster than five seconds, you can decrease this number.

The keyword's supplied value is **5.**

na.snmp.max_attempts
the maximum number of Get or Get-Next packets that the SNMP proxy agent will attempt to send to a target device for each reporting interval. If the SNMP proxy agent is on the same subnet as the devices it is managing, you may want to decrease this number to 1. If the SNMP proxy agent is communicating with the devices across a relatively unreliable link, you may want to increase this number.

The keyword's supplied value is **3.**

na.snmp.report_timeout
the number of seconds before the SNMP proxy agent will time out while trying to send a report to an SNM rendezvous.

The keyword's supplied value is **5.**

na.snmp.ack_timeout
the number of seconds before the SNMP proxy agent will time out while waiting for an acknowledgement from a subprocess that the subprocess will perform a new request. If the proxy agent times out, it will create a new subprocess to handle the request. Note that a subprocess may fail to acknowledge a new request if it is blocked while sending a request to the target device or blocked while sending a report to an SNM rendezvous. You should increase this time out value only if you have a large number of subprocesses in relation to the number of requests the proxy agent is performing.

The keyword's supplied value is **15.**

na.snmp.max-subprocs
the maximum number of subprocesses the SNMP proxy agent will fork to handle

new requests.

The keyword's supplied value is **20.**

na.snmp.max-requests
the maximum number of requests that an SNMP proxy agent subprocess will handle.

The keyword's supplied value is **50**.

na.snmp.trap-if-no-response
specifies whether the SNMP proxy agent is to send a 'no response' trap instead of an error report if the target device does not respond to a poll.

The keyword's supplied value is **true.**

na.snmp.exit-if-no-requests
specifies whether the SNMP proxy agent is to exit if it is not performing any requests. The value 'true' minimizes the number of processes running on your system. Set the value to 'false' if you have one or more large SNMP schema files and you find that the SNMP proxy agent is exceedingly slow to start up. Since the proxy agent loads all SNMP schema files when it starts up, setting this keyword value to 'false' removes the overhead of loading the schema files each time the proxy agent restarts.

The keyword's supplied value is **true.**

na.snmp.get-next-force-full-varbindlist
specifies whether the SNMP proxy agent should forcibly send the full variable binding list everytime while performing a SNMP get-next request. It is particularly useful especially when querying tables from target SNMP agents, where different rows of the same table do not support all the attributes as defined in the variable bind list, but only a subset of them. If the value is set to 'false' then all the variables not serviced / returned from the previous get-next request are dropped before performing the next get-next operation. Setting the value to 'true' will force the full variable bind list for each get-next request, even though the previous request might not have returned certain attributes.

The keyword's supplied value is **false.**

na.snmpv2.default-schema
file name of default MIB (in schema format).

The keyword's supplied value is **/opt/SUNWconn/snm/agents/snmpv2-mibII.schema** for Solaris 2.x and **/usr/snm/agents/snmpv2-mibII.schema** for Solaris 1.x if Site / SunNet / Domain Manager was installed in **/opt/SUNWconn/snm** for Solaris 2.x and **/usr/snm** for Solaris 1.x.

na.snmp-trap.default-priority
specifies the default SNMP trap priority. The possible values are: low, medium, high, and discard.

The keyword's supplied value is **low.**

na.snmp-trap.forward.snmp-traps
forwards the SNMP trap PDU to other hosts. Up to two hosts can be specified.

na.snmp-trap.raw
specifies a flag used to indicate whether the trap daemon should also send raw oid/value in the trap report.

The keyword's supplied value is **false.**

na.hostperf.request_timeout
the number of seconds the hostperf proxy agent will wait for an **rstat(8r)** response from the target device.

The keyword's supplied value is **5.**

na.hostperf.report_timeout
the number of seconds before the hostperf proxy agent will time out while trying to send a report to an SNM rendezvous.

The keyword's supplied value is **5.**

na.hostperf.ack_timeout
the number of seconds before the hostperf proxy agent will time out while waiting for an acknowledgement from a subprocess that the subprocess will perform a new request. If the proxy agent times out, it will create a new subprocess to handle the request. Note that a subprocess may fail to acknowledge a new request if it is blocked while sending a request to the target device or blocked while sending a report to an SNM rendezvous. You should increase this time out value only if you have a large number of subprocesses in relation to the number of requests the proxy agent is performing.

The keyword's supplied value is **15.**

na.hostperf.max-subprocs
the maximum number of subprocesses the hostperf proxy agent will fork to handle new requests.

The keyword's supplied value is **20.**

na.hostperf.max-request
the maximum number of requests that a hostperf proxy agent subprocess will handle.

The keyword's supplied value is **50.**

na.hostperf.trap-if-no-response
specifies whether the hostperf proxy agent is to send a 'no response' trap instead of an error report if the target device does not respond to a poll.

The keyword's supplied value is **true.**

na.ping.reach-packets
the maximum number of ICMP ping request packets that are sent to the target device before the device is reported as 'unreachable'.

The keyword's supplied value is **3.**

na.ping.stats-packets
the number of ICMP ping request packets that are sent to the target device to gather statistics for the 'stats' group.

The keyword's supplied value is **5.**

na.ping.request_timeout
the number of seconds the ping proxy agent will wait for a ping response from the target device.

The keyword's supplied value is **1.**

na.ping.report_timeout
the number of seconds before the ping proxy agent will time out while trying to send a report to an SNM rendezvous.

The keyword's supplied value is **5.**

na.ping.ack_timeout
the number of seconds before the ping proxy agent will time out while waiting for an acknowledgement from a subprocess that the subprocess will perform a new request. If the proxy agent times out, it will create a new subprocess to handle the request. Note that a subprocess may fail to acknowledge a new request if it is blocked while sending a request to the target device or blocked while sending a report to an SNM rendezvous. You should increase this time out value only if you have a large number of subprocesses in relation to the number of requests the proxy agent is performing.

The keyword's supplied value is **15.**

na.ping.max-subprocs
the maximum number of subprocesses the ping proxy agent will fork to handle new requests.

The keyword's supplied value is **20.**

na.ping.max-requests
the maximum number of requests that a ping proxy agent subprocess will handle.

The keyword's supplied value is **50**.

na.snmp-trap.default-trapfile
file name where enterprise-specific SNMP traps are defined. See **na.snmp.trapfile**(5).

The keyword's supplied value is **/var/opt/SUNWconn/snm/snmp.traps** for Solaris 2.x and **/var/adm/snm/snmp.traps** for Solaris 1.x.

na.snmp-trap.rendez
list of hosts to which the SNMP trap proxy should send traps. Separate each host with a colon.

This keyword is not supplied in the default file. This causes the SNMP trap proxy to send traps to the local host.

na.snmp-trap.raw
specifies whether the SNMP trap daemon returns raw OIDs and their values without translation for attributes in trap reports, in addition to the translated attribute names and values.

The keyword's supplied value is **false.**

activity-log
file name of the activity log. The activity daemon, **na.activity**, uses this file to maintain a record of the currently active data and event reporting activities started from manager applications on the local machine.

The keyword's supplied value is **/var/opt/SUNWconn/snm/activity.log** for Solaris 2.x and **/var/adm/snm/activity.log** for Solaris 1.x.

event-log
file name of the event log. The event dispatcher, **na.event**, uses this file to log all event reports it receives.

The keyword's supplied value is **/var/opt/SUNWconn/snm/event.log** for Solaris 2.x and **/var/adm/snm/event.log** for Solaris 1.x.

monitor-log
file name the logger, **na.logger**, uses to log all data and event reports it receives.

The keyword's supplied value is **/var/opt/SUNWconn/snm/monitor.log** for Solaris 2.x and **/var/adm/snm/monitor.log** for Solaris 1.x.

request-log
file name of the request log. The Agent Services library uses this file to track the requests running on the local machine with the **restart** option specified. When an agent starts after an abnormal termination, the Agent Service library attempts to restart any requests for the agent that are listed in this file.

The keyword's supplied value is **/var/opt/SUNWconn/snm/request.log** for Solaris 2.x and **/var/adm/snm/request.log** for Solaris 1.x.

na.agentname
agent's security level, from zero (none) to five (highest). There is one entry for each agent.

The keyword's supplied value is **0 (zero — no security)**.

linkmap specifies the location for the link map file. This file is used by Discover and Console for link management.

The keyword's supplied value is **/var/opt/SUNWconn/snm/linkmap** for Solaris 2.x and **/var/adm/snm/linkmap** for Solaris 1.x.

snmdb-directory
specifies the location for the database.

The keyword's supplied value is **/var/opt/SUNWconn/snm** for Solaris 2.x and **/var/adm/snm** for Solaris 1.x.

get-requested-attributes-only
retrieves only the specified attributes. On previous versions of SunNet manager, all the attributes in a group were retrieved by default. If this behaviour is required, set this keyword to false.

The keyword's supplied value is **true.**

WARNING

Keywords are **very** agent-specific. Agents must never assume there is any relationship between keywords, never use a keyword for more than one purpose, and never use another agent's keywords.

SEE ALSO

na.snmp.hostfile(5), **na.snmp**(8)

NOTES

Site / SunNet / Domain Manager does not currently provide any API functions to retrieve or validate the contents of this file; it must be done by the agent.

NAME

snm.logfile – format of Site / SunNet / Domain Manager log files

DESCRIPTION

All the programs in the Site / SunNet / Domain Manager package write log files in a standard format. The format is designed to be easy to parse, and written in ASCII, so that editors and shell scripts can operate on it.

This is the format read by the results browser, **snm_br**(1). The log file format is written by:

- The event dispatcher (**na.event**(8)) and report logger (**na.logger**(8)).
- The *Filename* option of the Console.
- The *Save* button of the Console's Data & Event report windows.
- The Console's Send-to-program option.
- Folders saved by the Results Browser.

SYNTAX

If the first column of a line contains a **#** the line is considered a comment, and the rest of the line is ignored. The first line of the log file is **#REPORTLOG-***N*, where *N* is the version of the file. SunNet Manager 1.0 log files, which did not support comments, are assumed to be version zero. SunNet Manager 1.1 and 1.2 log files are version 1.

Each entry is a newline-terminated ASCII string. Fields in the entry are separated by white space. Null strings are represented by a field containing two double quote characters. String values that contain embedded white space (tabs, spaces and newlines) or double quotes are surrounded by double quotes. A double quote is represented by two successive double quote characters, and the field that contains these characters is surrounded by double quotes. Therefore, a field of four double quotes corresponds to a single double quote, as originally returned by the agent.

The layout of the fields is:

1. *Report Type* — one character:
   - A action report
   - D data report
   - E event report
   - K killed (deleted) report
   - R error report
   - S set report
   - T trap report
   - U user-defined report
   - ? unknown report type
2. *Reserved* — a text string. Reserved for future use.
3. *Agent IP address* — machine where agent is running, in Internet dot notation, or 0.0.0.0 if no agent.
4. *Agent RPC program number* — 0 if the report was not generated by an agent.
5. *Manager IP address* — where request came from (in Internet dot notation)
6. *Request timestamp* (seconds since UNIX epoch)
7. *Request timestamp* (microseconds)

8. *Time received* by writer of the report (seconds since UNIX epoch)
9. *Time sent* by agent (seconds since UNIX epoch)
10. *Delta time* returned by agent (seconds field only)
11. *Error code* (0 for success. See **netmgt_errno.h**)
12. *Flag*:
    - 0 more reports to come
    - 64 last report
13. *Target system name* or ‘’’ if none given
14. *Group name* or ‘’’ if none given
15. *Key name* or ‘’’ if none given
16. *Attribute List*: (there can be more than one of these)
    1. *Attribute name*
    2. *Attribute type*
       - A address (IP dot notation)
       - C counter
       - E enumeration (stored as an integer)
       - F floating point, possibly in scientific notation
       - G gauge
       - H hexadecimal number
       - I integer
       - O object identifier (in dot notation)
       - S character string
       - T timeticks
       - U UNIX timestamp (seconds)
       - X octet string
    3. *Attribute value*

       For event reports, these additional fields are given if an event has occurred for this attribute:
    4. *Relational operator*:
       - == equal to
       - != not equal to
       - > greater than
       - >= greater than or equal to
       - < less than
       - <= less than or equal to
       - +- value changed
       - += increased by
       - +> increased by more than
       - +< increased by less than
       - -= decreased by
       - -> decreased by more than
       - -< decreased by less than
    5. *Threshold value*
    6. *Event priority*:
       - L low priority

M medium priority
H high priority

Error reports look like regular reports with special attribute names. Three attribute *name/type/value* triples are given:

1. attribute name ‘‘netmgt_service_error’’
2. ‘‘I’’
3. service_error value
4. attribute name ‘‘netmgt_agent_error’’
5. ‘‘I’’
6. agent_error value
7. attribute name ‘‘netmgt_error_message’’
8. ‘‘S’’
9. error message string

An additional field may be present to mark the end of a table row — ‘‘netmgt_endofrow’’. This field separates rows of a table in a report. It may or may not appear as the last field in an entry (some agents may not send an end-of-row marker for the last table entry in the report).

NOTES

Some SNM agents return table keys with the pseudo-attribute ‘‘netmgt_table_key’’. If the SNM Console or Results Browser see this pseudo-attribute, they convert it into ‘‘KEY’’ for presentation to the user.

The SNMP proxy agent returns a pseudo-attribute ‘‘netmgt_schema.’’ The pseudottribute indicates the name of the SNMP schema (**Note**: not the name of the file) that should be used for processing data and event reports — converting enumerations to text strings, for example. If the SNM Console or Results Browser see this pseudo-attribute, they do not display it. The pseudo-attribute is not returned in traps.

For error reports, the ‘‘netmgt_agent_error’’ value is returned as an integer. The only way to find out what the value means is to refer to the agent schema file.

Determining what the enumeration values mean requires the agent schema.

Existing utilities that depend on the version 0 format can use **snm_cvtlog**(1) to convert the version 1 log files to the old format.

NAME

snm_br – Site/SunNet/Domain Manager results browser

SYNOPSIS

snm_br [–b] [files...]

DESCRIPTION

snm_br is an OPEN LOOK application that processes log files generated by various components of the Site/SunNet/Domain Manager package. These files usually contain reports from agents. The program provides a way to scan log files quickly, and to optionally send portions of the logs to **snm_gr**(1)**,** the Site/SunNet/Domain Manager grapher.

All files read by **snm_br** must conform to the file format specified in **snm.logfile**(5). Any files specified on the command line are loaded on startup. Once the program is running, load files with the File/Load menu item.

Folders:

Folders are places to collect similar (or disjoint, if you prefer) report streams. You might have folders for reports about individual systems, critical gateways, or whatever.

Use the File/Load menu item to load data/event logs into the ''current'' folder. If you load multiple files into a folder, duplicate reports are ignored. Use the File/Save menu item to save the contents of the current folder to the named file.

Folders exist only while the program is running. If you restart the program, you'll only have one folder — the Main folder — available. This is different from **mailtool**(1), where folders are really files. In **snm_br**, folders are temporary organizational bins to simplify graphing and printing.

Folder Operations: Accessed via the Edit button and Folder menu

Change folders with the ''Folder'' menu button. Rename the current folder by entering the new name in the text item at the top of the window and typing the Return key. You can't rename the ''Main'' folder.

Create a new folder by selecting Edit–>New Folder. You're switched to a folder called ''New Folder,'' which you can rename.

Edit–>Empty Folder removes all reports from the current folder. Edit–>Delete Folder empties the folder, and then removes it from the Folder selection list. You are switched to the ''Main'' folder after the delete operation. Deleting the ''Main'' folder empties it but doesn't delete it.

Streams:

Streams are logical groupings of reports from agents. Each stream corresponds to a manager request (identified by the manager's address and the request timestamp). Error reports from a request are split off into separate streams.

The top window (just under the buttons) is the Streams window. Each stream has a summary line with the format:

system agent group type date #reports

where ''system'' is the name of the target system (not necessarily the name of the system where the agent was running). ''type'' is the report type -- Data, Event, etc. ''date'' is the time of the *original request* from the manager. The year is not shown. ''#reports'' is the total number of reports in the stream.

Clicking the SELECT button on a summary line selects the stream so that you can operate on it (print, delete, etc). Select multiple streams with the SELECT button. Double-clicking the SELECT button on a summary line makes that stream the ''current'' stream. The stream's individual reports appear in the lower window (called the report window). Double-clicking clears any existing selections. Selected streams have a highlighted box around them, and the ''current'' stream is additionally highlighted with a bold font.

The list is sorted by system name, agent name, and date.

Stream Operations: Accessed via the Streams Menu

The streams menu is available by pressing the MENU button while the mouse pointer is in the streams window.

You can select all streams that match certain properties: system name, agent/group name, and report type. Pull right on the ''Select'' item to get a list of the possible choices based on the reports currently in the folder.

All the other menu items in the stream menu operate on the streams that have been selected. You can copy the streams to another folder, print the streams and delete them.

Sending streams to snm_gr

You can send individual attributes (of type float and int) to **snm_gr**, Site/SunNet/Domain Manager Grapher. Using the streams menu, pull right on the ''Send'' item, through ''Graph'' to see a list of attributes that can be graphed. The list is based on the *first* selected stream in the streams window. All the reports in the selected streams that contain the attribute are sent to the Grapher.

Select the ''Choices...'' item to graph more than one attribute. A window appears where you select a number of attributes. Select the Apply button to send the data to the grapher.

If you want to graph all the attributes in the menu list, select the ''All Attributes'' menu item at the bottom of the menu.

Reports:

Reports are individual agent reports; displayed in the lower window (known as the report window). You ''move'' though the report stream, viewing individual reports, using the slider just above the report window. If there is no current stream, or the current stream has only one report, the slider doesn't appear. For attributes that return elapsed time (like the "uptime" attribute returned by the hostperf data group), the time is displayed in the format "<days>d <hours>:<minutes>:<seconds>.<subseconds>". Note that the same

attribute value is displayed by the Console in the format "<hours>:<minutes>:<seconds>.<subseconds>".

Report Operations: Accessed via the Reports menu

The reports menu is available by pressing the MENU button while the mouse pointer is in the reports window.

The ''Clone'' item makes a copy of the current report in a new window, enabling you to compare two or more reports. To remove the window, pull out the window's pushpin or select the Dismiss option from the window's menu.

The ''Delete'' option removes the current report from the report stream. This can be handy when you've got an abnormality in the stream, and the graph isn't scaling correctly because of it. If you delete the last report in a stream, the stream summary line is removed from the streams window.

The ''Print'' submenu allows you to print the current report, print all the reports in the current stream starting from the current report, and print the entire stream from the first report. This last option is the same as using the print option in the streams menu when there is only one stream selected.

OPTIONS

–b When starting the Grapher (snm_gr) automatically, this option instructs the grapher to create graphs with a white background. Please refer to the snm_gr man page for more information.

CUSTOMIZATION

On startup, the tool looks for the the following X resources:

OpenWindows.MultiClickTimeout
Two clicks within this interval (expressed in tenths of a second) indicates a mouse button double-click, rather than two single clicks. Default is 4.

SNM.Br.WPosX The initial window position, in pixels from the left of the display. Default is 300.

SNM.Br.WPosY The initial window position, in pixels from the top of the display. Default is 100.

SNM.Br.WSizeX The initial window width, in pixels. Default is 512.

SNM.Br.WSizeY The initial window height, in pixels. Default is 640.

SNM.Br.Print This command line string is passed the selected report streams when a Print menu item is selected. Default is **lp .**

SNM.Br.DisplayParams
If True, the system / agent / group is displayed in the report. Default is True.

SNM.Br.DisplayFrom If True, the time the request was made, as well as the name of the host making the request, is displayed. Default is True.

SNM.Br.DisplaySent If True, the time the report was sent by the agent, as well as the name of the host where the agent was running, is displayed.

Default is True.

SNM.Br.DisplayReceived
If True, the time the entry was logged is displayed. Default is True.

SNM.Br.DisplayMisc If True, Delta time, Error code, and report flags, as reported by the agent, are displayed. Default is True.

SNM.Br.DisplayAttr If True, attributes and values are displayed. Default is True.

SNM.Br.FormatRight If True, attribute names are right-justified; otherwise they are left-justified. Default is True.

Tool Properties: Accessed via the Edit menu

The Edit menu button entry ''Tool Properties...'' brings up a window where you can customize the program. Except for the multi-click timeout, each item in the window corresponds to an entry in the X resource database described above. The multi-click timeout can be set from the OpenWindows properties tool.

Use the Apply button to accept these changes. Use the Save button to accept these changes and write them out to the **~/.Xdefaults** file. Use the Load button to load the resources currently held by *the X resource manager* (note that it does not load the **~/.Xdefaults** file). Use the Reset button to reset the properties to the state of the window at the last Apply or Save.

The Print Option and Report Format properties take effect as soon as you select Apply or Save. None of the others take effect until you Save them, reload the **~/.Xdefaults** file into the X resource manager (see the **xrdb**(1) man page) and restart **snm_br**.

FILES

~/.Xdefaults
~/.brmenu

SEE ALSO

xrdb(1), **snm.logfile**(5)
Site/SunNet/Domain Manager Application and Agent Development Guide

NOTES

snm_br was designed to handle many agent reports. Because report logs may be extremely large, only pointers to individual reports are kept in memory. This means every time you load a file, the file is kept open so that reports can be read when you ask for them. You should **not** save a folder out to a file from which you've already loaded reports. This will destroy the existing file, and the program's pointers into the file will be no good.

There is no ''Undo.'' However, since the program operates on *pointers* to the data rather than the data itself (except for the File Save option), deleting reports doesn't do any permanent harm. You can always re-load the files.

Because the program keeps open every file it ever read, it eventually may run out of file descriptors. In that case, no more files can be loaded. Restarting the program helps, but you'll lose the organization of all the data you've loaded. Instead, you can start another **snm_br**.

NAME

snm_cmd – Site / SunNet / Domain Manager command-line manager application

SYNOPSIS

snm_cmd [-d] { -a *agent* | -f *schema* } -g *group* [*options ...*]
snm_cmd -e *threshold* -f *schema* -g *group* -n *attribute* -r *relation* [*options ...*]
snm_cmd -S *set_value* -f *schema* -g *group* -n *attribute* [*options ...*]
snm_cmd -v -f *schema* [-h *host*] [-t *target*]

DESCRIPTION

snm_cmd is a command-line manager for Site / SunNet / Domain Manager agents. It provides a way to send requests to agents without having to run the SNM Console or write dedicated test programs.

snm_cmd can be used to validate an agent schema by verifying the names and types of attributes in the agent schema against those returned by the agent.

In addition to testing agents, **snm_cmd** can be used with **at**(1) or **cron**(1M) to start agents at a particular time for a given duration and to log the information at a designated rendezvous.

You can use **snm_cmd** from a terminal or from a machine not running the SNM Console, although the SNM Console support environment (logger, agent schema files, run-time database, event dispatcher, etc) must be available. The SNM Console has no knowledge of agents started by **snm_cmd**.

OPTIONS

Each option is recognized as a separate argument.

–**a** *agent*
: name or RPC program number of the agent to send the request to, as it appears in the **rpc.bynumber** NIS map or **rpc** file. This option is only used for data reports if the **-f** option is not specified. If it is specified without the –**f** option and a schema file is needed, the program will look for the schema file that corresponds with the named agent. See the discussion of the –**f** option to understand the search for the schema file.

–**c** *count*
: maximum reports to generate. A value of zero asks the agent to report forever. The default is 1 unless the **-i** option is specified, in which case the default is zero.

–**d**
: start data reporting. This is the default unless **-e**, **-S** or **-v** is specified.

–**e** *threshold*
: start event reporting and set the threshold to *threshold*. Threshold is ignored if the relational operator given with the **-r** option is ‘‘CHANGED’’, but it is still required. The **-n** and **-r** options are also required to request an event report. **snm_cmd** will convert *threshold* to the correct data type based on the type of the attribute given with the **-n** option. You need to specify a key (use the –**k** option) if you want to do event reporting on tables.

–**E**
: print enumerations as text rather than integers.

–**f** *filename*
: name of the agent schema file. This is required for event reporting, setting,

graphing or validation requests—everything but data requests. If the name is not an absolute path name, or the **–f** option is not given but the **–a** option is, **snm_cmd** will try to look up the agent schema file based on the agent name. It first looks in the current directory for a file called *agent*.schema. If not found, it then uses the **SNMHOME** environment variable, if available; otherwise **snm_cmd** looks in **/opt/SUNWconn/snm** for Solaris 2.x and **/usr/snm** for Solaris 1.x. The following two subdirectories are searched: the **agents** subdirectory, and then the **schemas** subdirectory. (This won't work if the name given with the **–a** option is an RPC program number rather than an agent name.)

–g *group*
a named collection of attributes the agent treats as a unit. The group names are defined in the agent schema. Specify the special group **oids** when asking the SNMP proxy agent for attributes via object IDs.

–G send the named attributes to the SNM Results Grapher. Specify the attributes to graph with the **–n** option. You can specify more than one **–n** option when you are graphing. **snm_cmd** will start the grapher if it is not already running. When the request finishes, **snm_cmd** will exit, but the graph will stay around until you remove it. You need to specify a key (use the **–k** option) if you want to do graphing on tables.

