Solaris System Management Agent Administration Guide

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 817–3000–10
January 2005

040903@9495

Contents

Tables

Figures

Preface

The *Solaris System Management Agent Administration Guide* explains how to install, configure and work with the System Management Agent (SMA).

Note – This Solaris™ release supports systems that use the SPARC® and x86 families of processor architectures: UltraSPARC®, SPARC64, AMD64, Pentium, and Xeon EM64T. The supported systems appear in the *Solaris 10 Hardware Compatibility List* at `http://www.sun.com/bigadmin/hcl`. This document cites any implementation differences between the platform types.

In this document the term "x86" refers to 64-bit and 32-bit systems manufactured using processors compatible with the AMD64 or Intel Xeon/Pentium product families. For supported systems, see the *Solaris 10 Hardware Compatibility List*.

Who Should Use This Book

This document is targeted towards system administrators who need to use the System Management Agent to manage objects and devices on the Solaris Operating System. Also, system administrators who wish to migrate management agent tasks from other agents to the System Management Agent will should use this book.

Before You Read This Book

You must be familiar with general system administration of the Solaris system. A general understanding of SNMP and SNMP MIBs is beneficial. You should be familiar with the following areas:

- SNMPv1, SNMPv2c, and SNMPv3 protocols
- Structure of Management Information (SMI) v1 and v2
- Management Information Base (MIB) structure
- Abstract Syntax Notation (ASN.1)

How This Book Is Organized

This book is organized into the following chapters:

Chapter 1 provides an overview of SNMP and an introduction to the System Management Agent.

Chapter 2 describes the files you can use to configure the System Management Agent.

Chapter 3 shows you basic operations, such as stopping or restarting the System Management Agent.

Chapter 4 describes security administration and user management.

Chapter 5 shows you how to migrate the management of your devices to the System Management Agent from other agents.

Appendix A provides some reference information about the man pages, tools, and utilities that are provided with the System Management Agent.

Glossary is a list of key words and phrases found in this book and their definitions.

Recommended Reading

Ideally, you should be familiar with SNMP and SNMP MIBs.

- *Internet Engineering Task Force RFC 2741 on AgentX:* `http://www.ietf.org/rfc/rfc2741.txt`.

- *Internet Engineering Task Force RFC 3411 on An Architecture for Describing Simple Network Management Protocol Management Frameworks:* `http://www.ietf.org/rfc/rfc3411.txt`.
- *Understanding SNMP MIBs* by Perkins and McGinnis (Prentice Hall).
- *Solaris System Management Agent Developer's Guide*.
- *System Administration Guide: Advanced Administration*.
- *System Administration Guide: Security Services*.

Accessing Sun Documentation Online

The docs.sun.com[SM] Web site enables you to access Sun technical documentation online. You can browse the docs.sun.com archive or search for a specific book title or subject. The URL is `http://docs.sun.com`.

Ordering Sun Documentation

Sun Microsystems offers select product documentation in print. For a list of documents and how to order them, see "Buy printed documentation" at `http://docs.sun.com`.

Typographic Conventions

The following table describes the typographic changes that are used in this book.

TABLE P–1 Typographic Conventions

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`

TABLE P–1 Typographic Conventions *(Continued)*

Typeface or Symbol	Meaning	Example
AaBbCc123	What you type, contrasted with onscreen computer output	`machine_name%` **`su`** `Password:`
AaBbCc123	Command-line placeholder: replace with a real name or value	The command to remove a file is `rm` *filename*.
AaBbCc123	Book titles, new terms, and terms to be emphasized	Read Chapter 6 in the *User's Guide*. These are called *class* options. Do *not* save the file. (Emphasis sometimes appears in bold online.)

Shell Prompts in Command Examples

The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P–2 Shell Prompts

Shell	Prompt
C shell prompt	`machine_name%`
C shell superuser prompt	`machine_name#`
Bourne shell and Korn shell prompt	`$`
Bourne shell and Korn shell superuser prompt	`#`

CHAPTER 1

Introduction to the System Management Agent

The System Management Agent is the Sun Microsystems implementation of the open source Net-SNMP agent. This chapter describes the key principles of SNMP. This chapter also provides an overview of the System Management Agent.

This chapter contains information on the following topics:

Overview of SNMP and Network Management

The Simple Network Management Protocol (SNMP) is an Internet standard. SNMP provides a common way to query, monitor, and manage devices connected to IP networks. The protocol is defined in RFC 2571. For more information, see `http://www.ietf.org/rfc/rfc2571.txt`. Many details of SNMP are further defined in other RFCs.

SNMP is widely used in enterprise networks to effectively manage systems, network devices, and networks. One of the benefits of SNMP is how quickly solutions can be created to support the increasing numbers of networking components and applications. Within SNMP networks, systems, components, and applications are described as *entities*. The number of entities that need to be managed is growing rapidly.

SNMP uses a *manager* and *agent* architecture. The SNMP manager is a program, also known as a network management station (NMS), that runs on a host on the network. The manager sends requests to one or more SNMP agents running on devices connected to the network. An agent, or daemon, is a program that listens for SNMP requests from the manager.

Agent hierarchy consists of a master agent and subagents. The master agent receives the SNMP-based management requests from the managers. The master agent sends responses to these management requests. Responses are sent after retrieving the appropriate values from respective subagents.

Subagents provide management of different components. Management is based on a Management Information Base (MIB) specifically designed for components or applications. A MIB is a specification containing definitions of management information. Through the use of a MIB: networks and networked systems can be remotely monitored, remotely configured, and remotely controlled.

An agent receives a request and looks up information in the MIB and returns information to the manager. Each object in the MIB represents a piece of data about the managed device, and each object is assigned a unique identifier in the MIB. The manager and agent must have access to the same MIB to be able to communicate about the managed device. The manager uses the MIB to specify identifiers for the information that the agent is to act upon. The agent uses the MIB to look up the identifiers that were passed in the SNMP request from the manager. The agent gets or sets values for the requested data. The MIBs supported by the System Management Agent are listed in "Supported MIBs" on page 22.

SNMP Versions

The System Management Agent supports three SNMP protocols. Along with their associated RFCs, these protocols are:

SNMP v1 — SNMP v1 is defined in RFC 1155 and 1157 at `http://www.ietf.org/rfc/rfc1155.txt` and `http://www.ietf.org/rfc/rfc1157.txt`

SNMP v2c — SNMP v2c is defined in RFC 1901 at `http://www.ietf.org/rfc/rfc1901.txt`

SNMP v3 — SNMP v3 is defined in RFC 2570 at `http://www.ietf.org/rfc/rfc2570.txt`

These versions of SNMP supported by the System Management Agent can co-exist following the guidelines laid down in RFC 3584 at `http://www.ietf.org/rfc/rfc3584.txt`.

Some security models and other instances described in this manual do not support all versions of SNMP. Restrictions regarding which version of SNMP you can use are indicated in this book and in the relevant man pages. Restrictions are due in part to the enhanced packet structure of SNMPv3. The SNMPv3 packet structure is shown in Figure 1–1.

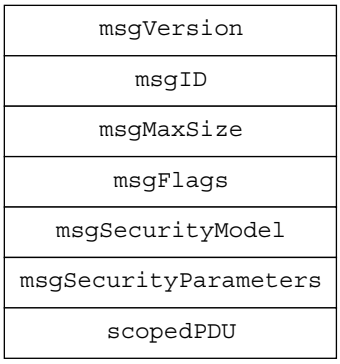

FIGURE 1–1 SNMPv3 Packet Structure

The packets outlined in Figure 1–1 are:

`msgVersion`	The SNMP version of the packet. Possible values are 1, 2, or, in the case of SNMPv3, 3.
`msgID`	Used to coordinate request and response messages between the manager and the agent. The `msgID` in a response must be the same as the `msgID` in a request.
`msgMaxSize`	Conveys the maximum size of a message that the sender can accept from another SNMP engine.
`msgFlags`	A single octet to indicate how the message is to be processed. For more information, see "Where VACM Security Information Is Contained" on page 47.
`msgSecurityModel`	Specifies the security model used to generate the message. For more information, see "Where VACM Security Information Is Contained" on page 47.
`msgSecurityParameters`	An octet string containing data about the security model. For more information, see "Where VACM Security Information Is Contained" on page 47.
`scopedPDU`	Contains the normal Protocol Data Unit (PDU) and information for identifying the administratively unique context for processing the PDU. For more information, see "Where VACM Security Information Is Contained" on page 47.

Structure of Management Information

The writing of MIBs is governed by a set of rules known as the Structure of Management Information, (SMI). This set of documents contains industry-accepted methods and rules for specifying the following information:

- The model of management information
- Types of management information
- Types of events

The System Management Agent uses SMIv2. SMIv2 instructs about organization object names so that logical access can occur. SMIv2 states that each managed object must have the following attributes:

A name	The name, an object identifier (OID), uniquely identifies the object. The assignment of an OID value to an object registers that object. For more information, see "ISO Namespace Tree" on page 22.
A syntax	The syntax defines the data type, such as an integer or a string of octets.
An encoding	The encoding describes how the information associated with the managed objects is serialized for transmission between machines.

Community String

In SNMP, one or several managers together with an agent is known as a community. SNMPv1 and v2c messages contain the name of a community, known as a *community string*. While SNMPv3 packets are associated to users specified in USM settings, SNMPv2 and v1 packets have an associated community string. The community string is an octet string variable used for the following checks:

- Identifying the requesting entity.
- Indicating the location of the requesting entity.
- Determining the MIB view information.

The VACM supported by the SMA elaborates on the community string model with a dynamic access control model. The dynamic access control model for SNMPv3 is explained in "Using VACM for Access Control" on page 46.

The `com2sec` token maps a community to an SNMPv3 security name, so that the community can use VACM views. For more information, see Chapter 4.

Overview of the System Management Agent

The System Management Agent implements RFC 3411 at `http://www.ietf.org/rfc/rfc3411.txt`. The SMA is a lightweight agent that uses SNMP protocols for the management of systems. The SMA provides a standardized SNMP agent infrastructure to the Solaris software. The SMA can be extended through the use of modules written to application programming interfaces and Agent X subagents. For information on extending modules in the System Management Agent, see *Solaris System Management Agent Developer's Guide*. For information about AgentX, see `http://www.ietf.org/rfc/rfc2741.txt`.

The System Management Agent is designed to be a standalone agent. The SMA can be accessed by multiple management applications, provided that these management applications communicate with the SMA using SNMP protocols. The SMA can coexist with existing SNMP agents. The SMA replaces some legacy SNMP agents.

The SMA is a new SNMP agent offering from Sun, based on the Net-SNMP open source implementation version 5.0.9. This open source implementation is described at `http://www.net-snmp.org/`. This open source implementation was formerly known as UCD-SNMP. The System Management Agent is designed to support the latest SNMP standards.

In this Solaris release, the System Management Agent can co-exist with the Solstice Enterprise Agents™ software. For more information about the Solstice Enterprise Agents software, see the *Solstice Enterprise Agents 1.0 User Guide*. From an SNMP manager view, the System Management Agent operates in the same way the Solstice Enterprise Agents software. Unlike the Solstice Enterprise Agents software, the System Management Agent supports SNMPv3. The System Management Agent supports more default MIBs than the Solstice Enterprise Agents software.

For information about migration from the Solstice Enterprise Agents to the System Management Agent, see "Migration From Solstice Enterprise Agents Software" on page 65. For information about migrating your applications from the Solstice Enterprise Agents to the System Management Agent, see the *Solaris System Management Agent Developer's Guide*.

System Management Agent Components

The System Management Agent implements the agent component of standards relating to the SNMP management framework. Several standards that form part of this framework. These standards include the following:

- AgentX Protocol: industry standard mechanism, defined in RFC 2741 at: `http://www.ietf.org/rfc/rfc2741.txt`.
- USM: User based Security Model for authentication and privacy, defined in RFC 3414 at: `http://www.ietf.org/rfc/rfc3414.txt`. See also "Using USM for Authentication and Message Privacy" on page 42.
- VACM: View based Access Control Model for authorization, defined in RFC 3415 at: `http://www.ietf.org/rfc/rfc3415.txt`. See also "Using VACM for Access Control" on page 46.

For details of other associated RFCs, see "Supported MIBs" on page 22. The System Management Agent is configurable. Command line tools are provided to handle configuration as well as other simple SNMP operations. The System Management Agent can be extended through dynamic modules as well as Agent-X subagents. For more information, see *Solaris System Management Agent Developer's Guide*.

The various packages that are included in the System Management Agent are outlined in "Platforms and Packages" on page 25.

The relationship of some of the components in the System Management Agent is illustrated by Figure 1–2.

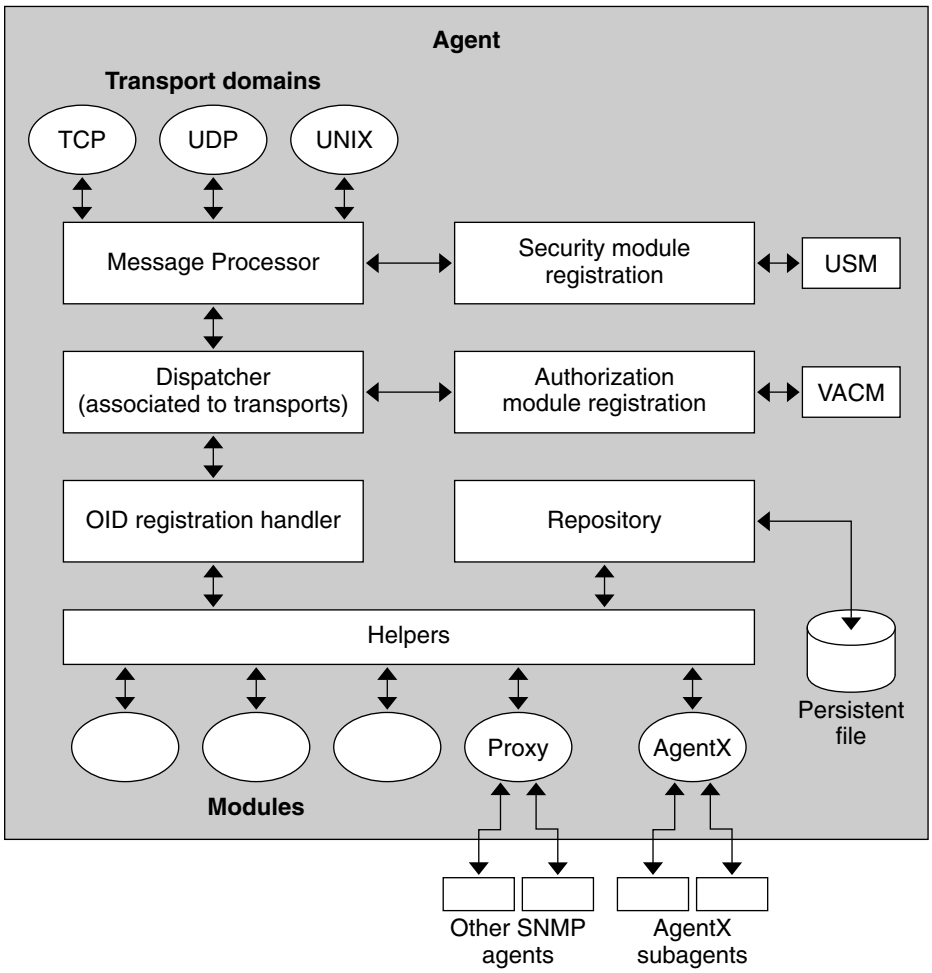

FIGURE 1–2 Components of the System Management Agent

This diagram shows the inter-relationship of the message processor, dispatcher and the programs that handle OID registration, with security and authorization. The diagram depicts other SNMP agents interacting with the System Management Agent by means of a proxy. The diagram also shows that AgentX subagents interact with the System Management Agent through the AgentX protocol. For further information on AgentX, see "Using the AgentX Protocol" on page 31. For further information on the interaction of the components described in Figure 1–2, see "Overview of the System Management Agent" in *Solaris System Management Agent Developer's Guide*.

ISO Namespace Tree

Every managed object, whether a device or the characteristics of a device, has a name, a syntax, and an encoding. The name, an object identifier (OID), uniquely identifies the object. The OID is written as a sequence of integers separated by periods. For example, the sequence 1.3.6.1.2.1.1.1.0 specifies the system description within the system group of the management subtree. The OID scheme was created partly by the ISO organization. The ISO organization gives its name to the rooted tree diagrams used to represent OID values. An ISO diagram of the overall System Management Agent is shown in Figure 1–3.

The OID for SMA is `1.3.6.1.4.1.42.2.2.4`

This OID corresponds to the data:

`iso.org.dod.internet.private.enterprises.sun.products.management.sma`

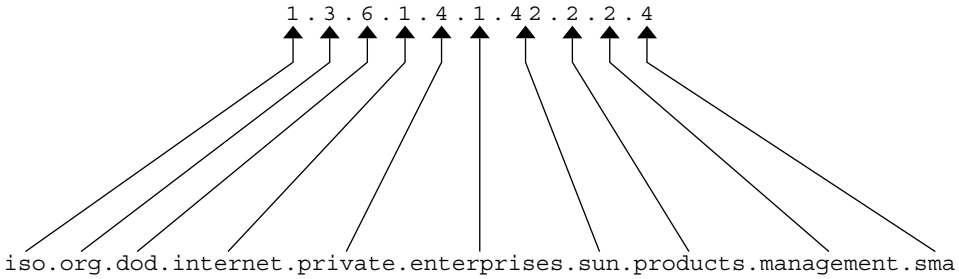

FIGURE 1–3 ISO Namespace Tree Diagram

Supported MIBs

The System Management Agent supports the following MIBs

SNMP-COMMUNITY MIB
Defined in RFC 2576. See `http://www.ietf.org/rfc/rfc2576.txt`

SNMPv2-TM (Transport Mappings)
Defined in RFC 3417. See `http://www.ietf.org/rfc/rfc3417.txt`

SNMP-MPD-MIB (Message Processing and Dispatching)
Defined in RFC 3412. See `http://www.ietf.org/rfc/rfc3412.txt`

SNMP-TARGET-MIB (Specification of targets for traps)
Defined in RFC 3413. See `http://www.ietf.org/rfc/rfc3413.txt`

SNMP-NOTIFICATION-MIB (Trap filtering)
Defined in RFC 3413. See `http://www.ietf.org/rfc/rfc3413.txt`

SNMP-PROXY-MIB (Trap forwarding)
Defined in RFC 3413. See `http://www.ietf.org/rfc/rfc3413.txt`

SNMP-USER-BASED-SM-MIB (User-based Security Model for SNMPv3)
Defined in RFC 3414. See `http://www.ietf.org/rfc/rfc3414.txt`

SNMP-VIEW_BASED-ACM-MIB (View-based Access Control Model for SNMP)
Defined in RFC 3415. See `http://www.ietf.org/rfc/rfc3415.txt`

SNMPv2-MIB
Defined in RFC 3418. See `http://www.ietf.org/rfc/rfc3418.txt`

MIB II
Defined in RFC 1213. See `http://www.ietf.org/rfc/rfc1213.txt`

Host Resources MIB
Defined in RFC 2790. See `http://www.ietf.org/rfc/rfc2790.txt`

Sun MIB
Related to migration from the Solstice Enterprise Agents software. For further information, see "Migration From Solstice Enterprise Agents Software" on page 65. For information about the migration of applications from the Solstice Enterprise Agents software, see "Migration of Solstice Enterprise Agents to the System Management Agent" in *Solaris System Management Agent Developer's Guide*.