–h *host* hostname where the agent is running. This is the system where the request will be sent (and performed), as opposed to the *target* device (see the **–t** discussion). The default is the local host.

–i *interval*
minimum time (in seconds) between each report. Default is zero, which asks the agent to select an appropriate default.

–k *key* identification of a particular row of a table. Default is NULL (no key). Only one key can be specified per request.

–m *recipient*
send mail to *recipient* when an event occurs. If this option is not given, no mail is sent.

–n *attribute*
attribute name, as defined in the named group in the agent schema file. For event reports, this is the attribute associated with the given threshold. For graphing or specifying SNMP object IDs, you can specify multiple -n options (up to 30), each with a single attribute. Specify the attribute in standard SNMP dot notation.

–o *string*
optional agent-specific information, passed as a string. The interpretation of this string (if any) is up to the agent.

–O have the agent stop after the first event report (''send Once'').

–p priority (HIGH, MEDIUM, or LOW) of the event.

–P *rpc#* RPC program number to use as a rendezvous, rather than the RPC number for **na.logger.**

–**r** *operator*
relational operator applied to the event reporting threshold. Operators don't always apply; for example, DECRBYLESS on a string is not supported. Case is not significant. Relational operators may be specified as text or symbols, which probably need to be escaped when specified from the shell. Only one operator is possible per request.

==	EQ	or =
!=	NE	
>	GT	
<	LT	
>=	GE	
<=	LE	
+-	CHANGED	or -+
+=	INCRBY	or +
-=	DECRBY	or -
+<	INCRBYMORE	
+>	INCRBYLESS	
->	DECRBYMORE	
-<	DECRBYLESS	

–**R** *rendez_host*
host name running the rendezvous process. Default is the local host.

–**s** send the reports to standard output. Without this option, reports from the agent are sent to the logger on the host specified with the **-R** option. Reports are formatted for human display only. The output cannot be read by the Results Browser.

–**S** *set_value*
the value to set the attribute specified by the **-g**, **-n**, and (possibly) **-k** flags. Note that you can only set one attribute per request.

–**t** *device*
name of the target device. While the request will be sent to the agent on the system given by the –**h** option, the agent there will be instructed to perform the requested activity on the target device rather than the host where it is running. This option is only useful for proxy agents, since native agents can only get information about the systems where they are running.

–**T** *timeout*
maximum time to wait for a confirmation from the agent, in seconds. The default is the value of NETMGT_TIMEOUT, defined in **netmgt_define.h**.

–**v** send a data report request to every group the agent handles, to verify the agent schema attributes and data types against those returned by the agent. Any discrepancies are reported.

–**V** *version*
protocol version of the agent. Use **10** for the current release, which is also the default.

−x attempt to restart the request if the agent stops from some external cause.

EXAMPLES

1. Start *ping* data reporting on host *delta*, returning a total of 10 sets of reachability group statistics for host *omega*, one every ten minutes, to the terminal:
   snm_cmd -h delta -a ping -g reach -t omega -c 10 -i 600 -s
2. Ask the *rpcnfs* agent to send an event report every time the server *tweedle* gets another bad NFS call. The check happens every five minutes until the agent is later killed:
   snm_cmd -f rpcnfs -g server -n nfs_bad -r changed -e 1 -h tweedle -i 300
3. Ask the *SNMP proxy* agent residing on the default host (localhost) to set the administrative status of interface 3 on router *ouch* to 2 (down):
   snm_cmd -f snmp -g ifStatus -k 3 -n ifAdminStatus -t ouch -S 2
4. Verify the *hostperf* proxy agent schema against the agent running on host *delta*:
   snm_cmd -v -f /opt/SUNWconn/snm/agents/hostperf.schema -h delta (Solaris 2.x)
   snm_cmd -v -f /usr/snm/agents/hostperf.schema -h delta (Solaris 1.x)
5. Get only the ifInUcastPkts for *jupiter*'s first two interfaces by raw object ID:
   snm_cmd -f snmp -g oids -t jupiter -s -n "1.3.6.1.2.1.2.2.1.11.1" -n "1.3.6.1.2.1.2.2.1.11.2"
6. Graph the average round trip time between *zigzag* and *exodus* every 10 seconds:
   snm_cmd -G -f ping -g stats -n tripavg -h zigzag -t exodus -i 10

FILES

netmgt_define.h

ERRORS

Attribute type does not match: group '(group)', attribute '(attr)': schema type '(type1)', returned type '(type2)'
The agent and schema disagree on the attribute's data type.

Cannot get agent ID : (why)
A call to the agent to get its ID failed. The reason is given.

Cannot request data report for group '(groupname)'
(Reason)
The request to the agent failed. The next line explains why.

Could not open schema file : (filename)
The listed file could not be opened for reading. (Fatal error)

Detected n errors between schema file and agent '(agentname)' on '(hostname)'
This is a summary of the number of errors found.

Duplicate attribute name (attrname)
The named attribute has been previously defined in this group/table. (Fatal error)

Duplicate definition of group (groupname)

The named group/table has been previously defined in this agent schema. (Fatal error)

Duplicate error code = n
The agenterror number listed has been previously defined in this agent schema. (Fatal error)

No groups in this agent?
There were no groups or tables listed in the schema file.

Note - attribute is 'writeonly': group '(group)', attribute '(attr)'
Since the named attribute is write-only, the agent doesn't return it, so it cannot be checked with snm_cmd -v.

Schema serial number mismatch: agent wants n, schema is m. No attribute validation done.
The agent was initialized with a particular serial number (hard coded in the agent). The agent schema serial number (given by the 'serial' keyword) should match, otherwise the agent and schema are out of sync. If no 'serial' keyword is given, serial 1 is assumed. This is a fatal error; snm_cmd will not continue.

These attribute(s) not returned by agent '(agentname)' group '(group)'
attribute '(attr1)' (type (type1))
attribute '(attr2)' (type (type2)) . . .
The given list of attributes was named in the agent schema but the agent didn't return any of them.

Two agent definitions in schema file
Only one agent definition is allowed per schema file. (Fatal error)

Undefined attribute '(attr)' (type (type)) returned by agent
An attribute with the given name and type was returned by the agent but was not listed in the agent schema.

(file): parsing error n

(file): n errors encountered.

SEE ALSO *Site/SunNet/Domain Manager Application and Agent Development Guide*

NAME

snm_cmdtool – execute a cmdtool that exec's a command line

SYNOPSIS

snm_cmdtool [command_line]

DESCRIPTION

snm_cmdtool uses **snm_exec**(1) to start a command running in a new **cmdtool(1)**. If the environment variable SNMCMDTOOL is set, its value is used to determine which cmdtool to start. If SNMCMDTOOL is undefined, then **snm_cmdtool** attempts to find a cmdtool in the OpenWindows product tree, usually **/usr/openwin** or the value of OPENWINHOME. If a cmdtool cannot be found, a **shelltool(1)** is started. If **snm_cmdtool** cannot find either of these, a message is printed in the window where **snm_cmdtool** was started, indicating that SNMCMDTOOL must be set.

SEE ALSO

snm_exec(1)**, cmdtool(1)**

NAME

snm_cvtlog – convert log to old format

SYNOPSIS

snm_cvtlog

DESCRIPTION

snm_cvtlog is a filter that converts a log file of the format described in **snm.logfile**(5) into the format used by the Console in the 1.0 release of SunNet Manager.

OPTIONS

snm_cvtlog has no options.

USAGE

snm_cvtlog takes its input from standard input and writes its output to standard output

NAME

snm_discover – Site / SunNet / Domain Manager network element discoverer / monitor

SYNOPSIS

snm_discover [options]

DESCRIPTION

snm_discover dynamically creates Site / SunNet / Domain Manager runtime database records for devices it finds on a network. When an element is discovered, some analysis is done on it to determine its host name, type, and capabilities. Then it is added to the database.

snm_discover allows you to perform a "targeted" search, looking for machines of a certain class, such as routers or machines running SNMP. See the **-d** option below for details.

snm_discover has, in addition to its discover function, a monitor function. This function compares the elements stored in the runtime database with the elements it finds at a specified interval or specified time. If new elements are detected, the monitor function stores the elements in a "holding area" view and records these elements in a log file. Through the same log file, the monitor function can also notify you if a previously discovered host is down or was down within a given period.

The hosts found by the monitor function are then added to a network since the last running of the discover function and hosts that the discover function did not find.

snm_discover has a command line version, described in this man page, and "window" version, invoked from the Site / SunNet / Domain Manager Console's Tool menu. The functions of the two versions are identical. The window version is described in the *Site/SunNet/Domain Manager Administration Guide.*

Before **snm_discover** can be started, the runtime database must be available. The easiest way to do this is to run the Console, **snm**. When **snm** is started for the first time, you have the choice of a "Head Start" option, in which Discover uses the *arp* table to find no more than 10 elements on the local subnetwork, or "Basic Start", in which no discovery is performed.

Once a record is added to the database, the record is never changed by **snm_discover**. The tool's monitor function can, however, detect the change, allowing you to revise your database through the Console's graphical editor function.

OPTIONS

The **snm_discover** program offers a rich set of configuration options for both the discover and monitor functions. These options allow you to fine-tune the extent and depth of the tool's search and monitoring activities. Most of the options determine the operation of the discover function of **snm_discover**. The ICMP- and SNMP-related discover parameters also apply to the monitor function.

In addition to use of the options described here, **snm_discover** uses a configuration file, **discover.conf ,** that allows you to perform even more specialized configuration than is provided by the options described here. See **discover.conf**(5) for details.

Each option is recognized as a separate argument.

–a Creates networks and subnets with IP addresses as names.

–c <*community*>
Use <*community*> for the SNMP community string. The discover function limits its search for SNMP devices to devices that have the community name(s) you specify here. You can specify up to five SNMP community names. Use a colon to delimit multiple names. Note that using multiple community names makes the discover function take a longer time than it would with a single name. This is because the discover function tries each device with each community name until a response is received.

–d <*objects*>
Where <*objects*> is one or a combination of **snmp**, **snm**, **router**, and **network**. A given object must be an SNMP device, a machine running SNM agents, a router, or a network. Separate multiple objects with colons. By default, **snm_discover** finds all types of objects. Routers are defined as systems that have multiple network interfaces to different IP networks. When the discover function finds a router, it fills in IP addresses for the first two interfaces and uses the alias feature to give a name to each interface (if, indeed, each interface has a unique name), including those beyond the first two. You can then specify any of these aliases when sending a data or event request. If you specify **network ,** the discover function finds networks and subnetworks, If you specify **network ,** in combination with the **-l** option, described below, the discover function manageable connections and routers, in addition to networks and subnetworks.

–F <*num*>
Specifies the number of fast ping retries. This is used for SNMP requests when the user selects "Fast Ping Search" method for discovery.

–f <*secs*>
Specifies the fast ping timeout in seconds. This is the timeout value discover waits between ping requests sent in by the "Fast Ping Search" method.

–h <*hops*>
An integer specifying the maximum number of hops the discover function will make (with the Console machine as the starting point). For a given route from the machine running **snm_discover**, this parameter specifies the maximum number of routers the packets sent by the discover function will traverse. The default for this parameter is zero hops, which means that the discover function limits its searches to the local subnetwork.

Setting the hops option to a value above zero can result in very long searches. For example, on a moderately burdened machine, using the default of 0 hops and a local subnetwork with 200 hosts, the discover function would take about 20 minutes to complete.

-l Add link information. When you use this option, the discover function creates manageable connections between discovered elements and adds the connection

information to the **/var/opt/SUNWconn/snm/linkmap** file for Solaris 2.x and to the **/var/adm/snm/linkmap** file for Solaris 1.x.

–m <*netmask*>
Specifies the netmask used for the discover function. By default, the discover function uses the netmask number used on the local subnetwork. Enter the netmask number in hexadecimal (preceded by 0x) or dotted decimal notation (for example, 255.255.255.0).

The netmask parameter has two uses: one is for situations in which you want discovery to begin in a subnetwork other than the local subnetwork. In this case, the discover functions needs to know the netmask used on that subnetwork. The second use is as a limit to the number of hosts the discover function will add to the runtime database. While the latter use can be helpful in certain networks, most users need not be concerned with setting limits. If the tool is about to ping more than 1024 elements, the discover function informs you of the number and displays an estimate of the time it will take to add all of those hosts to the runtime database. The discover function will ping a maximum of 2048 network elements per subnetwork.

For limiting the number of elements found, the discover function uses the inverse of the host portion of the subnet mask. For example, if you have a netmask of 0xffffff00, the host portion of the netmask, 0x00, becomes 0xff, or decimal 255.

-M Starts the monitor function, using the default or current configuration options.

–N <*num*>
Specifies the number of fast pings. This is the number of ping requests sent between timeouts. The requested number of ping requests are sent simultaneously. The default value is 1.

–n <*network*>
Limits the discover function to the specified network. By default, the discover uses the network number used by the local machine. When specifying a network number, be sure to enter a value for each byte within the number. For example, specify 129.144.41.0, not 129.144.41. If you enter a network name or number outside of the local subnetwork, the discover function limits its search to the subnetwork you specify.

-o Add to only one view, rather than to all views in the runtime database, which is the default behavior. The default behavior of **snm_discover** is to add to all views in the runtime database and not preserve the uniqueness of an element as it appears in one view or another. If you specify this option, the discover function adds discovered elements to all views within the runtime database *and* appends the view name to all views (that is, elements of category view) within the specified view. For an element that is in multiple, high-level views, this feature allows you to distinguish that element as it exists in different views.

–p <*num*>
Specifies the number of ICMP retries. The discover function uses the Internet

Control Message Protocol (ICMP) to find network elements that are not SNMP devices. This parameter determines the number of times the tool will retry "finding" an element using ICMP following an initial non-response. If your network is very busy, you might want to increase this value.

-q Perform a quick discovery: 10 or fewer nodes on the local subnetwork. Used in the HeadStart option in the Quick Start window. This option uses the **arp** table to find a small number of hosts.

-r <*IPAddress1:IPAddress2*>
A range of addresses for discover to ping. By default, discover attempts to probe the first 2048 hosts on a network. This can be a problem for networks that are unsubnetted class B addresses, or Class A and Class B addresses with more than 12 bits in the host portion of the address. This option takes a colon separated pair of IP addresses as the first and last addresses in the range that should be probed.

-s <*seconds*>
A timeout used when sending SNMP requests. Determines how long (in seconds) the discover function waits for a response to an SNMP packet before giving up or, for routers, sending another SNMP packet.

-S <*num*>
Number of SNMP retries when sending SNMP requests to routers. For SNMP devices, the discover function uses SNMP for discovery purposes. This parameter determines the number of times the tool will retry contacting an element using SNMP following an initial response. This parameter comes into play primarily when querying routers. ARP and routing tables are often large and sometimes require multiple retries to obtain complete tables.

-t <*seconds*>
Time between sending ICMP packets. Determines how long (in seconds) the discover function waits after sending an ICMP packet that did not receive a response, before sending another ICMP packet. To speed up the discovery process, reduce this value.

-T Bring up the Discover Tool window interface.

-v Use verbose mode. Prints analysis information to the standard error file (usually the shell where the program was started).

–**V** *viewname*
Put all discovered devices in the specified view. If *viewname* doesn't exist, it is created as a *view.subnet* in the **Home** view. If you do not specify the **-o** option (see above), the *viewname* you specify here is the only view for which the discover function preserves the uniqueness of discovered elements.

-x Do not add coordinates to elements. By default, the discover function performs minimal layout of elements within a view. Specify **-x** to have the discover function perform no layout of elements.

-P Use the PING search method for discovery. If this option is specified with the

options -N and -n, then the "Fast Ping Search" method is used.

-A Use the ARP search method for discovery.

–**G** *filename*
Specifies a list of gateway names for discovery. Discover uses the gateways listed in the file <*filename*> for discovery.

–**D** <*level*>
This option is used to print debug messages. A value of 1 is for high level messages and a value of 2 is for detailed level messages.

-D config
This option is used to print out configuration information from the *discover.conf* file.

–**D** <*snmp | traceroute hostname*>
This option is used to test the SNMP or traceroute information on the specified *host*.

If environment variable **SNMDBDIR** is set, it is used as the location of the runtime database file. Otherwise, **/var/opt/SUNWconn/snm** on Solaris 2.x or **/var/adm/snm** on Solaris 1.x is used.

NOTES

If you start the Console with the *-i* flag after running **snm_discover**, you will reinitialize the runtime database. You will lose the results of **snm_discover** unless you first save the database to an ASCII file.

SEE ALSO

discover.conf(5) **linkmap**(5)

NAME

snm_error – error code for snmdb routines

DESCRIPTION

snm_error is an external variable containing the error reason for the last error. It should not be used if the return code of **snmdb** routine is TRUE.

ERROR CODES

SNMDB_BAD_ARGUMENT 1 "Invalid argument"
An invalid argument was specified.

SNMDB_NOT_INITIALIZED 2 "Database not initialized"
An attempt was made to open an uninitialized database. The console must be started first with the **-i** switch to initialize the database.

SNMDB_UNKNOWN_AGENT 3 "Unknown agent name"
The agent name specified is not a known agent.

SNMDB_NO_MEMORY 4 "Out of memory"
The operation required memory to be allocated, but no additional memory was available.

SNMDB_INVALID_TYPE 5 "Invalid type"
The record name specified is not a known record type.

SNMDB_TOO_LONG 6 "Record too long"
The record being built is too long.

SNMDB_ELEMENT_NOT_FOUND 7 "Element not found"
The element specified was not found in the database.

SNMDB_NO_KEY_FIELD 8 "No key"
The record being written to the database does not contain a key field.

SNMDB_DUPLICATE_ID 9 "Duplicate ID"
The record being added to the database has the same ID as an existing record.

SNMDB_DUPLICATE_NAME 10 "Duplicate name"
The record being added to the database has the same name as an existing record.

SNMDB_CONNECTED_TO_ELEMENT_NOT_FOUND 11 "Connected to element not found"
The element to which current element is connected could not be found in the database.

SNMDB_CONNECTION_EXISTS 12 "Connection exists"
The connection to be added conflicts with an existing connection.

SNMDB_CONNECTION_NOT_FOUND 13 "Connection not found"
The connection to be deleted was not found.

SNMDB_CANNOT_CONNECT_TO_SELF 14 "Cannot connect to self"
The connection cannot be added since element cannot connect to itself.

SNMDB_ELEMENT_ALREADY_IN_VIEW 15 "Already in view"
The element could not be added to the view because it was already in the view.

SNMDB_ELEMENT_NOT_IN_VIEW 16 "Not in view"
The element could not be deleted from the view because it was not in the view.

SNMDB_SUBVIEW_IS_NOT_EMPTY 17 "Subview not empty"
The element cannot be deleted since its subview still contains objects.

SNMDB_UNABLE_TO_DELETE_ELEMENT 18 "Unable to delete"
The element cannot be deleted due to database error.

SNMDB_UNKNOWN_PROPERTY 19 "Unknown property"
The property specified is not defined for the element.

SNMDB_INVALID_DATA_IN_BUFFER 20 "Invalid data buffer"
The data in the element buffer is not valid.

SNMDB_NOT_CONNECTED_TO_ANY 21 "Not connected to any element"
The element is not connected to any other element.

SNMDB_NOT_IN_ANY_VIEW 22 "Not in any view"
The element does not have any membership record; it does not appear in any view.

SNMDB_NO_AGENT_APPLY 23 "No agents apply"
The element does not have any agent run on the system, or applied to the system.

SNMDB_AGENT_NOT_FOUND 24 "No such agent found"
The agent specified is not running on, or does not apply to the element.

SNMDB_X_AND_Y_UNDEFINED 25 "x and y coordinates not defined"
x and y coordinates have not been defined previously for this element glyph. Either the element glyph has been automatically positioned by the Console when found by the Discover tool, or the element has been added to the view (with **snmdb_add_to_view)** with -1 specified for x, y, and z arguments.

SNMDB_INVALID_FILE 26 "Invalid output file pointer"
The file pointer specified is invalid; no output is written to the file.

SNMDB_GLYPH_FILENAME_NOT_FOUND 27 "No glyph filename found"
There is no glyph file name defined for the specified element type.

SNMDB_ALIAS_NOT_FOUND 28 "No alias found"
The object does not have the specified alias.

SNMDB_NO_ALIAS_EXIST 29 "No alias exists"
The object does not have any alias.

SNMDB_UNABLE_TO_DELETE_ALIAS 30 "Unable to delete alias"
Unable to delete the specified alias.

SNMDB_READ_ONLY_MODE 31 "Read only mode"
The database is read-only.

SNMDB_LIMITS_EXCEEDED 32 "Limits exceeded"
The database limit has been exceeded for Site Manager.

SNMDB_NO_LICENSE 33 "No license found"
There is no license installed in the system.

NAME

snm_exec – execute a command and pause after the command exits

SYNOPSIS

snm_exec [command_line]

DESCRIPTION

snm_exec uses the Bourne shell's **eval** to execute a command, and then pause after the command has exited. **snm_exec** will prompt the user to press the Return key after execution completes.

snm_exec is usually used in Site / SunNet / Domain Manager schema files as the first argument to **snm_cmdtool**(1)**.** When used as part of an elementCommand instance record in a schema file, it causes the Site / SunNet / Domain Manager Console to: create a new **cmdtool(1)**, execute the argument of **snm_exec**, and wait for the user to press Return before killing the cmdtool window.

In the special case where arguments to the command being executed by **snm_exec** have embedded spaces, enclose the arguments in quotation marks, and precede each quote with the backslash escape character. For example:

```
% snm_exec command \"spaces within this argument\"
```

SEE ALSO

cmdtool(1) , snm_cmdtool(1)**, snm_schema**(5)

NAME

snm_gr – Site / SunNet / Domain Manager results grapher

SYNOPSIS

snm_gr [-b]

DESCRIPTION

snm_gr is an OPEN LOOK application that allows the display of information sent to it by **snm**(1)**,** the Site / SunNet / Domain Manager console, and by **snm_br**(1)**,** the Site / SunNet / Domain Manager browser. The grapher aids in the visual analysis of information collected in Site / SunNet / Domain Manager data and event reports.

You must explicitly start **snm_gr** from the command line or from the **snm** Tools menu before it can receive any data from **snm** or **snm_br**.

If the **-b** switch is given, backgrounds will appear White rather than Black. You might want to use this option if you intend to print the graphs.

Data Sets

Each **snm_gr** graph consists of one or more *data sets*, with each data set representing a series of values for a single attribute received from an agent. Two types of data sets are supported: dynamic and static.

Dynamic data sets are initially empty but receive updates during the lifetime of the graph. The resulting graphs are similar to "perfmeter" graphs. Dynamic data sets only retain the 180 most recent update value.

Static data sets are initially filled with values but cannot receive updates. Static data sets may contain an unlimited number of values.

USAGE

Sending data from snm

When preparing a data report within the Site / SunNet / Domain Manager console, you may specify that received report information be passed to the results grapher program. Do this by selecting the **Graph Tool** option in the data report's attribute specification pane. When that data report is launched, the **snm** program contacts the results grapher and tells it to prepare a *dynamic* graph. This causes a graph window to appear, which initially contains no information. As **snm** receives periodic updates from that data report, it forwards these results to **snm_gr**, which then displays them.

Sending data from snm_br

Data streams within **snm_br** can be sent to the results grapher. See the **snm_br**(1) man page for more information on how this is done.

Results Grapher Main Window

When **snm_gr** is started, a single window entitled **Results Grapher** appears. This indicates that the grapher is now ready to receive graphs sent to it by **snm** and **snm_br**. When these graphs arrive, a separate **Graph Window** is displayed, and the name of the graph is added to the scrolling list in the **Results Grapher** window. You may then select one or more of the items in the scrolling list, and manipulate them using the **View**, **Properties**, **Remove**, and **Merge** buttons, as explained below:

- **View** - displays and pins the selected graphs. This is useful for retrieving graphs that have been dismissed by unpinning them.
- **Properties** - displays and pins the properties window for the selected graphs.

- **Remove** - deletes and frees the storage associated with the selected graphs.
- **Merge** - creates a new graph containing the data sets of the selected graphs. The selected graphs are *not* automatically removed after they are merged.

Graph Windows

As each graph is started, a window is created that contains the graph of the data. The graph window initially contains a 3 dimensional graph in the center, a legend for the data sets in the graph, and the starting and ending times of the graph in the lower left and right corners of the window.

It is possible to manipulate some aspects of the displayed graph by selecting from the **Graph Menu**, which is obtained by pressing the MENU mouse button while within a graph window.

Graph Menu entries include:

- **Replot graph with new times**. You can "zoom" into a graph to get finer readings. This is accomplished by pressing the SELECT mouse button at a point within the graph and then dragging the mouse to another point within the graph. As the mouse is dragged from one position to another, two time labels appear at the bottom of the graph. The time label on the left displays the start boundary time, and the label on the right displays the end boundary time. Clicking on the **Replot Graph with new times** menu item then causes the graph to be replotted, based on the boundary times between the two points you selected.
- **Show entire graph** causes all data points to be displayed, undoing the effect of any "zoom" requests.
- **Properties** displays the properties window for this graph.

Graph Properties Windows

This window provides fields that affect the display and interpretation of the data. It contains the following fields of interest:

- **Data Scale** determines whether the data sets for a graph are plotted using Absolute or Relative values.

  If Absolute data scale is chosen, all data sets are displayed using the same vertical scale. In addition, vertical scale labels are printed on the graph. Use Absolute data scale when displaying multiple data sets whose data values are of the same units (for example, cpu% for multiple systems).

  If Relative data scale is chosen, each data set is drawn using its own scale. This scale forces the maximum and minimum values of the data set to be displayed over the full height of the graph. Use Relative data scale when displaying multiple data sets whose data values are of different units (for example, cpu%, interrupts, and collisions for a single system).
- **Time Scale** determines whether the timestamps of the data sets are plotted in Absolute or Relative scale. In Absolute time scale, the timestamps of all data reports are interpreted literally, and the data sets are displayed in (possibly overlapping) chronological order. Use Absolute time scale when displaying data sets from reports with overlapping times (for example, yesterday's cpu% for three systems).

In Relative time scale, all data sets are assumed to start at time zero (that is, 00:00:00) and thus are displayed side-by-side on the graph. This is useful for comparing data sets whose timestamps differ by large amounts (for example, cpu% on three different days). Since there is no concept of a true starting time for relative graphs, the timestamps at the bottom of the graph only display the relative time (in hours, minutes, seconds) from their first data value.

- **Order** determines the order in which the data sets are displayed if there is more than one data set in the graph. If Normal is chosen, the first data set in the **Data Set** scrolling list will correspond to the data set drawn furthest to the back of the graph. Reverse causes the first data set in the scrolling list to correspond with the dataset drawn at the front of the graph. This field is useful to get a clearer view of data sets are partially obscured by other data sets.
- **Data Set Scrolling List** shows a list of all the data sets for a graph. This is an exclusive list; that is, only one item may be selected at any time.

The following properties only affect the selected **Data Set Scrolling List** item:

- **Data Set** controls the visibility of the selected data set. Show makes the data set information visible, while Hide makes the data set information invisible.
- **Plot Value** specifies that the data set is to be plotted in one of three modes: Absolute, which plots the values of the data set as is; Cumulative, which sums up each successive data value and plots them; and Delta, which takes the difference between each pair of data values and plots them.

No changes to this window take effect until the **Apply** button is pressed. The **Reset** button resets all fields to their values from the last time the **Apply** button was pressed.

Graph Rotation Angles

The **Rotation Angles** window allows you to change the default viewing angles in the graph using the following controls:

- **Elevation** controls the rotation of the graph about the horizontal axis. A zero elevation angle will view the graph from the side, and a 90 degree viewing angle allows you to view the graph from the top.
- **Rotation** controls the rotation of the graph about the vertical axis.

SEE ALSO

snm(1), **snm_br**(1)

NOTES

To help keep graphs reasonable, a maximum of four lines of attribute names are displayed in the legend. Increasing the width of the graph window allows more attribute names to be displayed.

No timestamps or legends are displayed when the graph window size is shrunk to less than 250x250 pixels.

The display of dynamic data sets from **snm** data reports is both extremely CPU-intensive and consumes large amounts of memory. Starting multiple data reports with low report intervals is not recommended. Your results may vary.

If a console data report (which contains a "Graph Tool" attribute request) is started when

snm_gr is not running, a dynamic graph is not created and no updates are processed. Thus, if the runtime database contains any restartable data requests, **snm_gr** should be started before **snm**, so that any graphs will be automatically created during the **snm** initialization.

When displaying dynamic graphs, **snm_gr** attempts to use X11 server pixmaps for double-buffering. X11 pixmaps are a limited server resource that may not be available on all servers. If **snm_gr** is unable to acquire a pixmap, it will instead perform dynamic update redisplays directly to the screen. The resulting flicker may make the graph hard to read.

The maximum value **snm_gr** can handle is limited to 2^31 - 1.

NAME

snm_ipx_discover – IPX nodes discoverer for Site / SunNet / Domain Manager

SYNOPSIS

snm_ipx_discover [–**d** *debuglevel*] [–**n** *nxisname* –**i** *ipaddress* –**p** *password*] ...

DESCRIPTION

snm_ipx_discover creates SunNet Manager runtime database records for IPX devices (clients / servers).

snm_ipx_discover uses NXIS (NetWare Export Import Service) to obtain its data by connecting to one or more NXIS servers. The data that is gathered is analyzed to determine its host type, capability, etc., which then is added to the database.

NXIS is a new service within Novell's NetWare Management System (NMS) where it exports the NMS topology database to other network management systems. This eliminates the need for SNM to natively discover NetWare on TCP port 1486.

Built into the **snm_ipx_discover** tool is the Schedule Discover function. Using this functionality, it can periodically query the NXIS server for new data that has been discovered.

Before **snm_ipx_discover** can be started, the runtime database must be available. The easiest way to do this is to run the Console, **snm.** When **snm** is started for the first time, you have the choice of a "Head Start" option, in which IP Discover uses the *arp* table to find no more than 10 elements on the local subnetwork, or "Basic Start", in which no discovery is performed.

OPTIONS

The **snm_ipx_discover** needs to know about the NXIS server and its password to talk. These functions can be specified through the command line or through the GUI. Following are the available options.

Each option is recognized as a separate argument.

–**d** <*debuglevel*>
Where <*debuglevel*> is 1. The only supported value for this release is 1.

–**n** <*nxisname*>
Specifies the server name.

–**i** <*ipaddress*>
Specifies the *ipaddress* of the NXIS server.

–**p** <*password*>
Specifies the *password* of the NXIS server.

If environment variable **SNMDBDIR** is set, it is used as the location of the runtime database file. Otherwise, **/var/opt/SUNWconn/snm** on Solaris 2.x or **/var/adm/snm** on Solaris 1.x is used.

SEE ALSO

snm_discover(8)

NAME

snm_kill – stop one or more agent requests

SYNOPSIS

snm_kill -h *agent_host* -m *manager_host* [-a *agent_name* | -A]
[-d *debug_level*] [-t *timeout*] [-v *agent_version_number*]
[-s *timestamp (seconds)* -u *timestamp (microseconds)*]

DESCRIPTION

snm_kill stops an agent's requests according to the given options.

You can stop requests from other than where they were started. For example, you can kill an agent from *host1* that was started from *host2*. This is useful when *host2* is inaccessible.

OPTIONS

Each option is recognized as a separate argument.

–**a** *agent_name*
name of the agent, as it appears in the NIS **rpc.bynumber** map or local **rpc**(5) database.

-A kill all requests started from the local host. **snm_kill** reads the activity daemon's request log (as specified by the **activity-log** keyword in **snm.conf**(5)) and sends a kill request for each entry in the log.

–**d** *debug_level*
set trace level (0 to 3). Usually not needed nor interesting.

–**h** *agent_host*
host where the agent is running.

–**m** *manager_host*
host where the request originated.

–**s** *timestamp(seconds)*
request ID, uniquely identified by a timestamp in seconds (that is, UNIX time). The **-u** option is required with this option.

–**t** *timeout*
maximum time to wait for agent confirmation, in seconds. The default is NETMGT_TIMEOUT (60 seconds).

–**v** *agent_version_number*
protocol version of the agent. Use **10** for the current release.

–**u** *timestamp(microseconds)*
request ID, uniquely identified by a timestamp in microseconds (that is, UNIX time). The **-s** option is required with this option.

NOTES

The values for the **-s** and **-u** options are the seconds and microseconds values, respectively, of the request timestamp. These are displayed by **snm_cmd**(1) when a request is started. If you do not use the **-s** and **-u** options, all *agent name* requests running on *agent host* that were started from *manager host* are terminated.

The ability to kill requests you didn't start could be considered a security hole.

Killing agents started by the Site/SunNet/Domain Manager Console will cause the Console to complain about agents suddenly dying.

NAME

snm_parser – Site / SunNet / Domain Manager schema parser

SYNOPSIS

snm_parser [file]

DESCRIPTION

snm_parser is a example application for parsing Site / SunNet / Domain Manager schema files. The parser program is provided in source form (as a YACC-grammar and a driver application) to allow others to easily build applications that can read schema files.

The parser is constructed to be used with callbacks in the user's program. When particular tokens are encountered, specific callbacks are made. See the test driver source code for these callback names.

SEE ALSO

See the *Site/SunNet/Domain Manager Administration Guide*, section ''Management Database.''

NAME

snm schema – format of Site / SunNet / Domain Manager schema

SYNOPSIS

∗**.schema**

DESCRIPTION

A schema provides information in ASCII form used to build the management database. A schema file consists of one or more:

Agent Definitions
Record Definitions
Instance Definitions
Cluster Definitions
Enumeration Definitions

Agent Definition

An agent definition provides information on the attributes an agent manages. The format of an agent definition is:

agent | proxy *agentName*
[description *"description text"***]**
[map *"mappings"***]**
[serial *integer***]**
[rpcid *program-number***]**
(
[Enumeration Definition]
...
Group/Table Definition
...
[Agent Errors]
)

Group/Table Definition

A group / table definition is part of an agent definition. A group is a logical set of the attributes in an agent. A table is a special case of a group that can return multiple instances of the group. One of the fields in a table is the key that is used to select a single instance of the table. The format of a table / group definition is:

group | table *groupName*
[description *"description text"***]**
[characteristics *characteristics options***]**
(
Field Definition
...
)

Record Definition

Record definitions are used to provide internal definition for the console or to define element types. The format of a record definition is:

record *recordName* **(**
Field Definition
...
)

Field Definition

A field definition defines a field in either a record, group, or table. Optional description and units fields can be provided. Each field consists of an access type and and a field name. The format is:

[*Access*] *Type fieldName*
[description *"description text"*
[characteristics *characteristics options*]
[units *units text*]
(

Access

The following access types are defined:

readwrite | **rw**
readonly | **ro**
writeonly | **wo**
notaccessible | **na**

Type

The following field types are defined:

[**unsigned**] **int** | **short** | **long**
string[*length***]**
octet[*length***]**
enum *enumName*
float | **double**
unixtime | **timeticks**
inetaddr | **netaddr**
counter | **gauge**
objectid

Enumeration Definition

Enumerations can be defined that map between integers and strings. The format of an enumeration definition is:

enum *enumName* **(**
value "*description*"
...
)

Agent Errors

Agents may define their own errors in the **agentErrors** section of an agent definition. This section provides the mapping between the agent error codes and the error strings. Its format is:

agentErrors (
errorCode "*description*"
...
)

Cluster definition

A cluster record is a group of records that make up an instance of an element or request. Its format is:

cluster (
recordName **(** *field_value field_value ...* **)**
...
)

Instance Definition

One or more instances of a record can be defined using an instance definition. Its format is:

instance (
(*field_value field_value ...* **)**
...
)

Characteristics Options

The characteristics options are used in schema files which describe snmp devices. The characteristics are passed along to the snmp agent when SNMP requests are made, supplying addtional information used in fufilling the request. See the schema files (e.g. snmp.schema) for more information. The valid characteristics are:

For Groups
-K key

For Tables
-K Key

For Attributes (Field Definitions)
-N <attribute name> e.g. sysDescr
-O <object identifier> e.g. 1.4.6.1.2.1.1.1
-T <type> where <type> is STRING, INTEGER, OBJECTID, etc.
-A <access>
-X <translation>
-F <fake out value>

-C <format string>

NAME

snm_set – Site / SunNet / Domain Manager Set Tool

SYNOPSIS

snm_set -t *target_system* **[-m** *system_mapped* **] -a** *agent_name* **[-g** *group_name* **]**

DESCRIPTION

snm_set is the Site / SunNet / Domain Manager Set Tool. It allows setting the writable attributes for a system.

OPTIONS

–**t** *target_system*
specifies the system containing the object to be set.

–**m** *system_mapped*
specifies the system name to which the target_system is mapped.

–**a** *agent_name*
specifies the agent or proxy name that appears in the agent schema file. Note that this name can be different from the name of the agent schema file.

–**g** *group_name*
specifies the group for the object to be set

USAGE

Showing current values

Clicking the "Get" button will cause a get request to be made for retrieving the current values for all attributes in the specified group. These values are displayed in the center portion of the tool window. If the group is a table, you are prompted to fill in the "Key" field. If no key field is entered, **snm_set** will display the first row in the center tool window, retrieve all the potential keys for that table, and provide a menu that can be used to select a key.

Building up a set request

In the center portion of the tool window, there are three columns showing the *Attribute Name* , its *Current Value* (if a "Get" has been made previously), and its *New Value* field. If the attribute is settable, there will be a line in the *New Value* column. Settable attributes that are enumerations will have a menu displayed in the *New Value* column from which new values can be selected.

To change a settable attribute, enter the new value in the *New Value* column and press Return. The new value will be shown in the list in the Set Information window.

Deleting settings

To delete a setting from the Set Information window, select the setting line, and use the "Delete Selection" option from the "Delete" menu. The "Delete All" option from the "Delete" menu will delete all the settings from the Set Information window.

Saving and restoring settings

Settings can be saved and restored to the Set Information window using the "Load" or "Save" option from the "File" menu.

Executing the set request

Clicking the "Set" button will cause the settings in the Set Information window to be made. The footer will display a message indicating whether the setting was successful. If the setting was successful, the Set Information window will be cleared.

Undo the last set request

Clicking the "Unset" button will restore the settings before the last Set request. The footer will display a message indicating whether the unsetting was successful.

NAME

snm_version – print version information about Site / SunNet / Domain Manager binaries

SYNOPSIS

snm_version [-a] [filename ...]

DESCRIPTION

snm_version displays information about the currently installed version of Site / SunNet / Domain Manager. By default, version information is displayed for the Site / SunNet / Domain Manager Console. If a filename or list of filenames is passed to **snm_version**, then information is returned about all files listed on the command line.

OPTIONS

-a Displays version information for all Site / SunNet / Domain Manager tools and agents installed on the machine. Installation information is taken from the **/etc/opt/SUNWconn/snm/snm.conf** configuration file for Solaris 2.x and **/etc/snm.conf** for Solaris 1.x.

SEE ALSO

snm.conf(5)

NAME

snmdb_add – adds a new element into the database.

SYNOPSIS

```
include <netmgt/netmgt_db.h>

snmdb_add(buffer)
        snmdb_buffer *buffer;
```

DESCRIPTION

Before adding a new element into the database, you should call **snmdb_init_buffer**(3N) to initialize the internal buffer. Then, use **snmdb_set_property**(3N) to set the properties of the element. Routines like **snmdb_add_to_view**(3N)**, snmdb_add_connection**(3N)**, snmdb_set_color**(3N)**,** or **snmdb_add_agent**(3N) may be called to add more records in the buffer describing the element.

INPUT ARGUMENTS

buffer pointer to a structure of type **snmdb_buffer**.

RETURN VALUE

Returns 1 if element has been added successfully. Otherwise, it returns 0, and the external variable **snm_error**(3N) is set to the error reason.

SEE ALSO

snmdb_init_buffer(3N)**, snmdb_set_property**(3N)**, snmdb_add_to_view**(3N)**, snmdb_add_connection**(3N)**, snmdb_set_color**(3N)**, snmdb_add_agent**(3N)

NAME

snmdb_add_agent – adds an agent record for the element

SYNOPSIS

```
#include <netmgt/netmgt_db.h>

snmdb_add_agent(buffer, agent, proxy)
        snmdb_buffer *buffer;
        char *agent;
        char *proxy;
```

DESCRIPTION

You may add an agent record in the element buffer by specifying the name of the agent, and the name of the system providing the proxy function if the agent is a proxy.

INPUT ARGUMENTS

buffer pointer to a structure of type **snmdb_buffer**.

agent name of the agent.

proxy optional, name of the system that provides the proxy function if it is a proxy agent.

RETURN VALUE

Returns 1 if successful. Otherwise, it returns 0, and the external variable **snm_error**(3N) is set to the error reason.

SEE ALSO

snmdb_get_agent(3N)**, snmdb_delete_agent**(3N)**, snmdb_enumerate_agents**(3N)

NAME

snmdb_add_alias – adds an alias record for the element

SYNOPSIS

```
#include <netmgt/netmgt_db.h>

snmdb_add_alias(buffer, aliasName)
        snmdb_buffer *buffer;
        char *aliasName;
```

DESCRIPTION

The name of element used at creation time is its primary name. There may be one or more secondary names added later for the element by adding alias records. Both primary and secondary names should be unique in the elements name space.

An element may be identified by multiple names through its alias records. **snmdb_add_alias**(3N) adds an alias record into the buffer, specifying that the element is also known as *aliasName*.

INPUT ARGUMENTS

buffer pointer to a structure of type **snmdb_buffer**.

aliasName
the secondary name which the element is also known as.

RETURN VALUE

Returns 1 if successful. Otherwise, it returns 0, and the external variable **snm_error**(3N) is set to the error reason.

SEE ALSO

snmdb_delete_alias(3N), **snmdb_enumerate_aliases**(3N)

NAME

snmdb_add_connection – adds a connection record for the element

SYNOPSIS

```
#include <netmgt/netmgt_db.h>

snmdb_add_connection(buffer, connectedto)
        snmdb_buffer *buffer;
        char *connectedto;
```

DESCRIPTION

Connections are described in the connect record in the element buffer. **snmdb_add_connection** adds a connect record into the buffer, specifying that the element is connected to *connectedto*.

INPUT ARGUMENTS

buffer pointer to a structure of type **snmdb_buffer**.

connectedto
name of the target that the element is connected to.

RETURN VALUE

Returns 1 if successful. Otherwise, it returns 0, and the external variable **snm_error**(3N) is set to the error reason.

SEE ALSO

snmdb_delete_connection(3N)**, snmdb_enumerate_connections**(3N)

NAME

snmdb_add_to_view – adds the element into a particular view

SYNOPSIS

```
#include <netmgt/netmgt_db.h>

snmdb_add_to_view(buffer, viewname, x, y, z, x1, y1)
        snmdb_buffer *buffer;
        char *viewname;
        int x;
        int y;
        int z;
        int x1;
        int y1;
```

DESCRIPTION

The view in which the element exists is described in the element's membership record. **snmdb_add_to_view** adds a membership record with the specified view name into the element buffer. x, y, and z values specify the position of the element in that view. If the element is of type **bus.ethernet** (as defined in the **elements.schema** file), you will also need to specify the x1, y1 values of the end point of the bus. You may also let the element be auto positioned by specifying '-1' as the value of all x, y, and z variables.

INPUT ARGUMENTS

buffer cluster record buffer for the element.

viewname
name of the view in which the element exists.

x X coordinate of the element in the view.

y Y coordinate of the element in the view.

z Z value of the element in the view. This value is used when glyphs are stacked on top of each other.

x1 optional. This is used for element of category **bus** only, the X position of second end point of the bus element.

y1 optional. This is used for element of category **bus** only, the Y position of second end point of the bus element.

RETURN VALUE

Returns 1 if successful. Otherwise, it returns 0, and the external variable **snm_error**(3N) is set to the error reason.

FILES

elements.schema

SEE ALSO

snmdb_delete_from_view(3N)**, snmdb_get_view**(3N)**, snmdb_enumerate_views**(3N)

NAME

snmdb_console_load – causes the Site / SunNet / Domain Manager Console to load a specified ASCII management database (MDB) file

SYNOPSIS

```
int snmdb_console_load(pathname, rpc_error, console_error)
        char    *pathname;
        int     *rpc_error, *console_error;
```

DESCRIPTION

Loading the database is accomplished through an RPC call to the Site / SunNet / Domain Manager Console.

The call returns only after the load operation is completed. The Console should not be locked when this call is issued.

INPUT ARGUMENTS

pathname
pointer to the character string that represents the path name of the MDB file.

rpc_error
pointer to integer for returned errors (can be NULL pointer).

console_error
pointer to integer for returned errors (can be NULL pointer).

RETURN VALUE

One of the following values is returned:

0 Success.

1 Could not contact the Site / SunNet / Domain Manager Console successfully. Either the Console was not running or the version of the Console that is running does not support this call. No specific errors are returned.

2 RPC error. The value of the integer pointed to by **rpc_error** is set to one of the values of type **enum clnt_stat** defined in the header file **/usr/include/rpc/clnt.h**, cast into an integer.

3 Operation failed. This means that the value of the integer pointed to by **console_error** is set to a UNIX error code, as defined in **/usr/include/sys/errno.h.**

NAME

snmdb_console_reload – causes the Site / SunNet / Domain Manager Console to reinitialize the runtime database and load the specific server

SYNOPSIS

```
int snmdb_console_reload(pathname, rpc_error, console_error)
        char    *pathname;
        int     *rpc_error, *console_error;
```

DESCRIPTION

Loading the database is accomplished through an RPC call to the Site / SunNet / Domain Manager Console.

The call returns only after the reload operation is completed. The Console should not be locked when this call is issued.

INPUT ARGUMENTS

pathname
pointer to the character string that represents the path name of the MDB file.

rpc_error
pointer to integer for returned errors (can be NULL pointer).

console_error
pointer to integer for returned errors (can be NULL pointer).

RETURN VALUE

One of the following values is returned:

0 Success.

1 Could not contact the Site / SunNet / Domain Manager Console successfully. Either the Console was not running or the version of the Console that is running does not support this call. No specific errors are returned.

2 RPC error. The value of the integer pointed to by **rpc_error** is set to one of the values of type **enum clnt_stat** defined in the header file **/usr/include/rpc/clnt.h**, cast into an integer.

3 Operation failed. This means that the value of the integer pointed to by **console_error** is set to a UNIX error code, as defined in **/usr/include/sys/errno.h.**

NAME

snmdb_console_save_components – causes the Site/SunNet/Domain Manager Console to save all instances in its runtime database to a specified ASCII management database (MDB) file

SYNOPSIS

```
int snmdb_console_save_components(pathname, rpc_error, console_error)
        char    *pathname;
        int     *rpc_error, *console_error;
```

DESCRIPTION

Saving the database is accomplished through an RPC call to the Site/SunNet/Domain Manager Console. Instances of data and event requests as well as objects are saved.

The call returns only after the operation is completed. The Console should not be locked when this call is issued.

INPUT ARGUMENTS

pathname
pointer to the character string that represents the path name of the MDB file.

rpc_error
pointer to integer for returned errors (can be NULL pointer).

console_error
pointer to integer for returned errors (can be NULL pointer).

RETURN VALUE

One of the following values is returned:

0 Success.

1 Could not contact the Site/SunNet/Domain Manager Console successfully. Either the Console was not running or the version of the Console that is running does not support this call. No specific errors are returned.

2 RPC error. The value of the integer pointed to by **rpc_error** is set to one of the values of type **enum clnt_stat** defined in the header file **/usr/include/rpc/clnt.h ,** cast into an integer.

3 Operation failed. This means that the value of the integer pointed to by **console_error** is set to a UNIX error code, as defined in **/usr/include/sys/errno.h.**

NAME

snmdb_delete – deletes an existing element from the database

SYNOPSIS

```
#include <netmgt/netmgt_db.h>

snmdb_delete(name)
        char *name;
```

DESCRIPTION

snmdb_delete deletes an existing element.

INPUT ARGUMENTS

name name of the element.

RETURN VALUE

Returns 1 if element has been deleted successfully. Otherwise, it returns 0, and the external variable **snm_error**(3N) is set to the error reason.

NAME

snmdb_delete_agent – deletes an agent record for the element

SYNOPSIS

```
#include <netmgt/netmgt_db.h>

snmdb_delete_agent(buffer, agent)
        snmdb_buffer *buffer;
        char *agent;
```

DESCRIPTION

snmdb_delete_agent removes the agent record from the element buffer.

INPUT ARGUMENTS

buffer pointer to a structure of type **snmdb_buffer**.

agent name of the agent.

RETURN VALUE

Returns 1 if successful. Otherwise, it returns 0, and the external variable **snm_error**(3N) is set to the error reason.

SEE ALSO

snmdb_add_agent(3N)**, snmdb_get_agent**(3N)**, snmdb_enumerate_agents**(3N)

NAME

snmdb_delete_alias – deletes an alias record for the element

SYNOPSIS

```
#include <netmgt/netmgt_db.h>

snmdb_delete_alias(buffer, aliasName)
        snmdb_buffer *buffer;
        char *aliasName;
```

DESCRIPTION

An element may be identified by the name it was created initially or by its secondary names specified in the alias records. **snmdb_delete_alias** removes an alias record from the element buffer so that the element can no longer be accessed via *aliasName*.

INPUT ARGUMENTS

buffer pointer to a structure of type **snmdb_buffer**.

aliasName
the secondary name which the element is also known as.

RETURN VALUE

Returns 1 if successful. Otherwise, it returns 0, and the external variable **snm_error**(3N) is set to the error reason.

SEE ALSO

snmdb_add_alias(3N)**, snmdb_enumerate_aliases**(3N)

NAME

snmdb_delete_color – deletes the color value of an element

SYNOPSIS

```
#include <netmgt/netmgt_db.h>

snmdb_delete_color(buffer)
        snmdb_buffer *buffer;
```

DESCRIPTION

Color of an element is described in the glyphColor record in the element buffer. **snmdb_delete_color** removes the glyphColor record from the buffer.

INPUT ARGUMENTS

buffer pointer to a structure of type **snmdb_buffer**.