You can see a list of those MIBs that are initialized after starting the System Management Agent by following the procedure described in "To See Which MIBs Are Initialized" on page 35.

You can see the text files of MIB definitions at `/etc/sma/snmp/mibs/`.

CHAPTER 2

Configuring the System Management Agent

This chapter describes how to configure the System Management Agent for use within your network. This chapter covers the configuration files and security features of the System Management Agent. This chapter contains material on the following topics:

Platforms and Packages

The System Management Agent is bundled with this Solaris release. To install the System Management Agent on the Solaris software, follow the standard procedures for bundled products. These procedures are described in the *Solaris 10 Installation Guide: Basic* and in the *Solaris 10 Installation Guide: Advanced, Custom JumpStart*. Note that you only need to use the `pkgadd` command to install the packages of the System Management Agent if you have previously removed these packages.

All of the packages of the System Management Agent are bundled with this Solaris release. These packages are automatically installed when you boot your Solaris system.

The SMA package is divided into two parts, because the Solaris 10 Operating System is supported on the SPARC and x86 platforms.

The runtime SMA product includes the following unique packages:

`SUNWsmaS` The `SUNWsmaS` package contains the source files that are needed to rebuild the System Management Agent. These source files comprise the source code of Net-SNMP version 5.0.9. The files contained in `SUNWsmaS` are useful for configuring a lightweight daemon. As a

source code package, SUNWsmaS is not installed by default with the Solaris software. You can install this package from the mounted CD, by using the pkgadd command:

```
# pkgadd -d /sol_10_sparc_2/Solaris_10/Product SUNWsmaS
```

SUNWsmagt — The SUNWsmagt package contains the 32–bit and 64–bit libraries. In addition, this package contains both the snmpd agent and the snmptrapd trap daemon. The package also contains header files, which are required to build the SMA.

SUNWsmcmd — The SUNWsmcmd package contains the SMA SNMP applications and utilities. These applications and utilities include developer tools such as snmpget, and Perl scripts such as mib2c. In addition, the SUNWsmcmd package contains the SDK demo modules. The SDK demo modules illustrate how to implement some types of data modeling. For more information about the demo modules, see the *Solaris System Management Agent Developer's Guide*.

SUNWsmdoc — The SUNWsmdoc package contains the HTML documentation files for the SMA. These files are generated from Net-SNMP source. Do not confuse these generated HTML files with the product documentation for the SMA. The product documentation for the SMA includes this document, the *Solaris System Management Agent Developer's Guide* and the man pages. This product documentation is provided by Sun.

SUNWsmmgr — The SUNWsmmgr package contains all files that are installed under /etc/sma, including:

- The SMA start up script, init.sma.
- All MIBs. For more information, see "Supported MIBs" on page 22.
- Default snmpd.conf files. For more information, see "Configuration Files and Scripts" on page 28.
- The helper scripts related to mib2c. For more information about mib2c, see the *Solaris System Management Agent Developer's Guide*.

Removing Packages

If you remove the packages that are mentioned in this chapter, you remove all files related to the System Management Agent.

Note – Ensure that you stop the System Management Agent before uninstalling it. Failure to stop the agent before removing the packages can cause agent files in various locations to remain installed even after the packages are removed. Stop the agent before removing packages, to remove those files created the first time that the agent was initially started. For information about stopping the agent, see "Starting and Stopping the System Management Agent" on page 33.

Before uninstalling any packages, log in as root. Then use the following procedure to uninstall the packages.

▼ To Uninstall the Packages of the System Management Agent

1. **As root, launch the `init.sma` script and use the `stop` option.**

   ```
   # /etc/init.d/init.sma stop
   ```

2. **Remove the `SUNWsmaS` package.**

   ```
   # pkgrm SUNWsmaS
   ```

3. **Remove the `SUNWsmdoc` package.**

   ```
   # pkgrm SUNWsmdoc
   ```

4. **Remove the `SUNWsmcmd` package.**

   ```
   # pkgrm SUNWsmcmd
   ```

5. **Remove the `SUNWsmmgr` package.**

   ```
   # pkgrm SUNWsmmgr
   ```

6. **Remove the `SUNWsmagt` package.**

   ```
   # pkgrm SUNWsmagt
   ```

Provided that you stopped the System Management Agent before removing these `SUNW` packages, the following files and their persistent stores are removed if they exist:

- `/etc/sma/snmp/snmptrapd.conf`
- `/etc/sma/snmp/snmp.conf`
- `/var/sma_snmp/snmp.conf`
- `/var/sma_snmp/snmptrapd.conf`

Default Software Locations

The daemon of the System Management Agent is named, `snmpd`. This daemon is located in the `/usr/sfw/sbin/` directory.

The trap daemon of the System Management Agent is named `snmptrapd`. This trap daemon is located in the`/usr/sfw/sbin/` directory.

After using traps, the file `snmptrapd.conf` is created.

The main configuration file of the, System Management Agent is named `snmpd.conf`. This configuration file is installed by default in the `/etc/sma/snmp/` directory. For more information about the `snmpd.conf` file, see "Configuration Files and Scripts" on page 28.

The files created when the `snmpd` daemon starts up are as follows:

- `/etc/sma/snmp/mibs/.index`
- `/var/log/snmpd.log`
- `/var/sma_snmp/snmpd.conf` (persistent file)

Library files for 32–bit x86 platforms are placed in the `/usr/sfw/lib` directory.

Library files for 64–bit SPARC platforms are placed in the `/usr/sfw/lib/sparcv9` directory.

Configuration scripts and other commands are placed in `/usr/sfw/bin`.

Configuration Files and Scripts

As a standard Net-SNMP implementation, configuration of the System Management Agent can be done principally through the `snmpd.conf` user configuration file. Usage of this file is described in "Managing Configuration With the Main Configuration File" on page 30. For configuring the default settings that the System Management Agent uses, a separate configuration file, named `snmp.conf`, is provided. This file is described in Appendix A.

Some configuration files, scripts and man pages bear similar names. These files, scripts and man pages are summarized for clarity in the following list.

`snmpconf` — A script that helps you to create and modify SMA configuration files. The SMA `snmpconf` script is located in the `/usr/sfw/bin/` directory. An associated man page, `snmpconf`(1M), is provided.

`snmp.conf` — A configuration file for the System Management Agent. This file defines how applications operate. Use this file to configure default settings to reduce the number of required arguments when using SNMP commands, for example, in defining an SNMPv3 default user. An associated man page, `snmp.conf`(4), is provided.

`snmpd.conf` — Several files named `snmpd.conf` are provided in this Solaris release. These files +are as follows:

- The most important `snmpd.conf` file is the configuration file for operation of the System Management Agent. This file is located in the `/etc/sma/snmp` directory. An associated man page, `snmpd.conf`(4), is provided.
- The Solstice Enterprise Agents configuration file is also named `snmpd.conf`. This file is located in the `/etc/snmp/conf` directory.
- The persistent storage file of the SMA is also named `snmpd.conf`. This persistent storage file is located in the `/var/sma_snmp/` directory. This file is described in "Persistent Storage Files" on page 30.
- The template file used by a migration script for Sun Fire™ servers is also named `snmpd.conf`. This template file is located in the `/usr/sfw/lib/sma_snmp/` directory. Sun Fire servers use the migration script to modify the System Management Agent's main configuration file, `snmpd.conf`. For more information about migration to the SMA from the management agent for Sun Fire servers, see "Migration From the Sun Fire Management Agent" on page 71.

`snmpd` — Two `snmpd` daemons are provided in this Solaris release:

- The System Management Agent `snmpd` daemon is the SNMP agent that executes requests of the SMA software. The SMA `snmpd` daemon is located in the `/usr/sfw/sbin/` directory. An associated man page, `snmpd`(1M), is provided.
- The Sun SNMP Management Agent for Sun Fire and Netra Systems (Sun SNMP Management Agent for Sun Fire and Netra Systems) uses an agent also named `snmpd`.

`sma_snmp` — Overall introductory man page for the System Management Agent. An alias, or alternative name, for the `sma_snmp`(5) man page is the `netsnmp`(5) man page.

`snmp_config` — The `snmp_config`(4) man page provides an overview of the System Management Agent configuration file, `snmpd.conf`.

For further information see "Man Pages" on page 75.

Persistent Storage Files

The persistent storage file, `/var/sma_snmp/snmpd.conf`, contains USM security information and any MIB components that are set for persistent storage. This file also contains the `engineID` and the `engineID` boots. This persistent storage file is automatically updated when the System Management Agent starts. When the System Management Agent stops, the `snmpusm` and `snmpvacm` utilities write user security information to this storage file. For more information on security, see Chapter 4.

The persistent storage file is generated based upon the tokens that are placed in the main user configuration file in `/etc/sma/snmp/snmpd.conf`. For more information, see the `snmpd.conf`(4) man page.

Managing Configuration With the Main Configuration File

The main configuration file that is shipped with the System Management Agent is the `snmpd.conf` file. This file is located in the `/etc/sma/snmp` directory. A minimal version is provided as a standard template to help you get started.