RETURN VALUE

Returns 1 if successful. Otherwise, it returns 0.

SEE ALSO

snmdb_get_color(3N)**, snmdb_set_color**(3N)

NAME

snmdb_delete_connection – deletes a connection record for the element

SYNOPSIS

```
#include <netmgt/netmgt_db.h>

snmdb_delete_connection(buffer, connectedto)
        snmdb_buffer *buffer;
        char *connectedto;
```

DESCRIPTION

Connections are described in the connect record in the element buffer. **snmdb_delete_connection** looks through all the connect records in the buffer, and finds and removes the one that describes the connection between the element and *connectedto.*

INPUT ARGUMENTS

buffer pointer to a structure of type **snmdb_buffer**.

connectedto
name of the target that the element is connected to.

RETURN VALUE

Returns 1 if successful. Otherwise, it returns 0, and the external variable **snm_error**(3N) is set to the error reason.

SEE ALSO

snmdb_add_connection(3N)**, snmdb_enumerate_connections**(3N)

NAME

snmdb_delete_from_view – deletes the element from a particular view

SYNOPSIS

```
#include <netmgt/netmgt_db.h>

snmdb_delete_from_view(buffer, viewname)
        snmdb_buffer *buffer;
        char *viewname;
```

DESCRIPTION

The view in which the element exists is described in the element's membership record. **snmdb_delete_from_view** removes the membership record with the specified view name from the element buffer.

INPUT ARGUMENTS

buffer pointer to a structure of type **snmdb_buffer**.

viewname
name of the view in which the element exists.

RETURN VALUE

Returns 1 if successful. Otherwise, it returns 0, and the external variable **snm_error**(3N) is set to the error reason.

SEE ALSO

snmdb_add_to_view(3N)**, snmdb_get_view**(3N)**, snmdb_enumerate_views**(3N)

NAME

snmdb_enumerate_agents – enumerates the agents that apply to the element

SYNOPSIS

```
#include <netmgt/netmgt_db.h>

char **
snmdb_enumerate_agents(buffer)
        snmdb_buffer *buffer;
```

DESCRIPTION

Agents and proxy agents are described in the agent record in the element buffer. **snmdb_enumerate_agents** enumerates the agent records in the buffer, and returns the pointer pointing to the list of all the agent names. After using the list, you should call **snmdb_free_list**(3N) to free the space.

INPUT ARGUMENTS

buffer pointer to a structure of type **snmdb_buffer**.

RETURN VALUE

Returns a NULL terminated list of all agent names that apply to the element. Returns NULL if an error occurs, and the external variable **snm_error**(3N) is set to the error reason.

SEE ALSO

snmdb_free_list(3N)

NAME

snmdb_enumerate_aliases – enumerates all the alias records of the element

SYNOPSIS

```
#include <netmgt/netmgt_db.h>

char **
snmdb_enumerate_aliases(buffer)
        snmdb_buffer *buffer;
```

DESCRIPTION

An element may be identified by its secondary names specified in the alias records in the element buffer. **snmdb_enumerate_aliases** enumerates all the alias records in the buffer and returns a pointer pointing to the list of all the names which the element is also known as. After using the list, you should call **snmdb_free_list**(3N) to free the space.

INPUT ARGUMENTS

buffer pointer to a structure of type **snmdb_buffer**.

RETURN VALUE

Returns a NULL terminated list of all the secondary names of the element. Returns NULL if an error occurs, and the external variable **snm_error**(3N) is set to the error reason.

SEE ALSO

snmdb_free_list(3N)

NAME

snmdb_enumerate_connections – enumerates all the connections of the element

SYNOPSIS

```
#include <netmgt/netmgt_db.h>

char **
snmdb_enumerate_connections(buffer)
        snmdb_buffer *buffer;
```

DESCRIPTION

Connections are described in the connect record in the element buffer. **snmdb_enumerate_connections** enumerates all the connect records in the buffer and returns a pointer pointing to the list of all the names to which the element is connected. After using the list, you should call **snmdb_free_list**(3N) to free the space.

INPUT ARGUMENTS

buffer pointer to a structure of type **snmdb_buffer**.

RETURN VALUE

Returns a NULL terminated list of all the names to which the element is connected. Returns NULL if an error occurs, and the external variable **snm_error**(3N) is set to the error reason.

SEE ALSO

snmdb_free_list(3N)

NAME

snmdb_enumerate_elements – enumerates all the elements in a view

SYNOPSIS

```
#include <netmgt/netmgt_db.h>

snmdb_handle *
snmdb_enumerate_elements(viewname, type)
        char *viewname;
        char *type;
```

DESCRIPTION

snmdb_enumerate_elements enumerates all the elements in a particular view named *viewname*. If *viewname* is NULL, it enumerates all the elements in all views. *type* is used to specify that only the elements of that type should be enumerated in the view. If it is NULL, it means that elements of all types should be enumerated.
snmdb_enumerate_elements returns a pointer pointing to the list of all the elements found. You should call **snmdb_get_next_element**(3N) to retrieve the name of the elements in the list. After using the list, you should call **snmdb_free_enumeration_handle**(3N) to free the space used.

INPUT ARGUMENTS

viewname
name of the view to enumerate.

type name of the type of element to enumerate.

RETURN VALUE

Returns NULL if there is no element matching the view and type. Otherwise, it returns the pointer pointing to the list of elements that matched.

SEE ALSO

snmdb_get_next_element(3N)**, snmdb_free_enumeration_handle**(3N)

NAME

snmdb_enumerate_views – enumerates the views in which the element exists.

SYNOPSIS

```
#include <netmgt/netmgt_db.h>

char **
snmdb_enumerate_views(buffer)
        snmdb_buffer *buffer;
```

DESCRIPTION

The view in which the element exists is described in the element's membership record. **snmdb_enumerate_views** enumerates the membership records in the element buffer, and returns a pointer pointing to the list of all the view names in which the element exists. After using the list, you may call **snmdb_free_list**(3N) to free the space.

INPUT ARGUMENTS

buffer pointer to a structure of type **snmdb_buffer**.

RETURN VALUE

returns a NULL terminated list of all view names in which the element exists. Returns NULL if error, and the external variable **snm_error**(3N) is set to the error reason.

SEE ALSO

snmdb_free_list(3N)

NAME

snmdb_free_enumeration_handle – frees enumeration storage of elements

SYNOPSIS

```
#include <netmgt/netmgt_db.h>

void
snmdb_free_enumeration_handle(handle)
        snmdb_handle *handle;
```

DESCRIPTION

snmdb_free_enumeration_handle frees the storage allocated previously for enumerating element names.

INPUT ARGUMENTS

handle pointer pointing to the enumerating list returned by a call to **snmdb_enumerate_elements**(3N)**.**

RETURN VALUE

None.

SEE ALSO

snmdb_enumerate_elements(3N)**, snmdb_get_next_element**(3N)

NAME

snmdb_free_list – frees enumeration storage of agent list, view list, and connection list

SYNOPSIS

```
#include <netmgt/netmgt_db.h>

void
snmdb_free_list(list)
        char **list;
```

DESCRIPTION

snmdb_free_list frees the storage allocated previously for enumerating agents, views, or connections.

INPUT ARGUMENTS

list pointer pointing to the enumerated list returned by a call to **snmdb_enumerate_agents**(3N)**, snmdb_enumerate_views**(3N)**,** or **snmdb_enumerate_connections**(3N)**.**

RETURN VALUE

None.

SEE ALSO

snmdb_enumerate_elements(3N)**, snmdb_get_next_element**(3N)

NAME

snmdb_get_agent – gets an agent record for the element

SYNOPSIS

```
#include <netmgt/netmgt_db.h>

snmdb_get_agent(buffer, agent, isproxy, proxy)
        snmdb_buffer *buffer;
        char *agent;
        int *isproxy;
        char *proxy;
```

DESCRIPTION

snmdb_get_agent retrieves the agent record for *agent*, indicating whether the agent is a proxy, and, if it is, the name of the system that provides the proxy function.

INPUT ARGUMENTS

buffer pointer to a structure of type **snmdb_buffer**.

agent name of the agent.

isproxy Boolean value specifying if the agent is a proxy.

proxy name of the system that provides the proxy function.

RETURN VALUE

Returns 1 if an agent is found. Otherwise, it returns 0.

SEE ALSO

snmdb_add_agent(3N)**, snmdb_delete_agent**(3N)**, snmdb_enumerate_agents**(3N)

NAME

snmdb_get_color – reads the color value of an element

SYNOPSIS

```
#include <netmgt/netmgt_db.h>

snmdb_get_color(buffer, red, green, blue)
        snmdb_buffer *buffer;
        int *red;
        int *green;
        int *blue;
```

DESCRIPTION

snmdb_get_color finds out the color value in RGB numbers of the element loaded in the buffer. The colors are contained in the glyphColor record in the buffer. If the color is not defined, the value of variables *red*, *green*, and *blue* may not be valid.

INPUT ARGUMENTS

buffer	pointer to a structure of type **snmdb_buffer**.
red	red color value of the element.
green	green color value of the element.
blue	blue color value of the element.

RETURN VALUE

Returns 1 if successful; otherwise, it returns 0.

SEE ALSO

snmdb_set_color(3N)**, snmdb_delete_color**(3N)

NAME

snmdb_get_element_glyph – returns the glyph file name of an element type

SYNOPSIS

```
#include <netmgt/netmgt_db.h>

char *
snmdb_get_element_glyph(type)
        char *type;
```

DESCRIPTION

snmdb_get_element_glyph finds out the glyph / icon file name for a particular type of the element.

INPUT ARGUMENTS

type type of element.

RETURN VALUE

Returns the glyph file name for the element type specified if it is defined. Otherwise, it returns NULL. In case of an error, the external variable **snm_error**(3N) is set to the error reason.

SEE ALSO

snmdb_get_element_type(3N)

NAME

snmdb_get_element_type – returns the type of an element

SYNOPSIS

```
#include <netmgt/netmgt_db.h>

char *
snmdb_get_element_type(buffer)
        snmdb_buffer *buffer;
```

DESCRIPTION

snmdb_get_element_type finds out the type of the element loaded in the internal buffer.

INPUT ARGUMENTS

buffer pointer to a structure of type **snmdb_buffer**.

RETURN VALUE

Returns NULL if any error occurs; otherwise, it returns the type of the element. In case of error, the external variable **snm_error**(3N) is set to the error reason.

SEE ALSO

snmdb_enumerate_elements(3N)**, snmdb_get_next_element**(3N)

NAME

snmdb_get_next_element – get the next element in the enumerated list returned from **snmdb_enumerate_elements**

SYNOPSIS

```
#include <netmgt/netmgt_db.h>

char *
snmdb_get_next_element(handle)
        snmdb_handle *handle;
```

DESCRIPTION

Call **snmdb_enumerate_elements**(3N) to generate: the enumerated list of the elements of a certain type in a view, the elements of certain type in all views, all elements in a view, or all elements in all views. Then you may get the element name one at a time by calling **snmdb_get_next_element** to get the next name in the list.

INPUT ARGUMENTS

handle pointer pointing to the enumerated list returned by a call to **snmdb_enumerate_elements**(3N).

RETURN VALUE

Returns NULL if there is no remaining element in the list. Otherwise, it returns the name of next element in the list.

SEE ALSO

snmdb_enumerate_elements(3N)**, snmdb_free_enumeration_handle**(3N)

NAME

snmdb_get_property – gets the value and data type of a particular property of an element that has been read into the internal buffer

SYNOPSIS

```
#include <netmgt/netmgt_db.h>

snmdb_data
snmdb_get_property(buffer, property, type)
        snmdb_buffer *buffer;
        char *property;
        snmdb_type *type;
```

DESCRIPTION

snmdb_get_property returns the value of a property, which is of type **snmdb_data**, a union of all possible data types, as defined in **netmgt_db.h**. The *buffer* should be loaded with an element by calling **snmdb_read**(3N). The data type of the property will also be returned in *type* if *type* is not NULL.

INPUT ARGUMENTS

buffer pointer to a structure of type **snmdb_buffer**.

property
name of the property.

type data type of the property, as defined in **netmgt_db.h**.

RETURN VALUE

Returns the value of the property. Check the data type before using the value. If the data type is 0 (undefined), the value may not be valid.

FILES

netmgt_db.h

SEE ALSO

snmdb_read(3N)**, snmdb_set_property**(3N)

NAME

snmdb_get_view – reads the coordinates of the element in a particular view

SYNOPSIS

```
#include <netmgt/netmgt_db.h>

snmdb_get_view(buffer, viewname, x, y, z, x1, y1)
        snmdb_buffer *buffer;
        char *viewname;
        int *x;
        int *y;
        int *z;
        int *x1;
        int *y1;
```

DESCRIPTION

The membership records in the element buffer describe the x and y coordinate values of the element in the views, and whether the glyph of the element is stacked on top of another glyph (z value). **snmdb_get_view** retrieves the x and y coordinates in the membership record which describe the position of the element in view *viewname.* It also retrieves the z value, which is used when glyphs are stacked up on top of each other. If the element is of type **bus.ethernet**, as defined in **elements.schema** file, it will also retrieve the x1, y1 values of the end point position of the bus.

INPUT ARGUMENTS

buffer pointer to a structure of type **snmdb_buffer**.

viewname
name of the view in which the element exists.

x X coordinate of the element in the view.

y Y coordinate of the element in the view.

z Z value of the element in the view. This value is used when glyphs are stacked on top of each other.

x1 optional, used for an element of type **bus.ethernet** only, the X position of second end point of the bus element.

y1 optional, used for an element of type **bus.ethernet** only, the Y position of second end point of the bus element.

RETURN VALUE

Returns 1 if successful; otherwise, it returns 0. In case of an error, the external variable **snm_error**(3N) is set to the error reason.

FILES

elements.schema

SEE ALSO

snmdb_add_to_view(3N)**, snmdb_delete_from_view**(3N)**, snmdb_enumerate_views**(3N)

NAME

snmdb_init_buffer – initialize a buffer with the element's name and type

SYNOPSIS

```
#include <netmgt/netmgt_db.h>

snmdb_init_buffer(name, type, buffer)
        char *name;
        char *type;
        snmdb_buffer *buffer;
```

DESCRIPTION

Before adding a new element into the database, you will need to call **snmdb_init_buffer** to initialize the buffer for the element, and also to set the name and type for the element. You may then call **snmdb_set_property**(3N) to set the values of any properties of the element, and then call **snmdb_add_to_view**(3N)**, snmdb_add_connection**(3N)**, snmdb_set_color**(3N)**,** or **snmdb_add_agent**(3N) to add other records describing the element. Finally, call **snmdb_add**(3N) to add the element into the database.

INPUT ARGUMENTS

name name of the element.

type type of the element.

buffer pointer to a structure of type **snmdb_buffer**.

RETURN VALUE

Returns 1 if successful. Otherwise, it returns 0, and the external variable **snm_error**(3N) is set to the error reason.

SEE ALSO

snmdb_set_property(3N)**, snmdb_add_to_view**(3N)**, snmdb_add_connection**(3N)**, snmdb_set_color**(3N)**, snmdb_add_agent**(3N)**, snmdb_add**(3N)

NAME

snmdb_lock – locks the database

SYNOPSIS

```
#include <netmgt/netmgt_db.h>

snmdb_lock(wait, clock)
        int wait;
        int clock;
```

DESCRIPTION

snmdb_lock locks the database so that entries can be changed. **snmdb_lock** returns a Boolean indicating whether the database could be locked. After the database updates are complete, unlock the database by using **snmdb_unlock**(3N)**.**

INPUT ARGUMENTS

wait Boolean indicating whether the process should wait if the database is locked by another process. If this argument is not set and the database is already locked, **snmdb_lock** returns 0, and **snm_error**(3N) indicates the reason for the error.

clock Boolean specifying whether the Console should display a clock to indicate that the database is locked.

RETURN VALUE

Returns 1 if database has been locked. Otherwise, it returns a 0. If the database has not been locked, the external variable **snm_error**(3N) is set to the error reason.

NOTE

Set the first argument to TRUE to cause the SNM Console to update its display if an element is added, modified, or deleted from the Console's current view. Otherwise, the Console user must leave the view and then return to the view in order to see the change.

SEE ALSO

snmdb_unlock(3N)

NAME

snmdb_open – open the database

SYNOPSIS

#include <netmgt/netmgt_db.h>

snmdb_open()

DESCRIPTION

snmdb_open should be called before any database operations. It opens the database and performs the necessary setup work. The database that is opened is the file **db.**<*name*>, where *name* is the name specified by the environment variable SNM_NAME, or the login username if SNM_NAME is not specified. The database must be located in the directory specified by the *database* keyword in the **/etc/opt/SUNWconn/snm/snm.conf** file for Solaris 2.x and the **/etc/snm.conf** file for Solaris 1.x, or the directory **/var/opt/SUNWconn/snm** for Solaris 2.x and **/var/adm/snm** for Solaris 1.x if there is no directory specified by the *database* keyword.

RETURN VALUE

Returns 1 if the database opened successfully; otherwise, it returns a 0.

NAME

snmdb_read – reads an existing element from the database into the specified internal buffer. This is necessary for retrieving or updating any properties of a particular element

SYNOPSIS

```
#include <netmgt/netmgt_db.h>

snmdb_read(name, buffer)
        char *name;
        snmdb_buffer *buffer;
```

DESCRIPTION

snmdb_read reads an existing element into the buffer. You can then read or modify properties using **snmdb_get_property**(3N) or **snmdb_set_property**(3N)**.** If you modify any properties, you will need to call **snmdb_update**(3N) in order to write the updated element back to the database.

INPUT ARGUMENTS

name name of the element.

buffer pointer to a structure of type **snmdb_buffer**.

RETURN VALUE

Returns 1 if successful. Otherwise, it returns 0, and the external variable **snm_error**(3N) is set to the error reason.

SEE ALSO

snmdb_get_property(3N)**, snmdb_set_property**(3N)**, snmdb_update**(3N)

NAME

snmdb_save_element – save an element record into the file specified in ASCII format

SYNOPSIS

```
include <netmgt/netmgt_db.h>

snmdb_save_element(buffer, fileptr)
        snmdb_buffer *buffer;
        FILE *fileptr;
```

DESCRIPTION

snmdb_save_element writes the contents in the element buffer into the file specified in ASCII format, which is the same format as the Console's **Save** function under the **File** button. The *buffer* should be loaded with an element by calling **snmdb_read**(3N)**.** The *fileptr* is the file pointer of the output file, which should be opened before calling **snmdb_save_element**. The user is responsible for validating the output file and closing the file after the saving is done completely.

INPUT ARGUMENTS

buffer pointer to a structure of type **snmdb_buffer**.

fileptr file pointer to the output file.

RETURN VALUE

Returns 1 if the element has been written successfully. Otherwise, it returns 0, and the external variable **snm_error**(3N) is set to the error reason.

SEE ALSO

snmdb_read(3N)

NAME

snmdb_set_color – sets the color of an element

SYNOPSIS

```
#include <netmgt/netmgt_db.h>

snmdb_set_color(buffer, red, green, blue)
        snmdb_buffer *buffer;
        int red;
        int green;
        int blue;
```

DESCRIPTION

snmdb_set_color changes the color value in RGB numbers of the element in the buffer. The colors are contained in the glyphColor record in the buffer. If the glyphColor record is not already defined in the buffer, a new one is created.

INPUT ARGUMENTS

buffer	pointer to a structure of type **snmdb_buffer**.
red	red color value of the element.
green	green color value of the element.
blue	blue color value of the element.

RETURN VALUE

Returns 1 if successful. Otherwise, it returns 0, and the external variable **snm_error**(3N) is set to the error reason.

SEE ALSO

snmdb_get_color(3N)**, snmdb_delete_color**(3N)

NAME

snmdb_set_property – sets the value of a particular property of an element

SYNOPSIS

```
#include <netmgt/netmgt_db.h>

snmdb_set_property(buffer, property, value)
        snmdb_buffer *buffer;
        char *property;
        snmdb_data value;
```

DESCRIPTION

If you are creating a new element, you should call **snmdb_init_buffer**(3N) to initialize the internal buffer of the element before you set the element's property. If you are modifying an existing element, you should call **snmdb_read**(3N) to read the element into the internal buffer before you make any change to the element's property. After you have modified the element, remember to call **snmdb_update**(3N) to write the modified element back to the database.

INPUT ARGUMENTS

buffer pointer to a structure of type **snmdb_buffer**.

property
name of the property describing the element.

value value of the property to be set to. The data type of this value should match the data type of the property. This parameter should not be the snmdb_data union structure. It should be the actual property type to set. If theproperty to set is of type integer, just pass the integer value.

RETURN VALUE

Returns 1 if successful. Otherwise, it returns 0, and the external variable **snm_error**(3N) is set to the error reason.

SEE ALSO

snmdb_read(3N)**, snmdb_update**(3N)**, snmdb_get_property**(3N)

NAME

snmdb_unlock – unlocks the database

SYNOPSIS

#include <netmgt/netmgt_db.h>

snmdb_unlock()

DESCRIPTION

snmdb_unlock unlocks the database and notifies the console to update its views.

INPUT ARGUMENTS

None

RETURN VALUE

Returns 1 if the database has been unlocked. Otherwise, it returns 0.

SEE ALSO

snmdb_lock(3N)

NAME

snmdb_update – update an existing element in the database

SYNOPSIS

```
#include <netmgt/netmgt_db.h>

snmdb_update(buffer)
        snmdb_buffer *buffer;
```

DESCRIPTION

snmdb_update writes the updated element back into the database.

INPUT ARGUMENTS

buffer pointer to a structure of type **snmdb_buffer**.

RETURN VALUE

Returns 1 if successful. Otherwise, it returns 0, and the external variable **snm_error**(3N) is set to the error reason.

SEE ALSO

snmdb_read(3N)**, snmdb_set_property**(3N)

NAME

snmp-mibII.schema – SNMP MIB-II schema.

DESCRIPTION

snmp-mibII.schema contains data definitions for MIB-II objects defined in RFC 1213. The SNMP proxy agent can use this schema to return information from SNMP agents that use the MIB-II data definitions.

The attribute definitions for this schema are shown below.

ATTRIBUTES

snmp-mibII.schema contains 16 attribute groups (**system, interfaces, ipStatus, ipInput, ipOutput, icmpInput, icmpOutput, tcp, udp, egp, transmission, snmpStatus, snmpInput, snmpInputErrors, snmpOutput, snmpOutputErrors**) and ten attribute tables (**ifStatus, ifInput, ifOutput, atTable, ipAddrTable, ipRoutingTable, ipNetToMediaTable, tcpConnTable, udpTable, egpNeighTable**).

The **system** group returns statistics about a particular RFC1213 system (e.g., a workstation or printer).

sysDescr - system description (string[128])

sysObjectId - vendor's object identifier (objectid)

sysUpTime - time, in hundredths of seconds, since last system restart (timeticks)

sysContact - name of person to contact

sysName - fully-qualified domain name of device

sysLocation - physical location of device

sysServices - services offered by device

The **interfaces** group reports the number of RFC1213 interfaces handled by the proxy agent.

ifNumber - number of network interfaces (int)

The **ifStatus** table reports status statistics about a particular nterface selector (int) RFC1213 interface, using *ifIndex* as the key. Interface table indices start with index number z

ifIndex - interface selector (int)

ifDescr - interface description (string[128])

ifType - interface type (int)

1 — other (none of the following)
2 — regular1822
3 — hdh1822
4 — ddn-x25
5 — rfc877-x25
6 — ethernet-csmacd
7 — iso88023-csmacd

8 — iso88024-tokenBus
9 — iso88025-tokenRing
10 — iso88026-man
11 — starLan
12 — proteon-10Mbit
13 — proteon-80Mbit
14 — hyperchannel
15 — fddi
16 — lapb
17 — sdlc
18 — t1-carrier
19 — cept (European equivalent of T-1)
20 — basicIsdn
21 — primaryIsdn
22 — propPointToPointSerial (proprietary serial)

ifMtu - maximum transmission unit (int)

ifSpeed - interface speed, in bits per second (gauge)

if1hysAddress - media physical address (octet[36])

ifAdminStatus - administrative status (int)

1 — up
2 — down
3 — testing

ifOperStatus - operational status (int)

1 — up
2 — down
3 — testing

ifLast Change - time, in hundredths of seconds, since operational state was entered (timeticks)

ifSpecific - references MIB definitions specific to interface

The **ifInput** table reports input statistics about a particular RFC1213 interface, using *ifIndex* as the key. Interface table indices start with index number zero.

ifIndex - interface selector (int)

ifInOctets - number of bytes received (counter)

ifInUcastPkts - number of unicast packets accepted (counter)

ifInNUcastPkts - number of non-unicast packets accepted (counter)

ifInDiscards - number of packets discarded (counter)

ifInerrors - number of malformed packets received (counter)

ifInUnknownProtos - number of packets of unknown protocol (counter)

The **ifOutput** table reports output statistics about a particular RFC1213 interface, using *ifIndex* as the key. Interface table indices start with index number zero.

ifIndex - interface selector (int)

ifOutOctets - number of octets sent (counter)

ifOutUcastPkts - number of unicast packets sent (counter)

ifOutNUcastPkts - number of non-unicast packets sent (counter)

ifOutDiscards - number of outbound packets discarded (counter)

ifOutErrors - number of output errors (counter)

ifOutQlen - length of output packet queue (gauge)

The **atTable** table reports statistics about the RFC1213 address resolution protocol (ARP) table. The ARP table key consists of the interface number, the constant value 1, and an IP address in dot notation. All fields must be separated by spaces or tabs. Note that the address translation group is marked deprecated in MIB-II.

atIf2ndex - interface for this entry (int)

atPhysAddress - media physical address (octet[36])

atNetAddress - network address (netaddress)

The **ipStatus** group reports status about the RFC1213 Internet Protocol (IP) group.

ipForwarding - IP forwarding (int)

1 — gateway
2 — host

ipDefaultTTL - default time-to-live (int)

The **ipInput** group reports input statistics about the RFC1213 Internet Protocol (IP) group.

ipInReceives - number of input datagrams (counter)

ipInHdrErrors - number of input datagrams discarded due to header errors (counter)

ipInAddrErrors - number of input datagrams discarded due to address errors (counter)

ipInUnknownProtos - number of input datagrams discarded due to unknown protocol (counter)

ipInDiscards - number of input datagrams discarded due to other reasons (counter)

ipInDelivers - number of datagrams delivered (counter)

ipInForwDatagrams - number of datagrams forwarded (counter)

ipReasmTimeout - IP reassembly timeout, in seconds (int)

ipReasmReqds - number of IP fragments received (counter)

ipReasmOKs - number of datagrams reassembled (counter)

ipReasmFails - number of reassembly failures (counter)

The **ipOutput** group reports output statistics about the RFC1213 Internet Protocol (IP) group.

ipOutRequests - number of output datagrams requested (counter)

ipOutDiscards - number of output datagrams discarded for other reasons (counter)

ipOutNoRoutes - number of output datagrams discarded for no route (counter)

ipFragOKs - number of datagrams fragmented (counter)

ipFragFails - number of datagrams for which fragmentation failed (counter)

ipFragCreates - number of fragments created (counter)

The **ipAddrTable** table reports statistics about the RFC 1213 IP address table. The IP address table key is a host or network IP address in dot notation.

ipAdEntAddr - IP address of this entry (netaddress)

ipAdEntIf2ndex - interface associated with this entry (int)

ipAdEntNetMask - subnet mask associated with this entry (int)

ipAdEntBcastAddr - IP broadcast address of this entry

ipAdEntReasmMaxSiz - maximum size of IP datagram that can be re-assembled by the entity

The **ipRoutingTable** table reports statistics about the RFC1213 IP routing table. The IP routing table key is a route or host IP address in dot notation.

ipRouteDest - destination IP address (netaddress)

ipRouteIf2ndex - interface to use (int)

ipRouteMetric1 - route metric 1 (int)

ipRouteMetric2 - route metric 2 (int)

ipRouteMetric3 - route metric 3 (int)

ipRouteMetric4 - route metric 4 (int)

ipRouteNextHop - next hop IP address (netaddress)

ipRouteType - type of route (int)

1 — other (none of the following)
2 — invalid (an invalid route)

3 — direct (route to a directly connected (sub-)network)
4 — remote (route to a non-local host, network, or sub-network)

ipRouteProto - routing protocol used (int)

1 — other (none of the following)
2 — local (non-protocol information, e.g., manually configured entries)
3 — netmgnt (set via a network management protocol)
4 — obtained via ICMP, e.g., Redirect
5 — egp
6 — ggp
7 — hello
8 — rip
9 — is-is
10 — es-is
11 — ciscoIgrp
12 — bbnSpf2grp
13 — oigrp

ipRouteAge - age of this route entry, in seconds (int)

ipRouteMask - subnet mask for route

The **ipNetToMediaTable** table maps IP addresses to physical addresses.

ipNetToMediaIf2ndex - interface number

ipNetToMediaPhysAddress - media address of mapping

ipNetToMediaNetAddress - IP address of mapping

ipNetToMediaType - type of mapping

1 — other (none of the following)
2 — invalid
3 — dynamic
4 — static

The **icmpInput** group reports input statistics about the RFC1213 ICMP group.

icmpInMsgs - number of ICMP messages received (counter)

icmpInErrors - number of ICMP messages received with errors (counter)

icmpInDestUnreachs - number of ICMP destination unreachables received (counter)

icmpInTimeExcds - number of ICMP time exceededs received (counter)

icmpInParmProbs - number of ICMP parameter problems received (counter)

icmpInSrcQuenchs - number of ICMP source quenches received (counter)

icmpInRedirects - number of ICMP redirects received (counter)

icmpInEchos - number of ICMP echo requests received (counter)

icmpInEchoReps - number of ICMP echo replies received (counter)

icmpInTimestamps - number of ICMP timestamp requests received (counter)

icmpInTimestampReps - number of ICMP timestamp replies received (counter)

icmpInAddrMasks - number of ICMP address mask requests received (counter)

icmpInAddrMaskReps - number of ICMP address mask replies received (counter)

The **icmpOutput** group reports output statistics about the RFC1213 ICMP group.

icmpOutMsgs - number of ICMP messages requested to be sent (counter)

icmpOutErrors - number of ICMP messages not sent due to errors (counter)

icmpOutDestUnreachs - number of ICMP destination unreachables sent (counter)

icmpOutTimeExcds - number of ICMP time exceededs sent (counter)

icmpOutParmProbs - number of ICMP parameter problems sent (counter)

icmpOutSrcQuenchs - number of ICMP source quenches sent (counter)

icmpOutRedirects - number of ICMP redirects sent (counter)

icmpOutEchos - number of ICMP echo requests sent (counter)

icmpOutEchoReps - number of ICMP echo replies sent (counter)

icmpOutTimestamps - number of ICMP timestamp requests sent (counter)

icmpOutTimestampReps - number of ICMP timestamp replies sent (counter)

icmpOutAddrMasks - number of ICMP address mask requests sent (counter)

icmpOutAddrMaskReps - number of ICMP address mask replies sent (counter)

The **tcp** group reports statistics about the RFC1213 TCP group.

tcpRtoAlgorithm - TCP round trip algorithm (int)

1 — other (none of the following)
2 — constant (a constant rto)
3 — rsre (see MIL-STD-1778, Appendix B)
4 — vanj (Van Jacobsen's algorithm)

tcpRtoMin - minimum round trip time, in milliseconds (int)

tcpRtoMax - maximum round trip time, in milliseconds (int)

tcpMaxConn - maximum number of TCP connections (int)

tcpActiveOpens - number of TCP connections actively open (counter)

tcpPassiveOpens - number of TCP connections passively open (counter)

tcpAttemptFails - number of failed connection attempts (counter)

tcpEstabResets - number of connection resets (counter)

tcpCurrEstab - number of TCP connections currently open (gauge)

tcpInSegs - number of segments received on TCP connections (counter)

tcpOutSegs - number of segments sent on TCP connections (counter)

tcpRetransSegs - number of TCP segments retransmitted (counter)

tcpInErrs - number of TCP segments discarded because of format error

tcpOutRsts - number of resets generated

The **tcpConnTable** table reports statistics about the RFC1213 TCP connection table. The TCP connection table key is a socket pair consisting of a local IP address expressed in dot notation, followed by a local port number, followed by a remote IP address in dot notation, followed by a remote port number. All fields must be separated by spaces or tabs.

tcpConnState - TCP connection state (int)

1 — closed
2 — listen
3 — synSent
4 — synReceived
5 — established
6 — finWait1
7 — finWait2
8 — closeWait
9 — lastAck
10 — closing
11 — timeWait

tcpConnLocalAddress - local IP address (netaddress)

tcpConnLocalPort - local TCP port (int)

tcpConnRemAddress - remote IP address (netaddress)

tcpConnRemPort - remote TCP port (int)

The **udp** group reports statistics about the RFC1213 UDP group.

udpInDatagrams - number of UDP datagrams received and delivered (counter)

udpNoPorts - number of UDP datagrams discarded due to no listen on local port (counter)

udpInErrors - number of UDP datagrams discarded due to other errors (counter)

udpOutDatagrams - number of UDP datagrams sent (counter)

The **udpTable** table reports UDP listener information.

udpLocalAddress - local IP address

udpLocalPort - local UDP port

The **egp** group reports statistics about the RFC1213 EGP group.

egpInMsgs - number of EGP messages received without error (counter)

egpInErrors - number of EGP messages received with error (counter)

egpOutMsgs - number of locally generated EGP messages (counter)

egpOutErrors - number of EGP messages not sent due to errors (counter)

egpAs - autonomous system number of entity (int)

The **egpNeighTable** table reports statistics about the RFC 1213 EGP neighbor table. The EGP neighbor key is a host IP address in dot notation.

egpNeighState - EGP state of neighbor (int)

egpNeighAddress - IP address of EGP neighbor (netaddress)

egpNeighAs - autonomous system number (int)

egpNeighInMsgs - number of EGP messages received without error (counter)

egpNeighInErrs - number of EGP messages received with error (counter)

egpNeighOutMsgs - number of locally-generated EGP messages (counter)

egpNeighOutErrs - number of locally-generated EGP messages not sent (counter)

egpNeighInErrMsgs - number of EGP error messages received (counter)

egpNeighOutErrMsgs - number of EGP error messages sent (counter)

egpNeighStateUps - number of EGP state transitions to the UP state (counter)

egpNeighStateDowns - number of EGP state transitions from the UP state (counter)

egpNeighIntervalHello - interval (in hundredths of a second) between EGP Hello command retransmissions (int)

egpNeighIntervalPoll - interval (in hundredths of a second) between EGP poll command retransmissions (int)

egpNeighMode - polling mode of EGP entity (int)

1 — active
2 — passive

egpNeighEventTrigger - trigger for operator-initiated Start and Stop events (int)

1 — start
2 — stop

The **transmission** group reports statistics about transmission media. Note that RFC 1213 does not define MIB objects in this group.

The **snmpStatus** group reports statistcis about an SNMP entity.

snmpEnableAuthTraps - indicates whether SNMP agent is permitted to generate authentication failure traps (int)

1 — enabled
2 — disabled

The **snmpInput** group reports input statistics for an SNMP entity.

snmpInPkts - number of messages delivered from transport service (counter)

snmpInTotalReqVars - number of MIB objects retrieved successfully (counter)

snmpInTotalSetVars - number of MIB objects altered successfully (counter)

snmpInGetRequests - number of SNMP Get-Request PDUs accepted and processed (counter)

snmpInGetNexts - number of SNMP Get-Next PDUs accepted and processed (counter)

snmpInSetRequests - number of SNMP Set-Request PDUs accepted and processed (counter)

snmpInGetResponses - number of SNMP Get-Response PDUs accepted and processed (counter)

snmpInTraps - number of SNMP Trap PDUs accepted and processed (counter)

The **snmpInputErrors** group reports statistics on input errors.

snmpInBadVersions - number of SNMP messages delivered for an unsupported SNMP version (counter)

snmpInBadCommunityNames - number of SNMP messages delivered which used an unknown SNMP community name (counter)

snmpInBadCommunity Uses - number of SNMP messages delivered which represented an operation not allowed by the SNMP community (counter)

snmpInASNParseErrs - number of ASN.1 or BER errors encountered when decoding received messages (counter)

snmpInBadTypes - reserved

snmpInTooBigs - number of SNMP PDUs delivered for which the error-status field is ‘tooBig’ (counter)

snmpInNoSuchNames - number of SNMP PDUs delivered for which the error-status field is ‘noSuchName’ (counter)

snmpInBadValues - number of SNMP PDUs delivered for which the error-status field is ‘badValue’ (counter)

snmpInReadOnlys - number of SNMP PDUs delivered for which the error-status field is ‘readOnly’ (counter)

snmpInGenErrs - number of SNMP PDUs delivered for which the error-status field is ‘genErr’ (counter)

The **snmpOutput** group reports output statistics for an SNMP entity.

snmpOutPkts - number of SNMP message passed to transport service (counter)

snmpOutGetRequests - number of SNMP Get-Request PDUS generated (counter)

snmpOutGetNexts - number of SNMP Get-Next PDUs generated (counter)

snmpOutSetRequests - number of SNMP Set-Request PDUs generated (counter)

snmpOutGetResponses - number of SNMP Get-Response PDUS generated (counter)

snmpOutTraps - number of SNMP Trap PDUs generated (counter)

The **snmpOutputErrors** group reports statistics on output errors.

snmpOutTooBigs - number of SNMP PDUS generated for which the error-status field is ‘tooBig’ (counter)

snmpOutNoSuchNames - number of SNMP PDUs generated for which the error-status field is ‘noSuchName’ (counter)

snmpOutBadValues - number of SNMP PDUs generated for which the error-status field is ‘badValue’ (counter)

snmpOutReadOnlys - reserved

snmpOutGenErrs - number of SNMP PDUs generated for which the error-status field is ‘genErr’ (counter)

NAME

snmp.schema – SNMP MIB-I schema.

DESCRIPTION

snmp.schema contains data definitions for MIB-I objects defined in RFC 1156. The SNMP proxy agent can use this schema to return information from SNMP agents that use the MIB-I data definitions.

The attribute definitions for this schema are shown below.

ATTRIBUTES

snmp.schema contains ten attribute groups (**system, interfaces, ipStatus, ipInput, ipOutput, icmpInput, icmpOutput, tcp, udp, egp**) and eight attribute tables (**ifStatus, ifInput, ifOutput, attable, ipaddrtable, iproutingtable, tcptable and egpneightable**).

The **system** group returns statistics about a particular RFC1156 system (e.g., a workstation or printer).

sysDescr - system description (string[128])

sysObjectId - vendor's object identifier (objectid)

sysUpTime - time, in hundredths of seconds, since last system restart (timeticks)

The **interfaces** group reports the number of RFC1156 interfaces handled by the proxy agent.

ifNumber - number of network interfaces (int)

The **ipStatus** group reports status about the RFC1156 Internet Protocol (IP) group.

ipForwarding - IP forwarding (int)

1 — gateway
2 — host

ipDefaultTTL - default time-to-live (int)

The **ipInput** group reports input statistics about the RFC1156 Internet Protocol (IP) group.

ipInReceives - number of input datagrams (counter)

ipInHdrErrors - number of input datagrams discarded due to header errors (counter)

ipInAddrErrors - number of input datagrams discarded due to address errors (counter)

ipInUnknownProtos - number of input datagrams discarded due to unknown protocol (counter)

ipInDiscards - number of input datagrams discarded due to other reasons (counter)

ipInDelivers - number of datagrams delivered (counter)

ipInForwDatagrams - number of datagrams forwarded (counter)

ipReasmTimeout - IP reassembly timeout, in seconds (int)

ipReasmReqds - number of IP fragments received (counter)

ipReasmOKs - number of datagrams reassembled (counter)

ipReasmFails - number of reassembly failures (counter)

The **ipOutput** group reports output statistics about the RFC1156 Internet Protocol (IP) group.

ipOutRequests - number of output datagrams requested (counter)

ipOutDiscards - number of output datagrams discarded for other reasons (counter)

ipOutNoRoutes - number of output datagrams discarded for no route (counter)

ipFragOKs - number of datagrams fragmented (counter)

ipFragFails - number of datagrams for which fragmentation failed (counter)

ipFragCreates - number of fragments created (counter)

The **icmpInput** group reports input statistics about the RFC1156 ICMP group.

icmpInMsgs - number of ICMP messages received (counter)

icmpInErrors - number of ICMP messages received with errors (counter)

icmpInDestUnreachs - number of ICMP destination unreachables received (counter)

icmpInTimeExcds - number of ICMP time exceededs received (counter)

icmpInParmProbs - number of ICMP parameter problems received (counter)

icmpInSrcQuenchs - number of ICMP source quenches received (counter)

icmpInRedirects - number of ICMP redirects received (counter)

icmpInEchos - number of ICMP echo requests received (counter)

icmpInEchoReps - number of ICMP echo replies received (counter)

icmpInTimestamps - number of ICMP timestamp requests received (counter)

icmpInTimestampReps - number of ICMP timestamp replies received (counter)

icmpInAddrMasks - number of ICMP address mask requests received (counter)

icmpInAddrMaskReps - number of ICMP address mask replies received (counter)

The **icmpOutput** group reports output statistics about the RFC1156 ICMP group.

icmpOutMsgs - number of ICMP messages requested to be sent (counter)

icmpOutErrors - number of ICMP messages not sent due to errors (counter)

icmpOutDestUnreachs - number of ICMP destination unreachables sent (counter)

icmpOutTimeExcds - number of ICMP time exceededs sent (counter)

icmpOutParmProbs - number of ICMP parameter problems sent (counter)

icmpOutSrcQuenchs - number of ICMP source quenches sent (counter)

icmpOutRedirects - number of ICMP redirects sent (counter)

icmpOutEchos - number of ICMP echo requests sent (counter)

icmpOutEchoReps - number of ICMP echo replies sent (counter)

icmpOutTimestamps - number of ICMP timestamp requests sent (counter)

icmpOutTimestampReps - number of ICMP timestamp replies sent (counter)

icmpOutAddrMasks - number of ICMP address mask requests sent (counter)

icmpOutAddrMaskReps - number of ICMP address mask replies sent (counter)

The **tcp** group reports statistics about the RFC1156 TCP group.

tcpRtoAlgorithm - TCP round trip algorithm (int)

1 — other (none of the following)
2 — constant (a constant rto)
3 — rsre (see MIL-STD-1778, Appendix B)
4 — vanj (Van Jacobsen's algorithm)

tcpRtoMin - minimum round trip time, in milliseconds (int)

tcpRtoMax - maximum round trip time, in milliseconds (int)

tcpMaxConn - maximum number of TCP connections (int)

tcpActiveOpens - number of TCP connections actively open (counter)

tcpPassiveOpens - number of TCP connections passively open (counter)

tcpAttemptFails - number of failed connection attempts (counter)

tcpEstabResets - number of connection resets (counter)

tcpCurrEstab - number of TCP connections currently open (gauge)

tcpInSegs - number of segments received on TCP connections (counter)

tcpOutSegs - number of segments sent on TCP connections (counter)

tcpRetransSegs - number of TCP segments retransmitted (counter)

The **udp** group reports statistics about the RFC1156 UDP group.

udpInDatagrams - number of UDP datagrams received and delivered (counter)

udpNoPorts - number of UDP datagrams discarded due to no listen on local port (counter)

udpInErrors - number of UDP datagrams discarded due to other errors (counter)

udpOutDatagrams - number of UDP datagrams sent (counter)

The **egp** group reports statistics about the RFC1156 EGP group.

egpInMsgs - number of EGP messages received without error (counter)

egpInErrors - number of EGP messages received with error (counter)

egpOutMsgs - number of locally generated EGP messages (counter)

egpOutErrors - number of EGP messages not sent due to errors (counter)

The **ifStatus** table reports status statistics about a particular RFC1156 interface, using *ifIndex* as the key. Interface table indices start with index number zero.

netmgt_table_key - the OBJECT IDENTIFIER suffix indicating the table key. Dot (".") separators are replaced by a single blank character.

ifIndex - interface selector (int)

ifDescr - interface description (string[128])

ifType - interface type (int)

1 — other (none of the following)
2 — regular1822
3 — hdh1822
4 — ddn-x25
5 — rfc877-x25
6 — ethernet-csmacd
7 — iso88023-csmacd
8 — iso88024-tokenBus
9 — iso88025-tokenRing
10 — iso88026-man
11 — starLan
12 — proteon-10Mbit
13 — proteon-80Mbit
14 — hyperchannel
15 — fddi
16 — lapb
17 — sdlc
18 — t1-carrier
19 — cept (European equivalent of T-1)
20 — basicIsdn
21 — primaryIsdn
22 — propPointToPointSerial (proprietary serial)

ifMtu - maximum transmission unit (int)

ifSpeed - interface speed, in bits per second (gauge)

if1hysAddress - media physical address (octet[36])

ifAdminStatus - administrative status (int)

1 — up
2 — down
3 — testing

ifOperStatus - operational status (int)

1 — up
2 — down
3 — testing

ifLast Change - time, in hundredths of seconds, since operational state was entered (timeticks)

The **ifInput** table reports input statistics about a particular RFC1156 interface, using *ifIndex* as the key. Interface table indices start with index number zero.

netmgt_table_key - the OBJECT IDENTIFIER suffix indicating the table key. Dot (".") separators are replaces by a single blank character.

ifIndex - interface selector (int)

ifInOctets - number of bytes received (counter)

ifInUcastPkts - number of unicast packets accepted (counter)

ifInNUcastPkts - number of non-unicast packets accepted (counter)

ifInDiscards - number of packets discarded (counter)

ifInerrors - number of malformed packets received (counter)

ifInUnknownProtos - number of packets of unknown protocol (counter)

The **ifOutput** table reports output statistics about a particular RFC1156 interface, using *ifIndex* as the key. Interface table indices start with index number zero.

netmgt_table_key - the OBJECT IDENTIFIER suffix indicating the table key. Dot (".") separators are replaces by a single blank character.

ifIndex - interface selector (int)

ifOutOctets - number of octets sent (counter)

ifOutUcastPkts - number of unicast packets sent (counter)

ifOutNUcastPkts - number of non-unicast packets sent (counter)

ifOutDiscards - number of outbound packets discarded (counter)

ifOutErrors - number of output errors (counter)

ifOutQlen - length of output packet queue (gauge)

The **attable** table reports statistics about the RFC1156 address resolution protocol (ARP) table. The ARP table key consists of the interface number, the constant value 1, and an IP address in dot notation. All fields must be separated by spaces or tabs.

netmgt_table_key - the OBJECT IDENTIFIER suffix indicating the table key. Dot (".") separators are replaces by a single blank character.

atIfIndex - interface for this entry (int)

atPhysAddress - media physical address (octet[36])

atNetAddress - network address (netaddress)

The **ipaddrtable** table reports statistics about the RFC 1156 IP address table. The IP address table key is a host or network IP address in dot notation.

netmgt_table_key - the OBJECT IDENTIFIER suffix indicating the table key. Dot (".") separators are replaces by a single blank character.

ipAdEntAddr - IP address of this entry (netaddress)

ipAdEntIfIndex - interface associated with this entry (int)

ipAdEntNetMask - subnet mask associated with this entry (int)

ipAdEntBcastAddr - IP broadcast address of this entry

The **iproutingtable** table reports statistics about the RFC1156 IP routing table. The IP routing table key is a route or host IP address in dot notation.

netmgt_table_key - the OBJECT IDENTIFIER suffix indicating the table key. Dot (".") separators are replaces by a single blank character.

ipRouteDest - destination IP address (netaddress)

ipRouteIfIndex - interface to use (int)

ipRouteMetric1 - route metric 1 (int)

ipRouteMetric2 - route metric 2 (int)

ipRouteMetric3 - route metric 3 (int)

ipRouteMetric4 - route metric 4 (int)

ipRouteNextHop - next hop IP address (netaddress)

ipRouteType - type of route (int)

1 — other (none of the following)
2 — invalid (an invalid route)
3 — direct (route to a directly connected (sub-)network)
4 — remote (route to a non-local host, network, or sub-network)

ipRouteProto - routing protocol used (int)

1 — other (none of the following)
2 — local (non-protocol information, e.g., manually configured entries)
3 — netmgnt (set via a network management protocol)
4 — obtained via ICMP, e.g., Redirect
5 — egp

6 — ggp
7 — hello
8 — rip
9 — is-is
10 — es-is
11 — ciscoIgrp
12 — bbnSpfIgrp
13 — oigrp

ipRouteAge - age of this route entry, in seconds (int)

The **tcptable** table reports statistics about the RFC1156 TCP connection table. The TCP connection table key is a socket pair consisting of a local IP address expressed in dot notation, followed by a local port number, followed by a remote IP address in dot notation, followed by a remote port number. All fields must be separated by spaces or tabs.

netmgt_table_key - the OBJECT IDENTIFIER suffix indicating the table key. Dot (".") separators are replaces by a single blank character.

tcpConnState - TCP connection state (int)

1 — closed
2 — listen
3 — synSent
4 — synReceived
5 — established
6 — finWait1
7 — finWait2
8 — closeWait
9 — lastAck
10 — closing
11 — timeWait

tcpConnLocalAddress - local IP address (netaddress)

tcpConnLocalPort - local TCP port (int)

tcpConnRemAddress - remote IP address (netaddress)

tcpConnRemPort - remote TCP port (int)

The **egpneightable** table reports statistics about the RFC 1156 EGP neighbor table. The egp neighbor key is a host IP address in dot notation.

netmgt_table_key - the OBJECT IDENTIFIER suffix indicating the table key. Dot (".") separators are replaces by a single blank character.

egpNeighState - EGP state of neighbor (int)

egpNeighAddress - IP address of EGP neighbor (netaddress)

NOTES

The schema restricts each group/table to sixteen attributes because some SNMP agent implementations cannot handle a request for more attributes at once.

NAME

snmp_set – perform an SNMP set request on a device

SYNOPSIS

snmp_set -h *hostname* -c *community-name* [-t *timeout*] -v "*variable-name type value ...*"

DESCRIPTION

snmp_set performs an SNMP *set* on a device.