Various tokens are available as with standard Net-SNMP for managing configuration. These tokens are managed through the `snmpd.conf` file. Each of these tokens has an `init` module that runs when the System Management Agent starts.

In addition to the standard Net-SNMP implementation, some extra modules are provided with the System Management Agent. These extra modules include the `seaProxy` module and the `seaExtensions` module, described in "Migration From Solstice Enterprise Agents Software" on page 65.

For more information about the `snmpd.conf` file, see the `snmpd.conf`(4) man page.

As an SNMP agent, the System Management Agent must run on port 161. If another process is running on port 161, the System Management Agent does not start. To see if the System Management Agent is not starting because another agent is running at port 161, check the contents of the `/var/log/snmpd.log` log file. This log file also details any other errors that might occur at startup.

Using the AgentX Protocol

The AgentX protocol is supported in the System Management Agent. By default, the System Management Agent ships with a secure profile, that is, read–only access. AgentX allows interaction with third party subagents, provided that these subagents support AgentX over Unix Domain Sockets. For security reasons, AgentX is not supported over TCP/UDP. For more information on the AgentX protocol, see `http://www.ietf.org/rfc/rfc2741.txt`

Edit the main `/etc/sma/snmp/snmpd.conf` configuration file to configure the System Management Agent to use the AgentX protocol. By default, the AgentX protocol is disabled. The following procedure describes how to enable the AgentX protocol.

▼ To Enable the AgentX Protocol

1. **As root, edit the main `/etc/sma/snmp/snmpd.conf` configuration file.**

   Add the following line:

   ```
   master agentx
   ```

2. **Restart the System Management Agent.**

   ```
   # /etc/init.d/init.sma restart
   ```

Various other options can be set for the AgentX protocol. For example, you can set the timeout period for AgentX requests. These options are described in the `snmpd.conf`(4) man page.

CHAPTER 3

Working with the System Management Agent

This chapter describes how to manage common operations with the System Management Agent. This chapter contains material on the following topics:

Starting and Stopping the System Management Agent

Start or stop the SMA by starting or stopping the `snmpd` daemon. Numerous options are available for starting or stopping the daemon, but several of these options override the `snmpd.conf` and `snmp.conf` files. The recommended way to start and stop the System Management Agent is to use the `init.sma` script as described in this section. Further information on the `snmpd` daemon can be found in the `snmpd`(1M) man page.

Note – As a standard SNMP agent, the System Management Agent must run on port 161. If another process is running on port 161, the System Management Agent does not start.

▼ To Start the System Management Agent

1. **As root, launch the `init.sma` script. Use the `start` option:**

```
# /etc/init.d/init.sma start
```

2. **Check whether errors occurred in attempting to start the System Management Agent by examining the `/var/log/snmpd.log` file.**

   If the log file reports that the port 161 is occupied, follow the procedure described in "To Check Whether Another Process Is Running on the SMA Port" on page 35.

   For more information about the `init.sma` script, see the `init.sma`(1M) man page.

Note – Once the `init.sma` script has launched the System Management Agent on your system, the `snmpd` daemon always starts at boot time during the Solaris system boot. If you are using other agents, you might want to prevent the `snmpd` daemon from starting at boot time, which initializes the System Management Agent. If you want to prevent the `snmpd` daemon from starting at boot time, see "To Prevent The System Management Agent Initializing at Boot Time" on page 66.

You can also start the System Management Agent with the `snmpd` daemon:

```
# /usr/sfw/sbin/snmpd
```

Using the full path to start the `snmpd` daemon enables you to specify certain options. For example, you can add `-a` to append the log, or `-c` to specify an alternative configuration file. For more information, see the `snmpd`(1M) man page.

▼ To Restart the System Management Agent

To enable changes made to the main SMA configuration file, `/etc/sma/snmp/snmpd.conf`, a signal must be sent to the SMA daemon, `snmpd`. This signal reads the changes to `snmpd.conf` and restarts the System Management Agent.

- **As root, launch the `init.sma` script. Use the `restart` option:**

  ```
  # /etc/init.d/init.sma restart
  ```

  This method is the recommended way to restart the System Management Agent.

  Note – Use of the `kill` command with the `–HUP` option is not recommended as a way of re-reading the `snmpd.conf` file. If you use `kill -HUP` instead of the restart option described in this procedure, you risk losing data. The data that is held in the persistent storage file `/var/sma_snmp/snmpd.conf` is not saved. For example, you risk losing any new user or users that you added with the `snmpusm` command in the same session.

▼ To Stop the System Management Agent

- **As root, launch the `init.sma` script. Use the `stop` option:**

```
# /etc/init.d/init.sma stop
```

Common Operations With the System Management Agent

▼ To Check Whether Another Process Is Running on the SMA Port

Port 161 is reserved for the System Management Agent. For more information, see "Managing Configuration With the Main Configuration File" on page 30.

- **Use the `netstat` command:**

```
# netstat -anv|grep 161
```

If a value of 161 is returned, a process is already bound to port 161.

▼ To View the Status of the Agent

- **As root, launch the `init.sma` script. Use the `status` option:**

```
# /etc/init.d/init.sma status
```

A typical response to this command is shown:

```
snmpd running under PID 26466
```

▼ To See Which MIBs Are Initialized

Use one of the following methods to list those MIBs that are initialized when the SMA is started.

-
  - Examine the debug trace that is produced by running the following command:

```
# /usr/sfw/sbin/snmpd  -Dregister_mib -Dmib_init -L
```

- Alternatively, use the `net-snmp-config` command to see a list of those modules that are compiled.

  ```
  # /usr/sfw/bin/net-snmp-config --snmpd-module-list
  ```

▼ To Check the Disk Space on a Local or Remote Machine

First find the total disk space of the disk, then find how much of this space is used. The difference between these two totals is the available disk space.

1. **Find the number of disks that are available on a given host.**

   ```
   # snmpwalk -v1 -c public hostname HOST-RESOURCES-MIB::hrStorageIndex
   ```

   This command returns a list of disks on the host, *hostname*:

   ```
   HOST-RESOURCES-MIB::hrStorageIndex.1 = INTEGER: 1
   HOST-RESOURCES-MIB::hrStorageIndex.2 = INTEGER: 2
   HOST-RESOURCES-MIB::hrStorageIndex.3 = INTEGER: 3
   HOST-RESOURCES-MIB::hrStorageIndex.4 = INTEGER: 4
   HOST-RESOURCES-MIB::hrStorageIndex.5 = INTEGER: 5
   HOST-RESOURCES-MIB::hrStorageIndex.6 = INTEGER: 6
   HOST-RESOURCES-MIB::hrStorageIndex.7 = INTEGER: 7
   HOST-RESOURCES-MIB::hrStorageIndex.8 = INTEGER: 8
   HOST-RESOURCES-MIB::hrStorageIndex.9 = INTEGER: 9
   HOST-RESOURCES-MIB::hrStorageIndex.10 = INTEGER: 10
   HOST-RESOURCES-MIB::hrStorageIndex.101 = INTEGER: 101
   HOST-RESOURCES-MIB::hrStorageIndex.102 = INTEGER: 102
   ```

   The disk is indicated by the index number:

   ```
   HOST-RESOURCES-MIB::hrStorageIndex.1 = INTEGER: 1
   ```

   This output represents disk 1, `/dev/dsk/c0t0d0s0`

2. **Use the `snmpget` command to retrieve the total storage space for that disk.**

   The following command would retrieve the total storage space for disk 1:

   ```
   # snmpget -v1 -c public hostname HOST-RESOURCES-MIB::hrStorageSize.1
   ```

   This command returns the total disk space at the end of the line:

   ```
   HOST-RESOURCES-MIB::hrStorageSize.1 = INTEGER: 2561695
   ```

3. **View a list of the disk space used by each disk.**

   ```
   # snmpwalk -v1 -c public hostname HOST-RESOURCES-MIB::hrStorageUsed
   ```

   ```
   HOST-RESOURCES-MIB::hrStorageUsed.1 = INTEGER: 2121747
   HOST-RESOURCES-MIB::hrStorageUsed.2 = INTEGER: 0
   HOST-RESOURCES-MIB::hrStorageUsed.3 = INTEGER: 0
   HOST-RESOURCES-MIB::hrStorageUsed.4 = INTEGER: 0
   HOST-RESOURCES-MIB::hrStorageUsed.5 = INTEGER: 11
   HOST-RESOURCES-MIB::hrStorageUsed.6 = INTEGER: 48
   ```

```
HOST-RESOURCES-MIB::hrStorageUsed.7 = INTEGER: 1892576
HOST-RESOURCES-MIB::hrStorageUsed.8 = INTEGER: 0
HOST-RESOURCES-MIB::hrStorageUsed.9 = INTEGER: 130565552
HOST-RESOURCES-MIB::hrStorageUsed.10 = INTEGER: 26036932
HOST-RESOURCES-MIB::hrStorageUsed.101 = INTEGER: 55995
HOST-RESOURCES-MIB::hrStorageUsed.102 = INTEGER: 17171
```

4. **Use the `snmpget` command to view the storage used by the disk in question.**

   # **`snmpget -v1 -c public`** *hostname* **`HOST-RESOURCES-MIB::hrStorageUsed.1`**

   This command returns the disk space used on disk 1:

   ```
   HOST-RESOURCES-MIB::hrStorageUsed.1 = INTEGER: 2121747
   ```

5. **Subtract this figure from the total disk space to find the available disk space:**
   2561695 – 2121747 = 439948

The `snmpnetstat` command

In the same way as you would use the `netstat` command, you can check the status of the network using the System Management Agent with the `snmpnetstat` command.

To show the state of all sockets, use the `snmpnetstat` command with the `–a` option. This option provides the default display, showing all active sockets, except those used by server processes.

```
# snmpnetstat -v 2c -c public -a testhost
```

The following information, including local and remote addresses, and protocols, is typically displayed:

```
Active Internet (tcp) Connections (including servers)
     Proto Local Address Foreign Address (state)
     tcp   *.echo        *.*             LISTEN
     tcp   *.discard     *.*             LISTEN
     tcp   *.daytime     *.*             LISTEN
     tcp   *.chargen     *.*             LISTEN
     tcp   *.ftp         *.*             LISTEN
     tcp   *.telnet      *.*             LISTEN
     tcp   *.smtp        *.*             LISTEN

     Active Internet (udp) Connections
     Proto Local Address
     udp   *.echo
     udp   *.discard
     udp   *.daytime
     udp   *.chargen
     udp   *.time
```

To show the state of network interfaces, use the `snmpnetstat` command with the `-i` option. This option provides a statistics table that shows packets transferred, errors, and collisions as well as network addresses of the interface and the maximum transmission units (MTU).

```
# snmpnetstat -v 2c -c public -i testhost
```

The following table, including local and remote addresses, and protocols, is typically displayed:

```
Name Mtu  Network   Address  Ipkts     Ierrs  Opkts  Oerrs Queue
     eri0 1500 10.6.9/24 testhost 170548881 245601 687976 0     0
     lo0  8232 127      localhost 7530982   0      7530982 0    0
```

Note – The `Ipkts`, or incoming packets, value reported by the `snmpnetstat` command is *not* identical to that reported by the `netstat` command. The `snmpnetstat` command displays the total number of unicast, multicast and broadcast packets. The `netstat` command displays the total number of unicast and multicast packets, omitting broadcast packets.

Resource Usage

The resident size of the `snmpd` daemon depends on how the SMA is used.

The `snmpd` daemon dynamically allocates memory for certain MIB table data, for example, when you have defined printers, and then walk the Host Resource MIB. The resident size of the `snmpd` daemon can increase by up to 100Kbytes depending upon the number of printers you have defined.

JDMK Interoperability

Java™ Development Management Kit (JDMK) implements the JMX specifications and in addition, enables SNMP based instrumentation within the JDMK agent infrastructure. Like the System Management Agent, JDMK also supports the following standards:

- SNMPv3
- SNMPv2c
- SNMPv1

- USM
- Proxying

JDMK does not support AgentX.

Though both JDMK and the SMA address SNMP instrumentation, JDMK is very suited to Java based environments. The SMA is more suited to native C based implementations.

Configuration and Proxying With JDMK

In Sun systems where both JDMK and the SMA are present, the SMA by default resides on port 161. JDMK agents can publish their SNMP MIBs by being proxied from the SMA. Proxying can be set up using the proxy forwarding mechanism within the SMA. See Example 3–1.

Security must to be handled by the master agent, which is the SMA, and by the proxied JDMK agent. Security parameters that are contained in the proxy definition are forwarded to the proxied JDMK agent. If the request passes the SMA authentication and authorization and is forwarded to its proxy handler, then the dispatched request is proxied to the JDMK agent. The JDMK agent has its own local datastore that authorizes or rejects the message.

If several JDMK agents have the same MIB, SNMP contexts must be used in conjunction with proxying to differentiate between different instances of the same MIB. The context name can be based on the process ID. Alternatively, the context name can be based on the port on which the JDMK agent is running.

EXAMPLE 3–1 Adding a JDMK Proxy Statement

Incoming requests for the JDMK agent can be received by the System Management Agent and then proxied to the JDMK agent. Set JDMK proxy entries using the proxy token in the main `snmpd.conf` configuration file. Add a proxy statement, such as the following:

```
# proxy --Cn jdmkMib -v3 -a MD5 -u
 SecureUser -l authNopriv -A 12345678 localhost:10161  1.3.6.1.4.1.42.5000.2
```

In this example, a MIB is running on port 10161 and registers the MIB region 1.3.6.1.4.1.42.5000 under the context `jdmkMib`. The user, who is named `SecureUser`, must also exist in the JDMK USM. The `SecureUser` user must allow a security level of `auth` with HMACMD5 as the authentication algorithm. For more information on authentication algorithms, see "Authentication Protocol Algorithms" on page 43.

See the `snmpd.conf`(4) man page for more information. For more information about how proxying is set up, see "Proxy Handling for Solstice Enterprise Agents Requests" on page 67.

CHAPTER 4

Managing Security

This chapter provides background information, practical procedures, and examples related to security within the System Management Agent. With its SNMPv3 features, the System Management Agent provides an enhanced and configurable level of security for the management of users and network devices.

This chapter contains information on the following topics:

Security Overview

The System Management Agent supports SNMPv1, SNMPv2c and SNMPv3. The SNMPv1 and SNMPv2c authentication service is based on community strings defined on the management station. The SNMPv3 authentication service is based on users. Each request must contain either a community name or a user name depending upon the protocol being used.

The SNMPv3 authentication process implements the User-based Security Model (USM) to obtain a security name and security level from a user name. Similarly, both SNMPv1 and SNMPv2c determine the security level from the community string. The security name and security level are then used together with a context string, a group name and a view name to perform access control. Access control is done through the View-based Access Control Model, (VACM). This access control model is used after the authentication process. So, while USM is designed and used for authentication, VACM is designed and used for authorization.

For information about the supported versions of SNMP within the System Management Agent, see "SNMP Versions" on page 16.

Using USM for Authentication and Message Privacy

The User-based Security Model (USM) is used by the System Management Agent for authentication, encryption, and decryption of SNMPv3 packets. USM is used for the following reasons:

- Authentication of SNMP users
- Privacy of communication
- Integrity of messages
- Replay protection

The `snmpusm` utility is an SNMP application for basic maintenance of an SNMP agent's USM table. You must have write access to the `usmUserTable` MIB table. For more information, see the `snmpusm`(1M) man page.

Note – The `snmpusm` subcommands are not supported for v1 or v2c versions of SNMP. Without proxying, only SNMPv3 users can execute these commands.

The agent enables you to manage user entries through the main `snmpd.conf` configuration file and by use of the `snmpusm` command, through the USM MIB. The USM MIB enables the System Management Agent to find information about the user, including whether the user exists. Every request from a user is checked against the USM MIB. If the user exists, the USM MIB checks the following permissions:

- Is the user allowed authenticated requests?
- What type of auth encoding is allowed?

The USM MIB uses the local store key to compute a new digest that is based upon the authentication protocol specified by a particular user in the MIB. The computed digest is compared to the digest saved from the incoming packet. If the digests are the same, the user is authenticated. For more information on message digests, see "Authentication Protocol Algorithms" on page 43.

The following list describes the USM settings:

User	Identifies the user that is authorized to communicate with the SNMP engine.
Authorization Type	Specifies the authentication protocol algorithm to be used: MD5 and SHA. For more information, see "Authentication Protocol Algorithms" on page 43.
Authorization Password	Specifies the user's authentication password. Passwords must consist of at least eight characters.

Privacy Type: Specifies the privacy protocol to be used. For the System Management Agent, the privacy protocol is DES (Data Encryption Standard). For more information, see "Message Privacy" on page 44.

Privacy Password: Specifies the user's privacy password. Passwords must consist of at least eight characters.

Security Level: Indicates the security level of the user with regard to authentication and privacy:

- `noAuthNoPriv` Uses just the user name match for authentication.
- `authNoPriv` Provides authentication that is based on the MD5 or SHA1 algorithm.
- `authPriv` Provides privacy (encryption) based on the DES protocol.

Authentication uses a secret key to generate a MAC (Message Authentication Code) stored in `msgAuthenticationParameters`, which is part of `usmSecurityParameters`. Both the sending and receiving entities use the same secret key to produce the MAC. The message is therefore authenticated if both the sending and receiving MACs match.

Authentication Protocol Algorithms

In USM as implemented by the System Management Agent, two authentication protocols are supported. These authentication protocols are described in the following list.

HMAC-MD5–96 In the System Management Agent, the *Message Digest* implementation is HMAC-MD5–96. Based on MD5, this one-way encryption uses a 96–bit hash a 16 octet key length. Computationally, no two messages can have the same message digest. Also you cannot produce a message from a given prespecified target message digest. The MD5 algorithm is designed for digital signature applications. In these applications, large files must be securely compressed before being encrypted with a private key under a public-key cryptosystem. The HMAC-MD5–96 algorithm can be used with 32-bit machines. No large substitution tables are required. The algorithm can be coded quite compactly. For more information on MD5, see RFC 1321 at `http://www.ietf.org/rfc/rfc1321.txt`.

HMAC-SHA–96 In the System Management Agent, the Secure Hash Algorithm (SHA) implementation is HMAC-SHA–96. This one-way encryption uses a 96–bit hash and a 20–octet key length. The

algorithm takes as input a message of less than 2^{64} bits in length. The input message is processed in 512–bit blocks. The algorithm produces a 160-bit message digest output. This message digest can then, for example, be used as an input to a signature algorithm, which generates or verifies the signature for the message. The message digest is signed, instead of the message itself, which improves efficiency because the message digest is smaller than the original message. If the creator of a digital signature uses SHA, then the verifier of the digital signature clearly must also use SHA. If the message is changed during transit, this change almost always changes the message digest, so the digital signature fails to verify. SHA is secure because computationally no two messages can have the same message digest. You also cannot produce a message from a given prespecified target message digest. The design of SHA is similar to the MD5 family of hash functions. For more information on SHA, see RFC 3174 at `http://www.ietf.org/rfc/rfc3174.txt`.