OPTIONS

Each option is recognized as a separate argument.

-c *community-name*
community name for access to the SNMP entity

-h *hostname*
hostname of the SNMP entity containing the variables to be set.

-t *timeout*
number of seconds before the *set* request times out if a reply is not received from the SNMP entity. If this option is not specified, the timeout is 30 seconds.

-v *variable-name type value ...*
a triplet for each variable to be *set*, enclosed in double quotes. The format of the triplet is:

1. variable name, in dot notation
2. variable type, as a single letter option:
   -i integer
3. the value to give to the variable name.

EXAMPLES

Set the administrative status of interface **1** on **golden** to **2 (down)**, using community name **private.**

snmp_set -c private -h golden -v "ifAdminStatus.1 -i 2"

NOTES

snmp_set currently supports integer values only.

NAME

snmpd - Sun SNMP Agent

SYNOPSIS

snmpd [**–r**] [**–p** *port*] [**–a**] [**–c** *config-file*] [**–T** *trace-level*]

DESCRIPTION

snmpd is an RFC 1157-compliant SNMP agent. **snmpd** supports MIB-II as defined in RFC 1213, with Sun extensions under Sun's enterprise number. The MIB (Management Information Base) is both readable and writable. **snmpd** supports all SNMP protocol operations including GET-REQUEST, GETNEXT-REQUEST, SET-REQUEST, GET-REPLY, and TRAP.

snmpd supports the coldStart, linkUp, linkDown, and authentication traps. The authentication trap may be disabled by a command-line switch, which itself may be overridden by a management station writing to a MIB variable in the standard SNMP MIB group.

snmpd supports four distinct views of the MIB. The view used for any request is determined by the community string contained in that request.

To enhance security, **snmpd** supports an option to block all writes to the MIB. You can also limit the set of management stations from which the agent will accept requests in the configuration file used when starting the **snmpd**. See the SECURITY section below for more information.

Unless overridden, **snmpd** uses UDP port 161, the standard SNMP port. **snmpd** issues traps through the same port on which it receives SNMP requests.

snmpd must run with ''root'' privileges and is typically started at system startup via **/etc/rc.local**. **snmpd** may not be started using **inetd .** When started, **snmpd** detaches itself from the keyboard, disables all signals except SIGKILL, SIGILL, SIGUSR1, and SIGUSR2, and places itself in the background.

OPTIONS

The following options are recognized:

-r places the MIB into read-only mode.

–**p***port* defines an alternative UDP port on which **snmpd** listens for incoming requests. The default is UDP port 161.

-a disables the generation of authentication traps. However, an SNMP manager may write a value into snmpEnableAuthenTraps to enable or disable authentication traps.

–**c***config-file*
defines a configuration file that is read when the agent starts up. If a configuration file is not specified, the file **/etc/opt/SUNWconn/snm/snmpd.conf** for Solaris 2.x or the file **/etc/snmpd.conf** for Solaris 1.x is used.

–**T***trace-level*
sets trace levels. A value of 0 disables all tracing and is the default. Levels 1 through 3 represent increasing levels of trace output. When **snmpd** receives the signal SIGUSR1, it resets the trace-level to 0. When **snmpd** receives the signal SIGUSR2, it increments the trace-level by one.

Trace output is sent to the standard output in effect at the time **snmpd** is started.

No matter what trace level is in effect, certain significant events are logged in the system log.

FILES

The following files are provided for agent operation:

snmpd	agent binary
snmpd.conf	configuration information (described below)
sun-snmp.schema	SNM schema generated by **mib2schema** utility from MIB
sun.mib	Sun-specific MIB in ASN.1 format
sun-snmp.oid	contains the object identifiers referenced by MIB-II and Sun-specific OIDs

These files, as delivered, are located in the **/opt/SUNWconn/snm/agents** directory for Solaris 2.x and in the **/usr/snm/agents** directory for Solaris 1.x. Upon installation, **snmpd** is usually placed in **/opt/SUNWconn/snm/agents** for Solaris 2.x and in **/usr/snm/agents** for Solaris 1.x. In addition, **snmpd.conf** is usually placed in **/etc/opt/SUNWconn/snm** for Solaris 2.x and in **/etc** for Solaris 1.x.

The **snmpd.conf** file is used for configuration information. Each entry in the file consists of a keyword followed by a parameter string. The keyword must begin in the first position. Parameters are separated from the keyword and from one another by white space. Case in keywords is ignored. Each entry must be contained on a single line. All text following (and including) a pound sign (#) is ignored. Keywords currently supported are:

sysdescr
The value to be used to answer queries for sysDescr.

syscontact
The value to be used to answer queries for sysContact.

syslocation
The value to be used to answer queries for sysLocation.

trap The parameter names one or more hosts to receive traps. Only five hosts may be listed.

system-group-read-community
The community name to get read access to the system group and Sun's extended system group.

system-group-write-community
The community name to get write access to the system group and Sun's extended system group.

read-community
The community name to get read access to the entire MIB.

write-community
The community name to get write access to the entire MIB (implies read access).

trap-community
The community name to be used in traps.

kernel-file

The name of the file to use for kernel symbols.

managers
The names of hosts that may send SNMP queries. Only five hosts may be listed on any one line. This keyword may be repeated for a total of 32 hosts.

newdevice
The additional devices which are not built in SNMPD. The format is as follows:

newdevice type speed name

where *newdevice* is the keyword, *type* is an integer which has to match your schema file, *speed* is the new device's speed, and *name* is this new device's name.

An example **snmpd.conf** file is shown below:

```
sysdescr       Sun SNMP Agent, SPARCstation 10, Company Property Number 123456
syscontact     Cliff Claven
sysLocation    Stool next to Norms at Cheers
#
system-group-read-community    public
system-group-write-community   private
#
read-community  all_public
write-community all_private
#
trap          localhost
trap-community  SNMP-trap
#
#kernel-file    /vmunix
#
managers       lvs golden
managers       swap
```

INSTALLATION

snmpd and its configuration file (**snmpd.conf**) may be placed in any directory. However, **/opt/SUNWconn/snm/agents** for Solaris 2.x or **/usr/snm/agents** for Solaris 1.x is suggested for **snmpd** itself and **/etc/opt/SUNWconn/snm** (Solaris 2.x) or **/etc** (Solaris 1.x) for the configuration file. You should modify the configuration file as appropriate. If you make any changes to **snmpd.conf** file keyword values, you must kill and restart **snmpd** for the changes to take effect.

Your **/etc/services** file (or NIS equivalent) should contain the following entries:

```
snmp            161/udp                         # Simple Network Mgmt Protocol
snmp-trap       162/udp         snmptrap        # SNMP trap (event) messages
```

Following is an example for Solaris 2.x:

```
#
# Start the SNMP agent
#
```

```
if [ -f /etc/opt/SUNWconn/snm/snmpd.conf -a -x /opt/SUNWconn/snm/agents/snmpd ];
then
    /opt/SUNWconn/snm/agents/snmpd -c /etc/opt/SUNWconn/snm/snmpd.conf &&
    echo 'Starting SNMP-agent.'
```

Following is an example for Solaris 1.x:

```
#
# Start the SNMP agent
#

if [ -f /etc/snmpd.conf -a -x /usr/snm/agents/snmpd ]; then
    /usr/snm/agents/snmpd -c /etc/snmpd.conf &&
    echo 'Starting SNMP-agent.'
```

(Note that you need not explicitly place **snmpd** into the background. Also note that **snmpd** may not be started by **inetd .)**

SECURITY

SNMP, as presently defined, offers relatively little security. **snmpd** accepts requests from other machines, which can have the effect of disabling the network capabilities of your computer. To limit the risk, the configuration file lets you specify a list of up to 32 manager stations from which **snmpd** will accept requests. If you do not specify any such manager stations, **snmpd** accepts requests from anywhere.

snmpd also allows you to mark the MIB as ‘‘read-only,’’ by using the **-r** option.

Finally, **snmpd** supports four different community strings. These strings, however, are visible in the configuration file and within the SNMP packets as they flow on the network.

The configuration file should be owned by, and readable only by "root". In other words the mode should be:

For Solaris 2.x:

```
-rw-------  1 root       2090 Oct 17 15:04 /etc/opt/SUNWconn/snm/agent.conf
```

For Solaris 1.x:

```
-rw-------  1 root       2090 Oct 17 15:04 /etc/agent.conf
```

MIB

This section discusses some of the differences between the **snmpd** MIB and the standard MIB-II (as defined in RFC 1213).

The following variables are read-only in the **snmpd** MIB:

```
sysName
atIfIndex
ipDefaultTTL
```

These variables are read-write in the standard MIB-II.

The **snmpd** MIB Address Translation tables support limited write access: only atPhysAddress may be written, either to change the physical address of an existing entry or to delete an entire ARP table entry.

The **snmpd** MIB IP Net to Media table supports limited write access: only ipNetToMediaPhysAddress and ipNetToMediaType may be written, either to change the physical address of an existing entry or to delete an entire ARP table entry.

The following variables are read-write in the **snmpd** MIB; however, these variables have fixed values. Any new values ''set'' to them are accepted, but have no effect:

ipRoutIfIndex
ipRouteMetric1
ipRouteMetric2
ipRouteMetric3
ipRouteMetric4
ipRouteType
ipRouteAge
ipRouteMask
ipRouteMetric5

The following **snmpd** MIB variable reflects the actual state of the related table entry. ''Sets'' are accepted but have no effect:

tcpConnState

The following **snmpd** MIB variables are readable, but return a fixed value:

```
ifInOctets          / / Returns 0
ifInNUcastPkts  / / Returns 0
ifInDiscards      / / Returns 0
ifInUnknownProtos       / / Returns 0
ifOutOctets       / / Returns 0
ifOutNUcastPkts           / / Returns 0
ifOutDiscards    / / Returns 0
ipAdEntBcastAddr        / / Returns 1
ipAdEntReasmMaxSiz   / / Returns 65535
ipRouteMetric1  / / Returns -1
ipRouteMetric2  / / Returns -1
ipRouteMetric3  / / Returns -1
ipRouteMetric4  / / Returns -1
ipRouteAge       / / Returns 0
ipRouteMetric5  / / Returns -1
ipNetToMediaType        / / Returns (3) dynamic
ipRoutingDiscards         / / Returns 0
udpInDatagrams           / / Returns 0
udpNoPorts       / / Returns 0
```

udpOutDatagrams / / Returns 0

ATTRIBUTES

The following describes the attributes in the group and table definitions in the **sun.schema** file.

The **system** group reports statistics about a particular system (for example, a workstation or a printer).

sysDescr -
A textual description of the entity. This value should include the full name and version identification of the system's hardware type, software operating-system, and networking software. This value must only contain printable ASCII characters. (string[255])

sysObjectID -
The vendor's authoritative identification of the network management subsystem contained in the entity. This value is allocated within the SMI enterprises subtree (1.3.6.1.4.1) and provides an easy and unambiguous means for determining what type of equipment is being managed. For example, if vendor ''Flintstones, Inc.'' was assigned the subtree 1.3.6.1.4.1.4242, it could assign the identifier 1.3.6.1.4.1.4242.1.1 to its ''Fred Router.'' (objectid)

sysUpTime -
Time (in hundredths of a second) since the network management portion of the system was last reinitialized. (timeticks)

sysContact -
The textual identification of the contact person for this managed node, together with information on how to contact this person. (string[255])

sysName -
An administratively-assigned name for this managed node. By convention, this is the node's fully-qualified domain name. (string[255])

sysLocation -
The physical location of this node (for example, ''telephone closet, 3rd floor''). (string[255])

sysServices -
A value indicating the set of services that this entity primarily offers. (int)

The value is a sum. This sum initially takes the value zero. Then, for each layer L in the range 1 through 7 for which this node performs transactions, 2 raised to (L - 1) is added to the sum. For example, a node that performs primarily routing functions would have a value of 4 (2^(3-1)). In contrast, a node that is a host offering application services would have a value of 72 (2^(4-1) + 2^(7-1)). Note that in the context of the Internet suite of protocols, values should be calculated accordingly:

layer	functionality
1	physical (such as repeaters)

2 datalink/subnetwork (such as bridges)
3 internet (such as IP gateways)
4 end-to-end (such as IP hosts)
7 applications (such as mail relays)

For systems including OSI protocols, Layers 5 and 6 may also be counted.

The **interfaces** group reports the number of interfaces handled by the agent.

ifNumber - The number of network interfaces, regardless of their current state, present on this system. (int)

ifTable is a table of interface entries. The number of entries is given by the value of ifNumber.

ifIndex - A unique value for each interface. Its value ranges between 1 and the value of ifNumber. The value for each interface must remain constant at least from one reinitialization of the entity's network management system to the next reinitialization. (int)

ifDescr - A textual string containing information about the interface. This string should include the name of the manufacturer, the product name, and the version of the hardware interface. (string[255])

ifType - The type of interface, distinguished according to the physical/link protocol(s) immediately below the network layer in the protocol stack. (enum)

ifMtu - The size of the largest datagram that can be sent/received on the interface, specified in octets. For interfaces used for transmitting network datagrams, this is the size of the largest network datagram that can be sent on the interface. (int)

ifSpeed - An estimate of the interface's current bandwidth in bits-per-second. For interfaces that do not vary in bandwidth, or for those where no accurate estimation can be made, this object should contain the nominal bandwidth. (gauge)

if1hysAddress - The interface's address at the protocol layer immediately below the network layer in the protocol stack. For interfaces without such an address (for example, a serial line), this object should contain an octet string of zero length. (octet[128])

ifAdminStatus - The desired state of the interface. The testing(3) state indicates that no operational packets can be passed. (enum)

if OperStatus - The current operational state of the interface. The testing(3) state indicates that no operational packets can be passed. (enum)

ifLastChange - The value of sysUpTime at the time the interface entered its current operational state. If the current state was entered prior to the last reinitialization of the local network management subsystem, then this object contains a zero value. (timeticks)

ifInOctets - The total number of octets received on the interface, including framing characters. (counter) Returns a fixed value of 0.

ifInUcastPkts - The number of subnetwork-unicast packets delivered to a higher-layer protocol. (counter)

ifInNUcastPkts - The number of non-unicast (that is, subnetwork- broadcast or subnetwork-multicast) packets delivered to a higher-layer protocol. (counter) Returns a fixed value of 0.

ifInDiscards - The number of inbound packets chosen to be discarded, even though no errors had been detected to prevent their being deliverable to a higher-layer protocol. One possible reason for discarding such a packet could be to free up buffer space. (counter) Returns a fixed value of 0.

ifInErrors - The number of inbound packets that contained errors preventing them from being deliverable to a higher-layer protocol. (counter)

ifInUnknownProtos - The number of packets received via the interface that were discarded because of an unknown or unsupported protocol. (counter) Returns a fixed value of 0.

ifOutOctets - The total number of octets transmitted out of the interface, including framing characters. (counter) Returns a fixed value of 0.

ifOutUcastPkts - The total number of packets that higher-level protocols requested be transmitted to a subnetwork-unicast address, including those that were discarded or not sent. (counter)

ifOutNUcastPkts - The total number of packets that higher-level protocols requested be transmitted to a non- unicast (that is, a subnetwork-broadcast or subnetwork-multicast) address, including those that were discarded or not sent. (counter) Returns a fixed value of 0.

ifOutDiscards - The number of outbound packets that were chosen to be discarded even though no errors had been detected to prevent their being transmitted. One possible reason for discarding such a packet could be to free up buffer space. (counter) Returns a fixed value of 0.

ifOutErrors - The number of outbound packets that could not be transmitted because of errors. (counter)

ifOutQLen - The length of the output packet queue (in packets). (gauge)

ifSpecific - A reference to MIB definitions specific to the particular media being used to realize the interface. For example, if the interface is realized by an Ethernet, then the value of this object refers to a document defining objects specific to Ethernet. If this information is not present, its value should be set to the OBJECT IDENTIFIER { 0 0 }, which is a syntactically valid object identifier. Any conformant implementation of ASN.1 and BER must be able to generate and recognize this value. (objectid)

atTable Address Translation tables contain the NetworkAddress to physical address equivalences. Some interfaces do not use translation tables for determining address equivalences (for example, DDN-X.25 has an algorithmic method). If all interfaces are of

this type, then the Address Translation table is empty, that is, has zero entries.

atIfIndex - The interface on which this entry's equivalence is effective. The interface identified by a particular value of this index is the same interface as identified by the same value of ifIndex. (int)

atPhysAddress - The media-dependent physical address. (octet[128])

Setting this object to a null string (one of zero length) has the effect of invaliding the corresponding entry in the atTable object. That is, it effectively dissociates the interface identified with said entry from the mapping identified with said entry. It is an implementation-specific matter as to whether the agent removes an invalidated entry from the table. Accordingly, management stations must be prepared to receive tabular information from agents that corresponds to entries not currently in use. Proper interpretation of such entries requires examination of the relevant atPhysAddress object.

atNetAddress - The NetworkAddress (that is, the IP address) corresponding to the media-dependent physical address. (netaddress)

The **ip** group reports statistics about the Internet Protocol (IP) group.

ipForwarding - The indication of whether this entity is acting as an IP gateway in respect to the forwarding of datagrams received by, but not addressed to, this entity. IP gateways forward datagrams. IP hosts do not— except those source-routed via the host. (enum)

Note that for some managed nodes, this object may take on only a subset of the values possible. Accordingly, it is appropriate for an agent to return a ''bad-Value'' response if a management station attempts to change this object to an inappropriate value.

ipDefaultTTL - The default value inserted into the Time-To-Live field of the IP header of datagrams originated at this entity, whenever a TTL value is not supplied by the transport layer protocol. (int)

ipInReceives - The total number of input datagrams received from interfaces, including those received in error. (counter)

ipInHdrErrors - The number of input datagrams discarded due to errors in their IP headers, including bad checksums, version number mismatch, other format errors, time-to-live exceeded, errors discovered in processing their IP options, and so on. (counter)

ipInAddrErrors - The number of input datagrams discarded because the IP address in their IP header's destination field was not a valid address to be received at this entity. This count includes invalid addresses (for example, 0.0.0.0) and addresses of unsupported Classes (for example, Class E). For entities that are not IP Gateways and therefore do not forward datagrams, this counter includes datagrams discarded because the destination address was not a local address. (counter)

ipForwDatagrams - The number of input datagrams for which this entity was

not their final IP destination, as a result of which an attempt was made to find a route to forward them to that final destination. In entities that do not act as IP Gateways, this counter will include only those packets that were Source-Routed via this entity, and the Source- Route option processing was successful. (counter)

ipInUnknownProtos - The number of locally-addressed datagrams received successfully but discarded because of an unknown or unsupported protocol. (counter)

ipInDiscards - The number of input IP datagrams for which no problems were encountered to prevent their continued processing, but which were discarded, for example, for lack of buffer space. Note that this counter does not include any datagrams discarded while awaiting reassembly. (counter)

ipInDelivers - The total number of input datagrams successfully delivered to IP user-protocols (including ICMP). (counter)

ipOutRequests - The total number of IP datagrams that local IP user-protocols (including ICMP) supplied to IP in requests for transmission. Note that this counter does not include any datagrams counted in ipForwDatagrams. (counter)

ipOutDiscards - The number of output IP datagrams for which no problem was encountered to prevent their transmission to their destination, but which were discarded (for example, for lack of buffer space). Note that this counter would include datagrams counted in ipForwDatagrams if any such packets met this (discretionary) discard criterion. (counter)

ipOutNoRoutes - The number of IP datagrams discarded because no route could be found to transmit them to their destination. Note that this counter includes any packets counted in ipForwDatagrams which meet this 'no-route' criterion. Note that this includes any datagrams that a host cannot route because all its default gateways are down. (counter)

ipReasmTimeout - The maximum number of seconds that received fragments are held while they are awaiting reassembly at this entity. (int)

ipReasmReqds - The number of IP fragments received that needed to be reassembled at this entity. (counter)

ipReasmOKs - The number of IP datagrams successfully reassembled. (counter)

ipReasmFails - The number of failures detected by the IP reassembly algorithm, for whatever reason: timed out, errors, and the like. Note that this is not necessarily a count of discarded IP fragments since some algorithms (notably the algorithm in RFC 815) can lose track of the number of fragments by combining them as they are received. (counter)

ipFragOKs - The number of IP datagrams that have been successfully fragmented at this entity. (counter)

ipFragFails - The number of IP datagrams that have been discarded because they needed to be fragmented at this entity but could not be, for example, because

their ''Don't Fragment'' flag was set. (counter)

ipFragCreates - The number of IP datagram fragments that have been generated as a result of fragmentation at this entity. (counter)

ipRoutingDiscards - The number of routing entries that were chosen to be discarded even though they were valid. One possible reason for discarding such an entry could be to free-up buffer space for other routing entries. (counter) Returns a fixed value of 0.

ipAddrTable is a table of addressing information relevant to this entity's IP addresses.

ipAdEntAddr - The IP address to which this entry's addressing information pertains. (netaddress)

ipAdEntIfIndex - The index value that uniquely identifies the interface to which this entry is applicable. The interface identified by a particular value of this index is the same interface as identified by the same value of ifIndex. (int)

ipAdEntNetMask - The subnet mask associated with the IP address of this entry. The value of the mask is an IP address with all the network bits set to 1, and all the hosts bits set to 0. (netaddress)

ipAdEntBcastAddr - The value of the least-significant bit in the IP broadcast address used for sending datagrams on the (logical) interface associated with the IP address of this entry. For example, when the Internet standard all-ones broadcast address is used, the value will be 1. This value applies to both the subnet and network broadcasts addresses used by the entity on this (logical) interface. (int) Returns a fixed value of 1.

ipAdEntReasmMaxSize - The size of the largest IP datagram that this entity can reassemble from incoming IP fragmented datagrams received on this interface. (int) Returns a fixed value of 65535.

ipRouteTable is this entity's IP Routing table.

ipRouteDest - The destination IP address of this route. An entry with a value of 0.0.0.0 is considered a default route. Multiple routes to a single destination can appear in the table, but access to such multiple entries is dependent on the table-access mechanisms defined by the network management protocol in use. (netaddress)

ipRouteIfIndex - The index value that uniquely identifies the local interface through which the next hop of this route should be reached. The interface identified by a particular value of this index is the same interface as identified by the same value of ifIndex. (int)

ipRouteMetric1 - The primary routing metric for this route. The semantics of this metric are determined by the routing-protocol specified in the route's ipRouteProto value. If this metric is not used, its value should be set to -1. (int) Returns a fixed value of -1.

ipRouteMetric2 - An alternate routing metric for this route. The semantics of this metric are determined by the routing-protocol specified in the route's

ipRouteProto value. If this metric is not used, its value should be set to -1. (int) Returns a fixed value of -1.

ipRouteMetric3 - An alternate routing metric for this route. The semantics of this metric are determined by the routing-protocol specified in the route's ipRouteProto value. If this metric is not used, its value should be set to -1. (int) Returns a fixed value of -1.

ipRouteMetric4 - An alternate routing metric for this route. The semantics of this metric are determined by the routing-protocol specified in the route's ipRouteProto value. If this metric is not used, its value should be set to -1. (int) Returns a fixed value of -1.

ipRouteNextHop - The IP address of the next hop of this route. (In the case of a route bound to an interface that is realized via a broadcast media, the value of this field is the agent's IP address on that interface.) (netaddress)

ipRouteType - The type of route. Note that the values direct (3) and indirect (4) refer to the notion of direct and indirect routing in the IP architecture. (enum)

Setting this object to the value invalid (2) has the effect of invalidating the corresponding entry in the ipRouteTable object. That is, it effectively dissociates the destination identified with said entry from the route identified with said entry. It is an implementation-specific matter as to whether the agent removes an invalidated entry from the table. Accordingly, management stations must be prepared to receive tabular information from agents that corresponds to entries not currently in use. Proper interpretation of such entries requires examination of the relevant ipRouteType object.

ipRouteProto - The routing mechanism through which this route was learned. Inclusion of values for gateway routing protocols is not intended to imply that hosts should support those protocols. (enum)

ipRouteAge - The number of seconds since this route was last updated or otherwise determined to be correct. Note that no semantics of ''too old'' can be implied except through knowledge of the routing protocol by which the route was learned. (int) Returns a fixed value of 0.

ipRouteMask - Indicate the mask to be logical-ANDed with the destination address before being compared to the value in the ipRouteDest field. For those systems that do not support arbitrary subnet masks, an agent constructs the value of the ipRouteMask by determining whether the value of the correspondent ipRouteDest field belongs to a class-A, B, or C network, and then using one of:

mask	network
255.0.0.0	class-A
255.255.0.0	class-B
255.255.255.0	class-C

If the value of the ipRouteDest is 0.0.0.0 (a default route), then the mask value is also 0.0.0.0. It should be noted that all IP routing subsystems implicitly use this

mechanism. (netaddress)

ipRouteMetric5 - An alternate routing metric for this route. The semantics of this metric are determined by the routing-protocol specified in the route's ipRouteProto value. If this metric is not used, its value should be set to -1. (int) Returns a fixed value of -1.

ipRouteInfo - A reference to MIB definitions specific to the particular routing protocol responsible for this route, as determined by the value specified in the route's ipRouteProto value. If this information is not present, its value should be set to the OBJECT IDENTIFIER { 0 0 }, which is a syntactically valid object identifier. Any conformant implementation of ASN.1 and BER must be able to generate and recognize this value. (objectid)

ipNetToMediaTable is the IP Address Translation table used for mapping from IP addresses to physical addresses.

ipNetToMediaIfIndex - The interface on which this entry's equivalence is effective. The interface identified by a particular value of this index is the same interface as identified by the same value of ifIndex. (int)

ipNetToMediaPhysAddress - The media-dependent physical address. (octet[128])

ipNetToMediaNetAddress - The IpAddress corresponding to the media- dependent physical address. (netaddress)

ipNetToMediaType - The type of mapping. (enum) Returns a fixed value of (3)dynamic.