For the System Management Agent, the default authentication protocol is HMAC-MD5–96. Setting is `auth proto = MD5`.

Message Privacy

USM supports privacy of messages. USM uses the CBC-DES Symmetric Encryption Protocol for encrypting and decrypting SNMPv3 packets. In the same way as with authentication, the same secret privacy key is used to encrypt and decrypt messages on both sides. Only the data portion is encrypted. The `auth` flag must be enabled and the security level must have privacy enabled to use encryption. Only the `scopedPDU` is encrypted. For more information, see "Where USM Security Information Is Contained" on page 45.

Currently, DES encryption is supported on the Solaris OS. DES encryption uses 56–bit key encryption. This level is the highest encryption level currently supported for DES, in this release of the Solaris software.

Public Keys

The System Management Agent supports Public Key Cryptography Standard (PKCS) #11. This token interface standard defines the interface that is used for communication between SSL and PKCS #11 modules. The version of PKCS on which the PKCS library compiled with the System Management Agent is based is the PKCS#11 v2.11 standard.

As described in "Authentication Protocol Algorithms" on page 43, the SHA1 algorithm is supported in addition to MD5. If the PKCS library is not found on your system, then the SMA attempts to use the standard Net-SNMP internal MD5, without DES support.

Where USM Security Information Is Contained

In an SNMPv3 packet string, USM information is contained in the following flags:

`msgFlags` — A single octet to indicate how the message is to be processed. For example, two bits of the `msgFlags` octet specify whether the packet has been encrypted and whether the packet has been authenticated. This flag is used to determine the security level of the message. Security levels, which are indicated in the main `snmpd.conf file`, are as follows:

- `noAuthNoPriv` — Represented by an integer: 1. Least access.
- `authNoPriv` — Represented by an integer: 2. More access than `noAuthNoPriv` but lower than `authPriv`.
- `authPriv` — Represented by an integer: 3. Most access, most secure.

`msgSecurityModel` — Specifies the security model used to generate the message, enabling the receiving entity to employ the appropriate model for security processing. In the System Management Agent, USM is the only supported security model.

`msgSecurityParameters` — An octet string containing data about the security model. This data is defined by the security model or models you are using. This data is used only by the security model or models you are using. The security model or models are specified in `msgSecurityModel`. USM uses this field to authenticate, encrypt, and decrypt SNMPv3 messages.

`scopedPDU` — Contains the normal Protocol Data Unit (`PDU`) and information for identifying the administratively unique context for processing the PDU. SNMPv2 and

SNMPv3 messages both use the same PDU format. This `scopedPDU` format is encrypted if privacy was enabled for the transaction.

The MIB definitions for USM can be found at `/etc/sma/snmp/mibs/SNMP-USER-BASED-SM-MIB.txt`.

For more information about USM, see RFC 3414 at `http://www.ietf.org/rfc/rfc3414.txt`.

Using VACM for Access Control

You can use the View-based Access Control Model (VACM) to find out whether access to a specified managed object is authorized. Access control is done at the following points:

- When processing retrieval request messages from the manager
- When processing modification request messages from the manager
- When notification messages must be sent to the manager

The VACM builds on the community string concept mentioned in "Community String" on page 18, by providing access control that you can easily administer in SMA.

Access control is defined by tokens in the main `snmpd.conf` configuration file. The SMA daemon `snmpd` recognizes the following tokens for VACM access security that you can use in the main `snmpd.conf` configuration file:

- `group`
- `access`
- `view`
- `com2sec`

The first three of these tokens are described in "Understanding VACM Tables" on page 48. The `com2sec` token takes NAME SOURCE and COMMUNITY options. You can use this token to give SNMPv3 security privileges to SNMPv1 and SNMPv2 users and communities. The `com2sec` token indicates the mapping from a source and community pair to a security name.

Faster and more usable wrappers are provided with the `snmpd.conf` file and are recognized by the SMA `snmpd` agent. These wrappers are defined using read write (rw) and read only (ro) syntax for users and communities, as follows:

- `rwuser`
- `rouser`
- `rwcommunity`

- `rocommunity`

The `rwuser` token entries specify the minimum allowed access that the user must specify:

`rwuser1 priv`	User must specify the privacy password.
`rwuser2 auth`	User can specify the privacy password if the user was created with a privacy password. Otherwise, the user must specify an authentication password.
`rwuser3 none`	User can specify either no password or an authentication password. Otherwise, the user can specify the privacy password if the user was created with a privacy password.

Where VACM Security Information Is Contained

VACM information is contained in several parameters in the SNMPv3 packet string. These parameters are passed to the `isAccessAllowed` mechanism. The `isAccessAllowed` mechanism is the single entry point in VACM for checking whether access should be granted.

VACM parameters are as follows:

`msgFlags`	A single octet that indicates how to process the message. For more information, see "Where USM Security Information Is Contained" on page 45.
`msgSecurityModel`	Indicates which security model was used at message generation, enabling the receiving entity to employ the appropriate model for security processing. You have a choice in SNMPv3 of using one security model or multiple security models.
`msgSecurityParameters`	An octet string containing data about the security model. The security model or models are determined in `msgSecurityModel`.
`scopedPDU`	Contains the PDU. Shows the administratively unique selector of management information for processing the PDU. In other words, the `scopedPDU` contains the context and managed object OIDs. The `scopedPDU` contains the following fields:

`contextEngineID`	Uniquely identifies an SNMP entity that can access an instance of a managed object within a context.

`contextName`	The name of the context to which the PDU data belongs. The `contextName` is unique.
`PDU`	The Protocol Data Unit (`PDU`) for SNMPv3 contains an operation for the data in the `contextName`. Identified by the combination of `contextEngineID` and the `contextName`.

For an explanation of the other fields of the SNMPv3 packet string, see "SNMP Versions" on page 16.

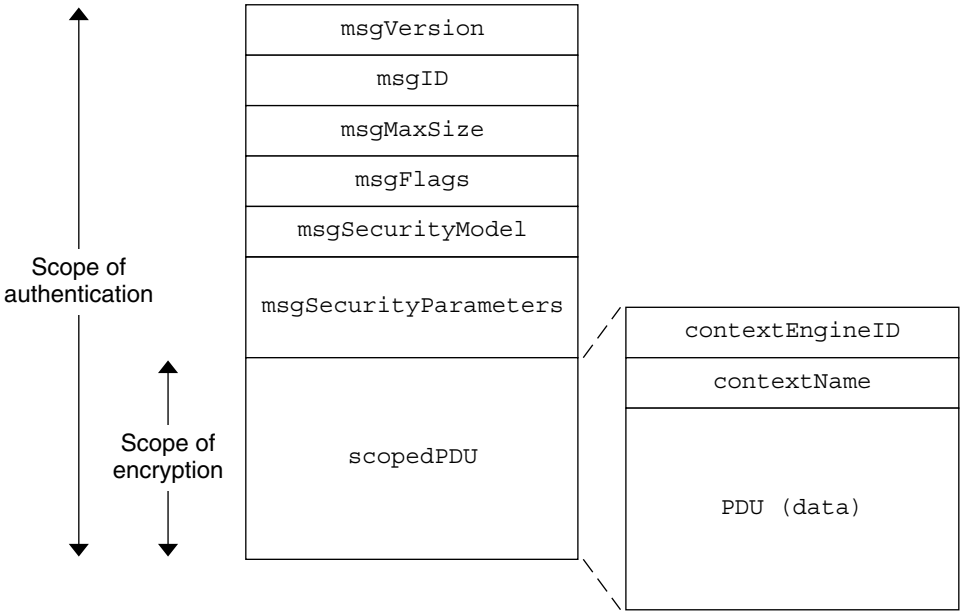

FIGURE 4–1 SNMPv3 Packet Format Showing Scopes of Authentication and Encryption

Understanding VACM Tables

In determining whether access should be granted to a message, VACM uses four tables:

- "Context Table" on page 49.

- "Security to Group Table" on page 50.
- "Access Table" on page 55.
- "View Tree Family Table" on page 52.

Each of these VACM tables handles a particular part of the access mechanism. Each table can be remotely configured using the VACM MIB. The VACM MIB is defined in RFC 3415 at `http://www.ietf.org/rfc/rfc3415.txt`.

The VACM determines whether a request that has been authenticated by the SMA's USM is authorized to access the MIB object that is contained in the request. The `snmpvacm` utility is an SNMP application for basic maintenance of an SNMP agent's VACM tables. You need to have write access to the `snmpvacm` MIB table to use the `snmpvacm` utility. For more information, see the `snmpvacm`(1M) man page.

This section describes each of the VACM tables, including how the tables are indexed and what each row contains.

Context Table

The `vacmContextTable` table stores the contexts that are available locally. A context is the selector of management information. A single managed object can be in several different contexts. For example, consider a single module designed to monitor the status of a printer. For a network with several printers, multiple instances of this module can be implemented, with each instance containing a unique printer name. The printer name in this case is the context.

A single SNMP entity can have access to several contexts.

The `vacmContextTable` table is indexed by a `contextName`. Each of its rows gives the context name in the form of a unique, readable string, *vacmcontextName*.

The System Management Agent looks in the `vacmContextTable` for the `contextName` found in the `scopedPDU`. For information on the `scopedPDU`, see "SNMP Versions" on page 16. If the System Management Agent does not find the `contextName` of a particular message in the `vacmContextTable`, access is denied. In this case, a return value of `noSuchContext` is returned.

If the `contextName` exists, access checking continues, as shown in Figure 4–2. An example of typical entries in a `vacmContextTable` is shown in Example 4–1.

EXAMPLE 4–1 Creating Typical Context Table Entries

Some typical `vacmContextTable` entries, created by a module, are:

```
SNMP-VIEW-BASED-ACM-MIB::vacmContextName."fileX" = STRING: fileX
SNMP-VIEW-BASED-ACM-MIB::vacmContextName."fileY" = STRING: fileY
```

The `contextNames` in this example are `fileX` and `fileY`.

Contexts are further explained in the *Solaris System Management Agent Developer's Guide*. To illustrate the concept of contexts, a demo module is supplied with the System Management Agent. This demo module shows the importance of contexts for implementing multiple instances of a module. For more information, see "Implementing Multiple Instances of a Module" in *Solaris System Management Agent Developer's Guide*.

Security to Group Table

The `vacmsecurityToGroupTable` table stores group information. A group name is given to a group of users and is used when managing their access rights. A group contains `SecurityModel` and a `SecurityName` value pairs. The resulting pair can only map to at most one group. The `vacmSecurityToGroupTable` table is indexed by the following:

- `securityModel`
- `securityName`

Each of the rows in the `vacmSecurityToGroupTable` table contains the following:

vacmSecurityModel	An SNMPv3 security model, in this case USM. For further information on USM, see "Using USM for Authentication and Message Privacy" on page 42. By using the `com2sec` token, SNMPv1 and SNMPv2c security models can be used. For more information about the `com2sec` token, see the `snmpd.conf`(4) man page.
vacmSecurityName	With USM, the *vacmSecurityName* is identical to *userName*. Represents a user in a format that is independent of the security model. By using the `com2sec` token, SNMPv1 and SNMPv2c security names can be used. For more information on the `com2sec` token, see the `snmpd.conf`(4) man page.
vacmGroupName	A readable string. Indicates the group that is associated with this entry.

The `SecurityName` is obtained by the `msgSecurityModel` specifier when a message is successfully authenticated and decrypted. The System Management Agent searches for this `msgSecurityModel` specifier and associated `SecurityName` in the `vacmSecurityToGroupTable` table. If the `msgSecurityModel` specifier and associated `SecurityName` are not found in the `vacmSecurityToGroupTable`, then access is denied. In this case, a return value of `noSuchGroupName` is returned.

If an entry is found, then the corresponding `groupName` is returned. Access checking continues, as shown in Figure 4–2.

Typical entries in a `vacmsecurityToGroupTable` are shown in Example 4–2.

EXAMPLE 4–2 Creating Typical Security to Group Table Entries

Create a group for two previously created users that are named `user2` and `user5`. In this example, the users are placed in a newly created group that is named `grpnam1`. Choose from one of two methods:

- Add the following lines to the main `/etc/sma/snmp/snmpd.conf` configuration file:

  ```
  group grpnam1 usm user2
  group grpnam1 usm user5
  ```

  If the group is created by adding to the main `/etc/sma/snmp/snmpd.conf` configuration file, then the entries that are created in the `vacmsecurityToGroupTable` table are as follows:

  ```
  SNMP-VIEW-BASED-ACM-MIB::vacmGroupName.3."user2" = STRING: grpnam1
  SNMP-VIEW-BASED-ACM-MIB::vacmGroupName.3."user5" = STRING: grpnam1
  SNMP-VIEW-BASED-ACM-MIB::vacmSecurityToGroupStorageType.3."user2" =
  INTEGER: permanent(4)
  SNMP-VIEW-BASED-ACM-MIB::vacmSecurityToGroupStorageType.3."user5" =
  INTEGER: permanent(4)
  SNMP-VIEW-BASED-ACM-MIB::vacmSecurityToGroupStatus.3."user2" = INTEGER:
  active(1)
  SNMP-VIEW-BASED-ACM-MIB::vacmSecurityToGroupStatus.3."user5" = INTEGER:
  active(1)
  ```

  Rebooting does not delete entries. To delete entries in this VACM table, use the `snmpvacm deleteGroup` command. This method works if the storage type is `nonVolatile`. For VACM table entries with other storage types, you must manually remove from the table entries from the main `/etc/sma/snmp/snmpd.conf` configuration file. If the group is created by editing the main `/etc/sma/snmp/snmpd.conf` configuration file, the `vacmsecurityToGroupTable` table entries can be deleted only by editing the main `/etc/sma/snmp/snmpd.conf` configuration file.

- Use the `snmpvacm` command. For `user2`, a group can be created using the `snmpvacm` command as follows:

  ```
  # snmpvacm -v3 -u myuser -a MD5
  -A my_password -l authNoPriv localhost
  createSec2Group 3 user2 grpnam1
  ```

  For `user5`, a group can be created using the `snmpvacm` command as follows:

  ```
  # snmpvacm -v3 -u myuser -a MD5
  -A my_password -l authNoPriv localhost
  createSec2Group 3 user5 grpnam1
  ```

  The user `myuser` has *rwuser* level access. Therefore, group entries are created in this example as the `myuser` user where appropriate for the context. The users `user2` and `user5` do not have rights to update VACM tables.

View Tree Family Table

The `vacmViewTreeFamilyTable` table stores all collections of view subtree families. These collections are called MIB views. A MIB view is an OID subtree value, which is the family name, together with a bitstring value, which is the family mask. The family mask identifies which sub-identifiers of the family name are in the MIB view. A mask is a list of hex octets, which separated by either "." or ":" The default is "ff".

In a MIB view, each view subtree family has a type. The type determines if the view subtree family is included in the MIB view. A managed object instance is contained within a MIB view only if both of these statements are true:

- The managed object's OID contains the same number of sub-identifiers as the OID subtree, or more.
- If the corresponding bit of the mask is not zero, each of the managed object's OID sub-identifiers match corresponding sub-identifiers in the OID subtree.

If the configured value of the mask is too short to check these statements, the value is implicitly extended by a series of ones. A view family subtree with a mask of zero bits therefore corresponds to a mask of all ones, which in turn corresponds to one MIB subtree.

The `vacmViewTreeFamilyTable` table is indexed by:

`viewName`	Specified by the access right selected in the `vacmAccessTable` table. Used for access checking.
An OID of a MIB subtree	The OID of the PDU is compared to the MIB view.

Each of the rows in the `vacmViewTreeFamilyTable` table contains:

vacmViewTreeFamilyViewName	The name of the MIB view.
vacmViewTreeFamilySubtree	OID subtree. The OID subtree couples with a mask to make MIB view subtrees.
vacmViewTreeFamilyMask	Bitstring mask. The bitstring mask couples with an OID subtree to make MIB view subtrees.
vacmViewTreeFamilyType	The type determines whether the view subtree family is included in the MIB view.

If the MIB view does not contain the OID searched for, access is denied. In this case, a return value of `notInView` is returned. Otherwise, where the MIB view does contain the correct OID, access is granted. In this case a value of `accessAllowed` is returned.

The overall flow chart for this VACM algorithm is illustrated by Figure 4–2. In this diagram, the guideline terms suggested by the RFC are given for clarity:

`securityName` and `securityModel`	Indicate *who* requires access.
`contextName`	Determines *where* access is granted.

`securityLevel` and `securityModel`	Determines *how* access is granted.
`viewType`	Can be read, write or notify. Determines *why* a particular level of access is required by a group or user.
Object type, or OID of the managed object	Indicates *what* type of management data is checked.
Instance of the managed object	Combines with the object type to deliver *which* particular instance is checked to be in the MIB view. This decision is yes/no.

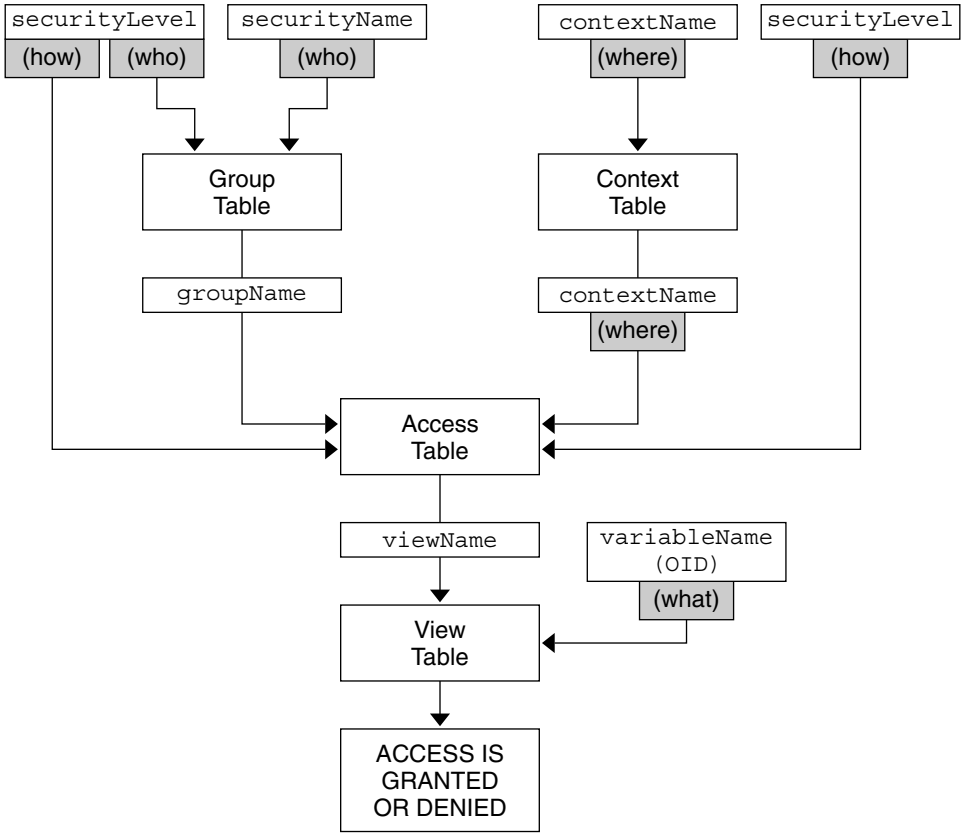

FIGURE 4–2 Overall Flow Chart for VACM

Example 4–3 shows typical entries in a `vacmViewTreeFamilyTable`.