Setting this object to the value invalid(2) has the effect of invalidating the corresponding entry in the ipNetToMediaTable. That is, it effectively dissociates the interface identified with said entry from the mapping identified with said entry. It is an implementation-specific matter as to whether the agent removes an invalidated entry from the table. Accordingly, management stations must be prepared to receive tabular information from agents that corresponds to entries not currently in use. Proper interpretation of such entries requires examination of the relevant ipNetToMediaType object.

The **icmp** group reports statistics about the ICMP group.

icmpInMsgs - The total number of ICMP messages that the entity received. Note that this counter includes all those counted by icmpInErrors. (counter)

icmpInErrors - The number of ICMP messages that the entity received but determined as having ICMP-specific errors (bad ICMP checksums, bad length, and the like.). (counter)

icmpInDestUnreachs - The number of ICMP Destination Unreachable messages received. (counter)

icmpInTimeExcds - The number of ICMP Time Exceeded messages received. (counter)

icmpInParmProbs - The number of ICMP Parameter Problem messages received. (counter)

icmpInSrcQuenchs - The number of ICMP Source Quench messages received. (counter)

icmpInRedirects - The number of ICMP Redirect messages received. (counter)

icmpInEchos - The number of ICMP Echo (request) messages received. (counter)

icmpInEchoReps - The number of ICMP Echo Reply messages received. (counter)

icmpInTimestamps - The number of ICMP Timestamp (request) messages received. (counter)

icmpInTimestampReps - The number of ICMP Timestamp Reply messages received. (counter)

icmpInAddrMasks - The number of ICMP Address Mask Request messages received. (counter)

icmpInAddrMaskReps - The number of ICMP Address Mask Reply messages received. (counter)

icmpOutMsgs - The total number of ICMP messages that this entity attempted to send. Note that this counter includes all those counted by icmpOutErrors. (counter)

icmpOutErrors - The number of ICMP messages that this entity did not send due to problems discovered within ICMP, such as a lack of buffers. This value should not include errors discovered outside the ICMP layer, such as the inability of IP to route the resultant datagram. In some implementations there may be no types of errors that contribute to this counter's value. (counter)

icmpOutDestUnreachs - The number of ICMP Destination Unreachable messages sent. (counter)

icmpOutTimeExcds - The number of ICMP Time Exceeded messages sent. (counter)

icmpOutParmProbs - The number of ICMP Parameter Problem messages sent. (counter)

icmpOutSrcQuenchs - The number of ICMP Source Quench messages sent. (counter)

icmpOutRedirects - The number of ICMP Redirect messages sent. For a host, this object will always be zero, since hosts do not send redirects. (counter)

icmpOutEchos - The number of ICMP Echo (request) messages sent. (counter)

icmpOutEchoReps - The number of ICMP Echo Reply messages sent. (counter)

icmpOutTimestamps - The number of ICMP Timestamp (request) messages sent. (counter)

icmpOutTimestampReps - The number of ICMP Timestamp Reply messages sent. (counter)

icmpOutAddrMasks - The number of ICMP Address Mask Request messages sent. (counter)

icmpOutAddrMaskReps - The number of ICMP Address Mask Reply messages sent. (counter)

The **tcp** group reports statistics about the TCP group.

tcpRtoAlgorithm - The algorithm used to determine the timeout value used for retransmitting unacknowledged octets. (enum)

tcpRtoMin - The minimum value permitted by a TCP implementation for the retransmission timeout, measured in milliseconds. More refined semantics for objects of this type depend upon the algorithm used to determine the retransmission timeout. In particular, when the timeout algorithm is rsre(3), an object of this type has the semantics of the LBOUND quantity described in RFC 793. (int)

tcpRtoMax - The maximum value permitted by a TCP implementation for the retransmission timeout, measured in milliseconds. More refined semantics for objects of this type depend upon the algorithm used to determine the retransmission timeout. In particular, when the timeout algorithm is rsre(3), an object of this type has the semantics of the UBOUND quantity described in RFC 793. (int)

tcpMaxConn - The limit on the total number of TCP connections that the entity can support. In entities where the maximum number of connections is dynamic, this object should contain the value -1. (int)

tcpActiveOpens - The number of times that TCP connections have made a direct transition to the SYN-SENT state from the CLOSED state. (counter)

tcpPassiveOpens - The number of times that TCP connections have made a direct transition to the SYN-RCVD state from the LISTEN state. (counter)

tcpAttemptFails - The number of times that TCP connections have made a direct transition to the CLOSED state from either the SYN-SENT state or the SYN-RCVD state, plus the number of times TCP connections have made a direct transition to the LISTEN state from the SYN-RCVD state. (counter)

tcpEstabResets - The number of times TCP connections have made a direct transition to the CLOSED state from either the ESTABLISHED state or the CLOSE-WAIT state. (counter)

tcpCurrEstab - The number of TCP connections for which the current state is either ESTABLISHED or CLOSE- WAIT. (gauge)

tcpInSegs - The total number of segments received, including those received in error. This count includes segments received on currently established connections. (counter)

tcpOutSegs - The total number of segments sent, including those on current connections but excluding those containing only retransmitted octets. (counter)

tcpRetransSegs - The total number of segments retransmitted - that is, the number of TCP segments transmitted containing one or more previously transmitted octets. (counter)

tcpInErrs - The total number of segments received in error (for example, bad TCP checksums). (counter)

tcpOutRsts - The number of TCP segments sent containing the RST flag. (counter)

tcpConnTable is a table containing TCP connection-specific information.

tcpConnState - The state of this TCP connection. (enum)

The only value that may be set by a management station is deleteTCB(12). Accordingly, it is appropriate for an agent to return a ''badValue'' response if a management station attempts to set this object to any other value.

If a management station sets this object to the value deleteTCB(12), then this has the effect of deleting the TCB (as defined in RFC 793) of the corresponding connection on the managed node. This results in immediate termination of the connection.

As an implementation-specific option, an RST segment may be sent from the managed node to the other TCP endpoint. (Note, however, that RST segments are not sent reliably.)

tcpConnLocalAddress - The local IP address for this TCP connection. For a connection in the listen state that is willing to accept connections for any IP interface associated with the node, the value 0.0.0.0 is used. (netaddress)

tcpConnLocalPort - The local port number for this TCP connection. (int)

tcpConnRemAddress - The remote IP address for this TCP connection. (netaddress)

tcpConnRemPort - The remote port number for this TCP connection. (int)

The **udp** group reports statistics about the UDP group.

udpInDatagrams - The total number of UDP datagrams delivered to UDP users. (counter) Returns a fixed value of 0.

udpNoPorts - The total number of received UDP datagrams for which there was no application at the destination port. (counter) Returns a fixed value of 0.

udpInErrors - The number of received UDP datagrams that could not be delivered for reasons other than the lack of an application at the destination port. (counter)

udpOutDatagrams - The total number of UDP datagrams sent from this entity. (counter) Returns a fixed value of 0.

The **udpTable** is a table containing UDP listener information.

udpLocalAddress - The local IP address for this UDP listener. For a UDP listener that is willing to accept datagrams for any IP interface associated with the node,

the value 0.0.0.0 is used. (netaddress)

udpLocalPort - The local port number for this UDP listener. (int)

The **snmp** group reports statistics about the SNMP group.

snmpInPkts - The total number of Messages delivered to the SNMP entity from the transport service. (counter)

snmpOutPkts - The total number of SNMP Messages passed from the SNMP protocol entity to the transport service. (counter)

snmpInBadVersions - The total number of SNMP Messages delivered to the SNMP protocol entity that were for an unsupported SNMP version. (counter)

snmpInBadCommunityNames - The total number of SNMP Messages delivered to the SNMP protocol entity that used a SNMP community name not known to said entity. (counter)

snmpInBadCommunityUses - The total number of SNMP Messages delivered to the SNMP protocol entity, which represented an SNMP operation not allowed by the SNMP community named in the Message. (counter)

snmpInASNParseErrs - The total number of ASN.1 or BER errors encountered by the SNMP protocol entity when decoding received SNMP Messages. (counter)

snmpInTooBigs - The total number of SNMP PDUs delivered to the SNMP protocol entity for which the value of the error-status field is ''tooBig.'' (counter)

snmpInNoSuchNames - The total number of SNMP PDUs delivered to the SNMP protocol entity for which the value of the error-status field is ''noSuchName.'' (counter)

snmpInBadValues - The total number of SNMP PDUs delivered to the SNMP protocol entity for which the value of the error-status field is ''badValue.'' (counter)

snmpInReadOnlys - The total number valid SNMP PDUs delivered to the SNMP protocol entity for which the value of the error-status field is ''readOnly.'' It should be noted that it is a protocol error to generate an SNMP PDU that contains the value ''readOnly'' in the error-status field. This object is provided as a means of detecting incorrect implementations of the SNMP. (counter)

snmpInGenErrs - The total number of SNMP PDUs delivered to the SNMP protocol entity for which the value of the error-status field is ''genErr.'' (counter)

snmpInTotalReqVars - The total number of MIB objects successfully retrieved by the SNMP protocol entity as the result of receiving valid SNMP Get-Request and Get-Next PDUs. (counter)

snmpInTotalSetVars - The total number of MIB objects successfully altered by the SNMP protocol entity as the result of receiving valid SNMP Set-Request PDUs. (counter)

snmpInGetRequests - The total number of SNMP Get-Request PDUs accepted

and processed by the SNMP protocol entity. (counter)

snmpInGetNexts - The total number of SNMP Get-Next PDUs accepted and processed by the SNMP protocol entity. (counter)

snmpInSetRequests - The total number of SNMP Set-Request PDUs accepted and processed by the SNMP protocol entity. (counter)

snmpInGetResponses - The total number of SNMP Get-Response PDUs accepted and processed by the SNMP protocol entity. (counter)

snmpInTraps - The total number of SNMP Trap PDUs accepted and processed by the SNMP protocol entity. (counter)

snmpOutTooBigs - The total number of SNMP PDUs generated by the SNMP protocol entity for which the value of the error-status field is ''tooBig.'' (counter)

snmpOutNoSuchNames - The total number of SNMP PDUs generated by the SNMP protocol entity for which the value of the error-status is ''noSuchName.'' (counter)

snmpOutBadValues - The total number of SNMP PDUs generated by the SNMP protocol entity for which the value of the error-status field is ''badValue.'' (counter)

snmpOutGenErrs - The total number of SNMP PDUs generated by the SNMP protocol entity for which the value of the error-status field is ''genErr.'' (counter)

snmpOutGetRequests - The total number of SNMP Get-Request PDUs which have been generated by the SNMP protocol entity. (counter)

snmpOutGetNexts - The total number of SNMP Get-Next PDUs generated by the SNMP protocol entity. (counter)

snmpOutSetRequests - The total number of SNMP Set-Request PDUs generated by the SNMP protocol entity. (counter)

snmpOutGetResponses - The total number of SNMP Get-Response PDUs generated by the SNMP protocol entity. (counter)

snmpOutTraps - The total number of SNMP Trap PDUs generated by the SNMP protocol entity. (counter)

snmpEnableAuthenTraps - Indicates whether the SNMP agent process is permitted to generate authentication-failure traps. The value of this object overrides any configuration information. As such, it provides a means whereby all authentication-failure traps may be disabled. (enum)

Note that this object must be stored in non-volatile memory, so that it remains constant between reinitializations of the network management system.

The following are Sun-specific group and table definitions.

The **sunSystem** group reports general system information.

agentDescr - The SNMP agent's description of itself. (string[255])

hostID - The unique Sun hardware identifier. The value returned is four byte binary string. (octet[4])

motd - The first line of **/etc/motd .** (string[255])

unixTime - The UNIX system time. Measured in seconds since January 1, 1970 GMT. (counter)

The **sunProcessTable** table reports UNIX process table information.

psProcessID - The process identifier for this process. (int)

psParentProcessID - The process identifier of this process's parent. (int)

psProcessSize - The combined size of the data and stack segments (in kilobytes.) (int)

psProcessCpuTime - The CPU time (including both user and system time) consumed so far. (int)

psProcessState - The run-state of the process. (octet[4])

R	Runnable
T	Stopped
P	In page wait
D	Non-interruptable wait
S	Sleeping (less than 20 seconds)
I	Idle (more than 20 seconds)
Z	Zombie

psProcessWaitChannel - Reason process is waiting. (octet[16])

psProcessTTY - Terminal, if any, controlling this process. (octet[16])

psProcessUserName - Name of the user associated with this process. (octet[16])

psProcessUserID - Numeric form of the name of the user associated with this process. (int)

psProcessName - Command name used to invoke this process. (octet[64])

psProcessStatus - Setting this variable will cause a signal of the set value to be sent to the process. (int)

The **sunHostPerf** group reports hostperf information.

rsUserProcessTime - Total number of timeticks used by user processes since the last system boot. (counter)

rsNiceModeTime - Total number of timeticks used by ''nice'' mode since the last system boot. (counter)

rsSystemProcessTime - Total number of timeticks used by system processes since the last system boot. (counter)

rsIdleModeTime - Total number of timeticks in idle mode since the last system

boot. (counter)

rsDiskXfer1 - Total number of disk transfers since the last boot for the first of four configured disks. (counter)

rsDiskXfer2 - Total number of disk transfers since the last boot for the second of four configured disks. (counter)

rsDiskXfer3 - Total number of disk transfers since the last boot for the third of four configured disks. (counter)

rsDiskXfer4 - Total number of disk transfers since the last boot for the fourth of four configured disks. (counter)

rsVPagesIn - Number of pages read in from disk. (counter)

rsVPagesOut - Number of pages written to disk. (counter)

rsVSwapIn - Number of pages swapped in. (counter)

rsVSwapOut - Number of pages swapped out. (counter)

rsVIntr - Number of device interrupts. (counter)

rsIfInPackets - Number of input packets. (counter)

rsIfOutPackets - Number of output packets. (counter)

rsIfInErrors - Number of input errors. (counter)

rsIfOutErrors - Number of output errors. (counter)

rsIfCollisions - Number of output collisions. (counter)

ERRORS

cannot dispatch request
The proxy cannot dispatch the request. The rest of the message indicates the cause of the failure.

select(2) failed
A **select**(2) system call failed. The rest of the message indicates the cause of the failure.

sendto(2) failed
A **sendto**(2) system call failed. The rest of the message indicates the cause of the failure.

recvfrom(2) failed
A **recvfrom**(2) system call failed. The rest of the message indicates the cause of the failure.

no response from system
The SNMP agent on the target system does not respond to SNMP requests. This error might indicate that the SNMP agent is not running on the target system, the target system is down, or the network containing the target system is unreachable.

response too big

The agent could not fit the results of an operation into a single SNMP message. Split large groups or tables into smaller entities.

missing attribute
An attribute is missing from the requested group.

bad attribute type
An object attribute type received from the SNMP agent that does not match the attribute type specified by the proxy agent schema. The rest of the message indicates the expected type and received type.

cannot get sysUpTime
The proxy agent cannot get the variable *sysUpTime* from the SNMP agent.

sysUpTime type bad
The variable *sysUpTime* received from the SNMP agent has the wrong data type.

unknown SNMP error
An unknown SNMP error was received.

bad variable value
The requested specified an incorrect syntax or value for a set operation.

variable is read only
The SNMP agent did not perform the set request because a variable to set may not be written.

general error
A general error was received.

cannot make request PDU
An error occurred building a request PDU.

cannot make request varbind list
An error occurred building a request variable binding list.

cannot parse response PDU
An error occurred parsing a response PDU.

request ID - response ID mismatch
The response ID does not match the request ID.

string contains non-displayable characters
A displayable string contains non-displayable characters.

cannot open schema file
An error occurred opening the proxy agent schema file.

cannot parse schema file
The proxy agent couldn't parse the proxy agent schema file.

cannot open host file
An error occurred opening the file associated with the *na.snmp.hostfile* keyword in **/etc/opt/SUNWconn/snm/snm.conf** for Solaris 2.x and **/etc/snm.conf** for Solaris 1.x.

cannot parse host file

The proxy agent was unable to parse the file associated with the *na.snmp.hostfile* keyword in **/etc/opt/SUNWconn/snm/snm.conf** for Solaris 2.x and **/etc/snm.conf** for Solaris 1.x.

attribute unavailable for set operations
The set could not be completed because the attribute was not available for set operations.

BUGS

snmpd returns the wrong interface speed for the SBUS FDDI interface (for example, ''bf0'').

snmpd does not return a MAC address for the SBUS FDDI interface (for example, ''bf0'').

Process names retrieved from **snmpd** contain a leading blank space.

When you change attribute values in the system group with an SNMP set request, the change is effective only as long as **snmpd** is running. **snmpd** does not save the changes to **/etc/opt/SUNWconn/snm/snmpd.conf** for Solaris 2.x or **/etc/snmpd.conf** for Solaris 1.x.

NAME

snmpv2d - Sun SNMPv2 Agent

SYNOPSIS

snmpv2d [–**r**] [–**p** *port*] [–**a**] [–**c** *config-file*] [–**T** *trace-level*]

DESCRIPTION

snmpv2d is a mostly RFC 1448-compliant SNMPv2 agent. **snmpv2d** supports MIB-II as defined in RFC 1213, with Sun extensions under Sun's enterprise number. The MIB (Management Information Base) is both readable and writable. **snmpv2d** supports all SNMP protocol operations including GET-REQUEST, GETBULK-REQUEST, GETNEXT-REQUEST, INFORM-REQUEST, SET-REQUEST, GET-REPLY, and TRAP.

snmpv2d supports the coldStart, linkUp, linkDown, and authentication traps.

snmpv2d uses UDP port 161, the standard SNMP port. **snmpv2d** issues traps through the same port on which it receives SNMP requests.

snmpv2d must run with ''root'' privileges and is typically started at system startup via **/etc/rc.local. snmpv2d** may not be started using **inetd .** When started, **snmpv2d** detaches itself from the keyboard, disables all signals except SIGKILL, SIGILL, SIGUSR1, and SIGUSR2, and places itself in the background.

OPTIONS

There are no options.

FILES

The following files are provided or created during installation for agent operation:

snmpv2d	agent binary
v2install	v2 installation script
snmpv2d.conf	configuration information (described below)
acl.pty	access control configuration
agt.pty	initial party table information for agents
context.pty	context information
mgr.cnf	configuration information for managers
mgr.pty	initial party table information for managers
view.pty	MIB view information

The first two files, as delivered, are located in the **/opt/SUNWconn/snm/agents** directory for Solaris 2.x and **/usr/snm/agents** for Solaris 1.x. The rest are created during installation, and usually reside in the **/etc/opt/snm/{agent | manager}** directory for Solaris 2.x and **/etc/snm/{agent | manager** for Solaris 1.x. Upon installation, **snmpv2d.conf** is usually placed in **/etc/opt/snm/agent** for Solaris 2.x and **/etc/snm/agent** for Solaris 1.x.

For more information on the configuration files, see their corresponding man page.

INSTALLATION

snmpv2d and its configuration file (**snmpv2d.conf**(5)) may be placed in any directory. However, **/opt/SUNWconn/snm/agents** for Solaris 2.x or **/usr/snm/agents** for Solaris 1.x is suggested for **snmpv2d** itself and **/etc/opt/snm/agent** (Solaris 2.x) or **/etc/snm/agent** (Solaris 1.x) for the configuration file. You should modify the configuration file as appropriate. If you make any changes to **snmpv2d.conf**(5) file keyword values, you must kill and restart **snmpv2d** for the changes to take effect.

To kill **snmpv2d,** as root, type **/etc/init.d/init.snmpd stop** for Solaris 2.x and **/etc/init.snmpd stop** for Solaris 1.x. To start snmpv2d, as root, type **/etc/init.d/init.snmpd start** for Solaris 2.x and **/etc/init.snmpd start** for Solaris 1.x.

Your **/etc/services** file (or NIS equivalent) should contain the following entries:

```
snmp            161/udp         # Simple Network Mgmt Protocol
snmp-trap       162/udp         snmptrap# SNMP trap (event) messages
```

snmpv2d is typically started at boot time through **/etc/rc.local.**

(Note that you need not explicitly place **snmpv2d** into the background. Also note that **snmpv2d** may not be started by **inetd .)**

SECURITY

SNMPv2 uses DES and MD5 for authentication and encryption. However, due to export licensing, DES and MD5 are not included in this release. See the configuration files man pages for more information about security.

ERRORS

cannot dispatch request
: The proxy cannot dispatch the request. The rest of the message indicates the cause of the failure.

select(2) failed
: A **select(2)** system call failed. The rest of the message indicates the cause of the failure.

sendto(2) failed
: A **sendto(2)** system call failed. The rest of the message indicates the cause of the failure.

recvfrom(2) failed
: A **recvfrom(2)** system call failed. The rest of the message indicates the cause of the failure.

no response from system
: The SNMP agent on the target system does not respond to SNMP requests. This error might indicate that the SNMP agent is not running on the target system, the target system is down, or the network containing the target system is unreachable.

response too big
: The agent could not fit the results of an operation into a single SNMP message. Split large groups or tables into smaller entities.

missing attribute
: An attribute is missing from the requested group.

bad attribute type
: An object attribute type received from the SNMP agent that does not match the attribute type specified by the proxy agent schema. The rest of the message indicates the expected type and received type.

cannot get sysUpTime
: The proxy agent cannot get the variable *sysUpTime* from the SNMP agent.

sysUpTime type bad
The variable *sysUpTime* received from the SNMP agent has the wrong data type.

unknown SNMP error
An unknown SNMP error was received.

bad variable value
The requested specified an incorrect syntax or value for a set operation.

variable is read only
The SNMP agent did not perform the set request because a variable to set may not be written.

general error
A general error was received.

cannot make request PDU
An error occurred building a request PDU.

cannot make request varbind list
An error occurred building a request variable binding list.

cannot parse response PDU
An error occurred parsing a response PDU.

request ID - response ID mismatch
The response ID does not match the request ID.

string contains non-displayable characters
A displayable string contains non-displayable characters.

cannot open schema file
An error occurred opening the proxy agent schema file.

cannot parse schema file
The proxy agent couldn't parse the proxy agent schema file.

cannot open host file
An error occurred opening the file associated with the *na.snmp.hostfile* keyword in **/etc/opt/snm/snm.conf** for Solaris 2.x and **/etc/snm.conf** for Solaris 1.x.

cannot parse host file
The proxy agent was unable to parse the file associated with the *na.snmp.hostfile* keyword in **/etc/opt/snm/snm.conf** for Solaris 2.x and **/etc/snm.conf** for Solaris 1.x.

attribute unavailable for set operations
The set could not be completed because the attribute was not available for set operations.

BUGS

snmpv2d returns the wrong interface speed for the SBUS FDDI interface (for example, ''bf0'').

snmpv2d does not return a MAC address for the SBUS FDDI interface (for example, ''bf0'').

Process names retrieved from **snmpv2d** contain a leading blank space.

When you change attribute values in the system group with an SNMP set request, the change is effective only as long as **snmpv2d** is running. **snmpv2d** does not save the changes to **/etc/opt/snm/agent/snmpv2d.conf** for Solaris 2.x and **/etc/snm/agent/snmpv2d.conf** for Solaris 1.x.

SEE ALSO

v2install(1), **agt.pty**(5), **context.pty**(5), **mgr.cnf**(5), **mgr.pty**(5), **snmpv2d.conf**(5), **view.pty**(5), SNMP RFCs (RFC1155 RFC1157 RFC1212 RFC1213 RFC1214, RFCs 1441-1452)

NAME

snmpv2d.conf – the SNMP agent configuration

SYNOPSIS

TAG VALUE

DESCRIPTION

The configuration file **snmpv2d.conf** is one of several configuration files required by the SNMPv2 entities and the only file required by the SNMPv1 entities. The default location of **snmpv2d.conf** is **/etc/opt/snm/agent** for Solaris 2.x and **/etc/snm/agent** for Solaris 1.x, but can be specified by the environment variable **SR_AGT_CONF_DIR.**

The **snmpv2d.conf** file defines initial values for the system group, community names, trap community names, and whether authentication-failure traps should be generated.

Note: for bilingual and SNMPv2 agents, the **snmpv2d.conf** initializes the system variables and defines authentication-failure traps. Communities are defined only when compiled as SNMPv1.

The form of each line is:

TAG VALUE

Where *TAG* is one of

sysDescr
sysLocation
sysContact
snmpEnableAuthenTraps
community
trap

and *VALUE* is a valid value for the given key.

White space (tabs, spaces, line-feeds / carriage-returns) and blank lines are ignored.

System Variable Initialization

When the *TAG* is one of the system variables:

sysDescr
sysLocation
sysContact
snmpEnableAuthenTraps

that variable will be initialized to whatever value follows it.

According to RFC1213, the variables sysDescr, sysLocation, and sysContact can be any string of 0 to 255 NVT ASCII characters, and snmpEnableAuthenTraps can be either

1 to enable traps

or

2 to disable traps.

For example:

```
sysDescr        SNMP agent from My Company
sysLocation      Anywhere, USA
sysContact      Woody Boyd (156) 555-7667
```

snmpEnableAuthenTraps 1

initializes sysDescr, sysLocation, and sysContact to appropriate strings and enables authentication-failure traps.

Community Entries

When the *TAG* is "community," the format of the *VALUE* clause is:

community-name IP-address privileges community-id

where

- community-name may be any string.
- IP-address indicates the remote's site for which this community is valid. If the IP address is 0.0.0.0,any address may communicate using that session name.
- privileges are any one of:

  READ for read only,
  WRITE for read/write, or
  NONE to lock out a community name.

- The community identifier may be used internally.

For example,

community test1 130.117.1.20 READ 0

defines a community named test1 that allows reads *only* from IP address 130.117.1.20.

Trap Community Entries

When the *TAG* is "trap," the format of the *VALUE* clause is:

trap-community-name IP-address

where

- The trap-community name may be any string.
- The IP address indicates the destination address to be included in the trap PDU.

For example,

trap test2 128.169.4.15

indicates to the agent that if a trap needs to be sent, the trap PDU should be built using the community name test2 and sent to the trap port at 128.169.4.15.

FILES

SNMPv2 Configuration Files

When the agent is compiled with either SNMPv2 or both SNMPv1 and SNMPv2 defined, the additional configuration files **acl.pty, agt.pty, context.pty,** and **view.pty** are required.

Additional SNMPv2 Configuration Files

When the agent is compiled with either SNMPv2 or both SNMPv1 and SNMPv2 defined, the configuration files **acl.pty, agt.pty, context.pty,** and **view.pty** are required.

acl.pty Access control privileges for the SNMPv2 parties.

agt.pty Party information for the manager routines.

context.pty
Context information for the SNMPv2 parties.

view.pty
MIB view information for the SNMPv2 parties.acl.pty

For Solaris 2.x, the files are located under:

/etc/opt/snm/agent/acl.pty
/etc/opt/snm/agent/agent.pty
/etc/opt/snm/agent/context.pty
/etc/opt/snm/agent/snmpv2d.conf
/etc/opt/snm/agent/view.pty

For Solaris 1.x, the files are located under:

/etc/snm/agent/acl.pty
/etc/snm/agent/agent.pty
/etc/snm/agent/context.pty
/etc/snm/agent/snmpv2d.conf
/etc/snm/agent/view.pty

SEE ALSO

snmpv2d(1), **v2install**(1), **acl.pty**(5), **agt.pty**(5), **context.pty**(5), **mgr.cnf**(5), **mgr.pty**(5), **view.pty**(5), SNMP RFCs (RFC1155 RFC1157 RFC1212 RFC1213 RFC1215, RFCs 1441-1452)

LIMITATIONS

Not all MIB variables can be supported without source code alterations to the kernel or device drivers.

NAME

v2install – generate SNMPv2 configuration files for agents and managers

SYNOPSIS

v2install <*host name*>

DESCRIPTION

The **v2install** utility generates the SNMPv2 configuration files for a given list of agents and managers, and copies them to the **/etc/opt/snm/{agent | manager}** directory for Solaris 2.x and **/etc/snm/{agent | manager}** for Solaris 1.x.

There are no options to the **v2install** command.

INPUT

Three files are required by **v2install;** all must be present but may be empty, depending on which manager files are being generated. The file *agents* contains a list of agents, the file *mgrs.v1,* contains a list of SNMPv1 managers, and *mgrs.v2* contains a list of SNMPv2 managers.

Note: The *mgrs.v2* and *agents* files should be present because information is shared between SNMPv2 agents and managers that must be reflected in their configuration files. The *mgrs.v1* file is necessary only if SNMPv1 managers will be used with bilingual agents.

agents lists address information for each agent.

The format is as follows:

IP-address Nodename

where

IP-address is the dotted decimal version of the IP address of the agent (i. e., 128.169.4.16),

Nodename is the alias string for the *IP-address* (i. e., myserver). Each entry must begin on a separate line.

mgrs.v1

lists any community strings––with read or read / write privileges––that should be included on each agent. Nothing extra is generated for the manager configuration files.

Note: The community strings found in **snmpv2d.conf**(5) are ignored in bilingual agents. The entries in *mgrs.v1* are used to generate the appropriate parties for SNMPv1 communities in a bilingual system.

The format is as follows:

IP-addr Nodename read-community write-community

where

IP-addr is the dotted decimal version of the IP address of the manager (i. e., 128.169.4.16),

Nodename is the alias string for the *IP-addr* (i. e., myserver),

read-community is the community name, found in the **snmpv2d.conf**(5) file, that

defines the read access privileges for this manager, and

write-community is the community name, found in the **snmpv2d.conf**(5) file, that defines the write access privileges for this manager. Each entry must begin on a separate line.

mgrs.v2

lists address information for each manager.

The format is as follows:

IP-address Nodename

where

IP-address is the dotted decimal version of the IP address of the manager (i. e., 128.169.4.16),

Nodename is the alias string for the *IP-address* (i. e., myserver). Each entry must begin on a separate line.

OUTPUT

The **v2install** tool will place each group of configuration files in a hostname directory based on the *Nodename* in either **configs/nodename/agent** or **configs/nodename/manager,** depending on which files were supplied. There will be two parties generated for each agent-manager pair.

Note: All managers will share a common noauth/nopriv pair of parties. The first two parties in the agent's table will always be *noauth/nopriv* parties with party names of

initialPartyId. *IP-address.* 1

and

initialPartyId. *IP-address.* 2

Each manager address will have the following files generated for it:

mgr.pty – party configuration file
acl.pty – access control configuration file
context.pty – MIB context configuration file
mgr.cnf – manager configuration file
view.pty –MIB view configuration file

Each agent address will have the following files generated for it:

agt.pty – party configuration file
acl.pty – access control configuration file
context.pty – MIB context configuration file
snmpv2d.conf – snmpv2d configuration file
view.pty – MIB view configuration file

EXAMPLES

Example agents file:

128.169.4.16 myserver
128.169.4.47 screamer

Example mgrs.v2 file:

```
128.169.4.16 myserver
128.169.4.20 bossie
128.169.4.15 murvle15
128.169.4.25 wildman
```

Example mgrs.v1 file:

```
128.169.4.16 myserver public TopSecret
128.169.4.47 screamer ten30 private
128.169.4.15 murvle15 sparc woof
128.169.4.25 wildman common secret
```

FILES

Input:

agents – list of agents
mgrs.v1 – list of managers that are SNMPv1
mgrs.v2 – list of managers that are SNMPv2

Output:

managers:

mgr.pty – party configuration file
acl.pty – access control configuration file
context.pty – MIB context configuration file
mgr.cnf – manager configuration file
view.pty – MIB view configuration file

agents:

agt.pty – party configuration file
acl.pty – access control configuration file
context.pty – MIB context configuration file
snmpv2d.conf – snmpv2d configuration file
view.pty – MIB view configuration file

SEE ALSO

acl.pty(5), **agt.pty**(5), **context.pty**(5), **mgr.cnf**(5), **mgr.pty**(5), **snmpv2d.conf**(5), **view.pty**(5), SNMP RFCs (RFC1155 RFC1157 RFC1212 RFC1213 RFC1215, RFCs 1441-1452)

KNOWN BUGS

Blank lines are *not* tolerated.

Duplicate *Nodenames* are not detected.

Randomly generated keys are not always unique. A better seeding function is needed.

NAME

v2mib2schema – Translates SNMPv2 MIB files to SunNet Manager schema files.

SYNOPSIS

v2mib2schema *mib_filename [schema_filename]*

DESCRIPTION

v2mib2schema translates MIB files for SNMP agents to the format required by SunNet Manager's agent schema files.

MIB files must conform strictly to the RFC 1155 SMI ("Structure and Identification of Management Information for TCP / IP-based Internets"). MIB files must also conform to the subset of ASN.1 syntax specified in RFC 1155. MIB files may also include "Concise MIB Definitions" as specified in RFC 1212 and the TRAP-TYPE macro, as defined in "A Convention for Defining Traps for use with the SNMP" (RFC 1215).

v2mib2schema also generates two other files: a ∗*.oid* file and a ∗*.traps* file. The ∗*.oid* file is an object identifier file that contains a table of object identifiers and names. This file is used by the **build_oid**(1) program. The ∗*.traps* file contains traps definitions; this file may or may not be generated, depending on whether traps were specified in the MIB.

It is important to note that **v2mib2schema** is not a compiler for MIB specifications. In fact, **v2mib2schema** expects the MIB specification to conform strictly to the SMI defined in RFC 1155 to work correctly. However, **v2mib2schema** does perform some primitive error detections. If any syntax error is reported, **v2mib2schema** aborts immediately. If warnings are detected, **v2mib2schema** generates warning messages but generates the schema file anyway.

Please note the following:

1. SNM schemas do not support nested groups and tables; therefore, **v2mib2schema** does not translate them. If your MIB uses nested groups or tables, you need to rewrite the groups and tables as un-nested groups or tables.
2. SNM schemas do not support ranges defined for data types. Therefore, **v2mib2schema** ignores ranges defined for data types.
3. **v2mib2schema** recognizes only the following set of data types as defined in RFC 1155: INTEGER, OCTET STRING, SEQUENCE and SEQUENCE OF.
4. **v2mib2schema** recognizes only the following set of defined types: NetworkAddress, IpAddress, Counter, Counter32, Gauge, Gauge32, Timeticks and Opaque, as defined in RFC 1155; and PhysAddress and DisplayString, as defined in RFC 1213 (MIB-II). All defined types must eventually resolve to one of the data types listed in 3.
5. **v2mib2schema** does not handle IMPORTed types very well. If a type is imported from any file other than the SMI, MIB I or MIB II, it will not be recognized. You will have to define the type in the MIB.
6. As defined in RFC 1155, you may not specify 0 as an object identifier or as an enumerated value. However, **v2mib2schema** would still generate the schema file, but warnings would be generated.
7. The key value in the characteristics field for tables cannot be determined in all cases from the INDEX statement in the MIB. In instances when the MIB contains no INDEX

clauses or when **v2mib2schema** cannot determine the key value, a warning is generated.

8. Although **v2mib2schema** parses DEFVAL clauses, it currently ignores them.
9. **v2mib2schema** generates only the objects found in the enterprise-specific MIB file into the schema file. If you want to include objects in the standard MIB-I schema file into the enterprise specific schema file, you would have to merge them yourself using your favorite editor.

OPTIONS

If only one file name is specified, **v2mib2schema** translates the MIB specifications in *mib_filename* to SunNet Manager schema format in *mib_filename.schema.* **v2mib2schema** also generates *mib_filename.oid* and, optionally, *mib_filename.traps.* If two file names are specified, **v2mib2schema** translates the MIB specifications in *mib_filename* to SunNet Manager schema format in *schema_filename.*

NOTES

This section describes some common errors that occur when writing MIBs. Reading this section may help to resolve any problems encountered when running **v2mib2schema.**

Sample MIB Module

Every MIB must have a module name, a BEGIN, an optional IMPORTS statement, the actual MIB body of definitions, and an END. The following is a skeletal structure of a MIB module:

```
FOO-MIB DEFINITIONS ::= BEGIN
-- FOO is the name you would give to the MIB module
IMPORTS
        -- Objects which you import, usually from
        -- RFC1155-SMI, for example:
        experimental, OBJECT-TYPE FROM RFC1155-SMI;

foo OBJECT IDENTIFIER ::= { enterprise 888 }
-- The above statement defines foo as an
-- object identifier with an enterprise number
-- of 888.
-- BODY OF DEFINITIONS
END
```

Note that ASN.1 syntax is case-sensitive, so it is very important to get the case right. For example, object variables and identifiers *must* start with a lowercase letter, while object types *must* start with an uppercase letter. Names cannot include underscore (_) characters. Failure to follow these conventions will result in **v2mib2schema** generating a syntax error and aborting. Comments are preceded by a pair of hyphens.

Common errors when writing MIBs

1. MIBs must be written in the SMI syntax, which is a subset of ASN.1 syntax. **v2mib2schema** does not accept MIBs written in ASN.1 syntax outside of the SMI subset.
2. ASN.1 syntax is case-sensitive. Be very careful when typing in names and types.

Remember that types must begin with an uppercase letter while names must begin with a lowercase letter. Names cannot include underscores (_).

3. Typographical errors make up the largest number of errors detected by **v2mib2schema.** Examples of common typos include (those on the left are typos):

   "enterprise" instead of "enterprises"
   "Timeticks" or "timeticks" instead of "TimeTicks"
   "IPADDRESS" instead of "IpAddress"
   "non-accessible" instead of "not-accessible"

4. Tables *must* be made up of entries; they cannot have primitive types as entries. For example, if you defined:

```
foo OBJECT-TYPE
        SYNTAX SEQUENCE OF Foo-Entry
        ACCESS read-only
        STATUS mandatory
        ::= { bar 1 }
```

   you must ensure that Foo-Entry is defined as a SEQUENCE constructor like:

```
Foo-Entry ::= SEQUENCE {
        entry1
                INTEGER,
        entry2,
                INTEGER
}
```

You may not define any other type for Foo-Entry.

Warning/Error Messages

This section describes the warning or error messages **v2mib2schema** generates and the actions you should take to correct your MIB. Warning messages indicate an inconsistency in your MIB, but nothing serious enough to prevent the generation of a schema file. Warning messages are preceded by the word "Warning" and the line number in your MIB that caused the message to be generated. Error messages are usually fatal, and cause **v2mib2schema to abort.** Error messages are accompanied by messages displaying the last token that was read from the MIB and the line number of that token in the MIB.

Warning Messages

"does not recognize IMPORTED symbol *[symbol-name]*"
: **v2mib2schema** only recognizes keywords and objects from the SMI, MIB-I, MIB-II, the Concise MIB revisions and the TRAP-TYPE macro. This warning message is generated if *symbol-name* is not an object from any of the aforementioned documents. *symbol-name* will have to be redefined in the MIB.

"*[symbol-name]* previously defined"
: If *symbol-name* is defined more than twice, this message is generated. When the *symbol-name* happens to be a group name, it is renamed by inserting its parent node name before its own name. This is necessary since the SNM schema format

cannot accept duplicate group names.

"Object Identifier cannot be 0"
: As defined in the SMI, you may not use 0 as an object identifier, since 0 is reserved for instances.

"Enumeration Element Value cannot be 0"
: As defined in the SMI, you may not use 0 as an enumerated value.

"*[symbol-name]* is not a valid status type"
: As defined in the SMI, you may only use the following status types: "deprecated", "mandatory", "obsolete" and "optional". However, since SNM schema ignores the status types, only a warning message is generated.

"The following INDEX entries in *table* not resolved: *name*"
: If **v2mib2schema** is unable to determine the key value for table attributes, it inserts the characters -K ??? into the characteristics field. Users of SunNet Manager 2.x do not need to take any action. If you intend to use the generated schema file with SNM release 1.x, then you should edit the schema and manually change the occurrences of -K ??? with the appropriate key value.

Error Messages

"Fatal Error: *[symbol-name]* not defined"
: This message is generated if *symbol-name* is part of an object's component ID and if it was not previously defined. Typos are the main reason this message is generated.

"Fatal Error: *[symbol-name]* type not resolved"
: This message is generated when *symbol-name* is used as a type by an object variable, but is not defined anywhere in the MIB. Since the object's type cannot be resolved, the translation has to be aborted.

"Fatal error: *[name]* unknown enterprise in trap definition"
: This message is generated when an unknown enterprise is encountered in trap definitions. *name* is the unknown enterprise name.

BUGS

As mentioned above, **v2mib2schema** is not a compiler for MIB specifications. Hence, the behavior of **v2mib2schema** is not guaranteed if the MIB specification is syntactically incorrect. Agents are not required to accept messages that exceed 484 octets. For groups with very long var-bind entries, you may have to split the group into groups with smaller var-bind entries.

SEE ALSO

build_oid(1), **snm_schema**(5), **mib2schema**(1), **na.snmpv2**(8), **na.snmp-trap**(8)

NAME

view.pty – MIB view access configuration for SNMPv2 entities

SYNOPSIS

viewIndex viewSubtree viewType viewStorageType
viewMask | –

DESCRIPTION

The configuration file **view.pty** is one of several configuration files required by the SNMPv2 entities. The default location of **view.pty** is **/etc/opt/snm/agent** for Solaris 2.x and **/etc/snm/agent** for Solaris 1.x, or **/etc/opt/snm/manager** (Solaris 2.x) and **/etc/snm/manager** (Solaris 1.x) but can be specified by the environment variable **SR_MGR_CONF_DIR** or **SR_AGT_CONF_DIR.**

The **view.pty** file contains the configuration information for the view table which defines the MIB view information for the SNMPv2 parties in the party table.

Each entry in the file consists of 2 lines:

viewIndex viewSubtree viewType viewStorageType
viewMask | –

where

viewIndex
is the *contextViewIndex* 0 or more contexts whose MIB view is being defined.

This value is an integer.

viewSubtree
in combination with *viewMask*, defines a family of MIB view subtrees. This family is included in or excluded from the MIB view for the context.

This value is an OID.

Note: If an OID is duplicated, the one which occurs first will be the one used and all subsequent OIDs will be discarded.

For a more detailed description, please see the Party MIB (RFC1447) and **context.pty**(5).

viewType
indicates the status for this view entry. It can have a value of either

included, which means this MIB view is included in the context's MIB view,

or

excluded, which means that this MIB view is not included in the context's MIB view.

viewStorageType
indicates the storage type for this row in the view table. Possible values are:

other
volatile
nonVolatile

permanent

According to RFC1447,

- volatile is lost upon reboot, e. g., in RAM,
- nonVolatile is backed up by stable storage, e. g., in NVRAM,
- permanent cannot be changed or deleted, e. g., in ROM,

and "other" is provided in the unlikely event that someone will find a need for a storage type not covered by the other three.

This field is a case-sensitive string corresponding to one of the above values.

viewMask

allows restriction of the view at a finer level than the *viewSubtree* and *viewType* pair. For instance, a view can be restricted to one row of a table. See the example below.

The value " – " causes the corresponding *viewMask* to be a null string, which in turn allows all entries "below" the *viewSubtree* entry to be visible, unless canceled by another *view.pty* entry.

The *viewMask* string can have a length from zero to sixteen bytes (so from " – " to 16 hexadecimal digits.)

Note: All positions of the numbers must be represented; i. e., 01 for 1.

For more information, it is strongly suggested that you read the security documents, RFC1446, and RFC1447.

EXAMPLES

Simple Subtree Inclusion

An entry that would include a subtree for viewing might look like

```
1 system included nonVolatile
–
```

This indicates that the SNMPv2 parties that have a SNMPv2 context with *contextViewIndex of 1*, have the *viewSubtree* of *system* included in their MIB view, and this view entry should be stored in non-volatile storage such as NVRAM. In other words, any parties with this *contextViewIndex* can "see" the system group unless limited by another entry.

Note: If an OID is duplicated, the one which occurs first will be the one used and all subsequent OIDs will be discarded.

Simple Subtree Exclusion

An entry that would exclude a subtree for viewing might look like

```
2 interfaces excluded nonVolatile
–
```

This indicates that the SNMPv2 parties that have a SNMPv2 context with *contextViewIndex* of *2*, have the *viewSubtree* of *interfaces* excluded in their MIB view, and this view entry should be stored in non-volatile storage such as NVRAM. In other words, any parties that have a SNMPv2 context with this *contextViewIndex* cannot "see" the interfaces group unless allowed by another entry.

Note: If an OID is duplicated, the one which occurs first will be the one used and all subsequent OIDs will be discarded.

Using the viewMask to Limit the View

The *viewMask* string is convoluted. Prepare for serious obfuscation:

The *viewMask* is built from octets that correspond to the OID being restricted. Say you want to restrict a *contextViewIndex* entry's view of the *ifTable* to the second row, all columns. The first three fields of the view entry would be:

3 ifEntry.0.2 included

A null entry for *viewMask* will not work because nothing will ever exactly match the *viewSubtree* OID, ifEntry.0.2. Specifying the ".0.2" on the *ifEntry* OID created two place holders that will be exploited by the *viewMask*.

The OID for ifEntry.0.2 would be:

1.3.6.1.2.1.2.2.1.0.2

The *viewMask* is a series of ones and zeros used for masking out parts of the tree. A zero indicates a "wildcard" (i. e., matches anything), a one indicates an exact match must be made. So

OID: 1.3.6.1.2.1.2.2.1.0.2
viewMask: 1 1 1 1 1 1 1 1 1 0 1

would require an exact match on all fields *except* the table column (i. e., the 0 in ifEntry.0.2).

Now for the obfuscation. The octets within the *viewMask* are rotated within bytes. Using the above example, you would first group the bits of the *viewMask* into bytes, padding the right end with ones if necessary, then rewrite each byte from right to left, but leaving the bytes "in order:"

```
   byte 1          byte 2
1 1 1 1  1 1 1 1  1 0 1          original mask
1 1 1 1  1 1 1 1  1 0 1 1  1 1 1 1  padded with 1's
1 1 1 1  1 1 1 1  1 1 1 1  1 1 0 1  rotated mask
   ff              fd       hex value
```

Notice that bit 1 of each original byte was "rotated" to the 8th position, bit 2 to the 7th, bit 3 to the 6th, etc.

So the *viewMask* entry would be:

ff fd

With this entry in *view.pty*, and all appropriate entries in the other configuration files, a *getmany(l)* on the *ifTable* will return something like:

ifIndex.2 = 2
ifDescr.2 = lo0
ifType.2 = softwareLoopback(24)
ifMtu.2 = 1536

ifSpeed.2 = 0
ifPhysAddress.2 =
ifAdminStatus.2 = up(1)
ifOperStatus.2 = up(1)
ifLastChange.2 = 0
ifInUcastPkts.2 = 182945
ifInErrors.2 = 0
ifOutUcastPkts.2 = 182949
ifOutErrors.2 = 0
ifOutQLen.2 = 0
ifSpecific.2 = ccitt.0

Of course, the second row would have to exist before you could retrieve it.

Note: If an OID is duplicated, the one which occurs first will be the one used and all subsequent OIDs will be discarded.

A Sample view.pty File

The following is the default **view.pty** file shipped in August of 1993. It allows all parties using *contextViewIndex* of 1 to see the system, snmpStats, snmpParties, snmpTrap, and snmpTraps groups. All parties using *contextViewIndex* of 2 are allowed to see everything under the internet group, and all parties using *contextViewIndex* of 3 are allowed to see the *partyAuthClock* only.

This last entry is the *contextViewIndex* used by the "clock synch" routines.

1 system included nonVolatile
–
1 snmpStats included nonVolatile
–
1 snmpParties included nonVolatile
–
1 snmpTrap included nonVolatile
–
1 snmpTraps included nonVolatile
–
2 internet included nonVolatile
–
3 partyAuthClock included nonVolatile
–

FILES

Additional SNMPv2 Configuration Files

When the agent is compiled with either SNMPv2 or both SNMPv1 and SNMPv2 defined, the configuration files **acl.pty, agt.pty, context.pty, mgr.cnf,** and **mgr.pty** are required.

acl.pty Access control privileges for the SNMPv2 parties.

agt.pty Initial party table information for the agent.

context.pty
Context information for the SNMPv2 parties.

mgr.cnf
Configuration information for the managers.

mgr.pty
Initial party table information for the managers.

snmpv2d.conf
Configuration information for the SNMPv1 entities.

snmpinfo.dat
OID translations used by the entities.

view.pty
MIB view information for the SNMPv2 parties.

For Solaris 2.x, the files are located under:

/etc/opt/snm/{agent,manager}/acl.pty
/etc/opt/snm/agent/agent.pty
/etc/opt/snm/{agent,manager}/context.pty
/etc/opt/snm/manager/manager.cnf
/etc/opt/snm/manager/manager.pty
/etc/opt/snm/manager/snmpinfo.dat
/etc/opt/snm/agent/snmpv2d.conf
/etc/opt/snm/{agent,manager}/view.pty

For Solaris 1.x, the files are located under:

/etc/snm/{agent,manager}/acl.pty
/etc/snm/agent/agent.pty
/etc/snm/{agent,manager}/context.pty
/etc/snm/manager/manager.cnf
/etc/snm/manager/manager.pty
/etc/snm/manager/snmpinfo.dat
/etc/snm/agent/snmpv2d.conf
/etc/snm/{agent,manager}/view.pty

SEE ALSO

v2install(1), **acl.pty**(5), **agt.pty**(5), **context.pty**(5), **mgr.cnf**(5), **mgr.pty**(5), **snmpv2d.conf**(5), SNMP RFCs (RFC1155 RFC1157 RFC1212 RFC1213 RFC1215, RFCs 1441-1452)

Index

S

V