EXAMPLE 4–3 Creating Typical View Tree Family Table Entries

You can create a view in two ways:

- You can create a view by adding the view to the main `/etc/sma/snmp/` configuration file.

  ```
  view all included .1 FF
  view none excluded .1 FF
  view vwnam1 included .1.3.6.1 FF
  ```

  The "FF" here is the mask.

  If the view is created by adding the group to the main `/etc/sma/snmp/snmpd.conf` configuration file, then the entries that are created in the `vacmViewTreeFamilyTable` table are as follows:

  ```
  SNMP-VIEW-BASED-ACM-MIB::vacmViewTreeFamilyMask."all".1.1 = STRING: "ÿ"
  SNMP-VIEW-BASED-ACM-MIB::vacmViewTreeFamilyMask."none".1.1 = STRING: "ÿ"
  SNMP-VIEW-BASED-ACM-MIB::vacmViewTreeFamilyMask."vwnam1".4.1.3.6.1
  = STRING: "ÿ"
  SNMP-VIEW-BASED-ACM-MIB::vacmViewTreeFamilyType."all".1.1
  = INTEGER: included(1)
  SNMP-VIEW-BASED-ACM-MIB::vacmViewTreeFamilyType."none".1.1
   = INTEGER: excluded(2)
  SNMP-VIEW-BASED-ACM-MIB::vacmViewTreeFamilyType."vwnam1".4.1.3.6.1
   = INTEGER: included(1)
  SNMP-VIEW-BASED-ACM-MIB::vacmViewTreeFamilyStorageType."all".1.1
   = INTEGER: permanent(4)
  SNMP-VIEW-BASED-ACM-MIB::vacmViewTreeFamilyStorageType."none".1.1
   = INTEGER: permanent(4)
  SNMP-VIEW-BASED-ACM-MIB::vacmViewTreeFamilyStorageType."vwnam1".4.1.3.6.1
   = INTEGER: permanent(4)
  SNMP-VIEW-BASED-ACM-MIB::vacmViewTreeFamilyStatus."all".1.1
  = INTEGER: active(1)
  SNMP-VIEW-BASED-ACM-MIB::vacmViewTreeFamilyStatus."none".1.1
   = INTEGER: active(1)
  SNMP-VIEW-BASED-ACM-MIB::vacmViewTreeFamilyStatus."vwnam1".4.1.3.6.1
   = INTEGER: active(1)
  ```

- You can create a view by using the `snmpvacm` command:

  ```
  # snmpvacm -v3 -u myuser -a MD5
  -A my_password -l authNoPriv -Ce localhost
  createView all .1 FF

  # snmpvacm -v3 -u myuser -a MD5
  -A my_password -l authNoPriv localhost
  createView none .1 FF

  # snmpvacm -v3 -u myuser -a MD5
  -A my_password -l authNoPriv localhost
  createView vwnam1 .1.3.6.1 FF
  ```

EXAMPLE 4–3 Creating Typical View Tree Family Table Entries *(Continued)*

Here, the user `myuser` has *rwuser* level access. Therefore, view entries are created in this example for `myuser`, where appropriate for the context.

If the view is created by using the `snmpvacm` command, the storage type would be `nonVolatile`.

Access Table

The `vacmAccessTable` table stores each group's access rights. Each group can have multiple access rights. The most secure access right is chosen. The `vacmAccessTable` table is indexed by:

`groupName`	Returned from the lookup into the `vacmSecurityToGroupTable` table.
`contextPrefix`	The valid `contextName` matched within the `vacmContextTable` table.
`securityModel`	Specified in the message's `msgSecurityModel` parameter.
`securityLevel`	Specified in the message's `msgFlags` parameter.

Each of the rows in the `vacmAccessTable` table contains:

vacmGroupName	The group's name. This group has one or multiple access rights.
vacmAccessContextPrefix	The `contextName` must match the value of *vacmAccessContextPrefix*. See *vacmAccessContextMatch*.
vacmAccessSecurityModel	Indicates the security model that must be used to get access rights.
vacmAccessContextMatch	If *vacmAccessContextMatch* is set to *exact*, then the `contextName` must exactly match the value of the *vacmAccessContextPrefix* object. If *vacmAccessContextMatch* is set to *prefix*, the `contextName` can match the first several characters of the *vacmAccessContextPrefix* object. This `contextName` is the name already matched within the `vacmContextTable` table.
vacmAccessSecurityLevel	Indicates the lowest security level necessary for having access to this access right. For information about security levels, see "Where VACM Security Information Is Contained" on page 47.

vacmAccessReadViewName	Authorized MIB `viewName` for read access. If *vacmAccessReadViewName* is empty, no active view exists for read access.
vacmAccessWriteViewName	Authorized MIB `viewName` for write access. If *vacmAccessWriteViewName* is empty, no active view exists for write access.
vacmAccessNotifyViewName	Authorized MIB `viewName` for notify access. If *vacmAccessWriteViewName* is empty, no active view exists for notify access.

If an access right is not found, access is denied. In this case, a return value of `noAccessEntry` is returned.

When an access right is selected, then the `viewName` indicated by that access right is selected. This `viewName` is determined by the PDU. If the SNMP operation in the PDU is a GETNEXT or GET operation, the *vacmAccessReadViewName* string is used. If the SNMP operation in the PDU is a TRAP operation, the *vacmAccessNotifyViewName* string is used. If the `viewName` is not configured, access is denied. In this case, a return value of `noSuchView` is returned.

If the access right is selected with a correctly configured `viewName`, access checking continues, as shown in Figure 4–2. Example 4–4 shows typical access table entries.

An additional example, Example 4–5, shows how to check that the users set up in this example and previous examples exist and are registered in VACM tables.

EXAMPLE 4–4 Creating Typical Access Table Entries

You can create access table entries in two ways:

- You can create an access table entry by adding the entry to the main `/etc/sma/snmp/snmpd.conf` configuration file:

  ```
  access grpnam1 fileX usm priv exact all none none
  access grpnam1 "" usm auth exact all vwnam1 none
  ```

  If the group is created by adding to the main `/etc/sma/snmp/snmpd.conf` configuration file, then the entries that are created in the `vacmAccessTable` table are as follows:

  ```
  SNMP-VIEW-BASED-ACM-MIB::vacmAccessContextMatch.
  "grpnam1"."".3.authNoPriv = INTEGER: exact(1)
  SNMP-VIEW-BASED-ACM-MIB::vacmAccessContextMatch.
  "grpnam1"."fileX".3.authPriv = INTEGER: exact(1)
  SNMP-VIEW-BASED-ACM-MIB::vacmAccessReadViewName.
  "grpnam1"."".3.authNoPriv = STRING: all
  SNMP-VIEW-BASED-ACM-MIB::vacmAccessReadViewName.
  "grpnam1"."fileX".3.authPriv = STRING: all
  SNMP-VIEW-BASED-ACM-MIB::vacmAccessWriteViewName.
  "grpnam1"."".3.authNoPriv = STRING: vwnam1
  ```

EXAMPLE 4–4 Creating Typical Access Table Entries *(Continued)*

```
SNMP-VIEW-BASED-ACM-MIB::vacmAccessWriteViewName.
"grpnam1"."fileX".3.authPriv = STRING: none
SNMP-VIEW-BASED-ACM-MIB::vacmAccessNotifyViewName.
"grpnam1"."".3.authNoPriv = STRING: none
SNMP-VIEW-BASED-ACM-MIB::vacmAccessNotifyViewName.
"grpnam1"."fileX".3.authPriv = STRING: none
SNMP-VIEW-BASED-ACM-MIB::vacmAccessStorageType.
"grpnam1"."".3.authNoPriv = INTEGER: permanent(4)
SNMP-VIEW-BASED-ACM-MIB::vacmAccessStorageType.
"grpnam1"."fileX".3.authPriv = INTEGER: permanent(4)
SNMP-VIEW-BASED-ACM-MIB::vacmAccessStatus.
"grpnam1"."".3.authNoPriv = INTEGER: active(1)
SNMP-VIEW-BASED-ACM-MIB::vacmAccessStatus.
"grpnam1"."fileX".3.authPriv = INTEGER: active(1)
```

If the group was created by directly editing the main `/etc/sma/snmp/snmpd.conf` configuration file, the group's storage type would be `permanent`.

- Alternatively, you can create an access table entry by using the `snmpvacm` command:

```
# snmpvacm -v3 -u myuser -a MD5
-A my_password -l authNoPriv localhost
createAccess grpnam1 "fileX" 3 3 1 all none none
```

Here, the user `myuser` has *rwuser* level access. Therefore, access entries are created in this example as this `myuser` user where appropriate for the context.

```
# snmpvacm -v3 -u myuser -a MD5
-A my_password -l authNoPriv localhost
createAccess grpnam1 "" 3 2 1 all vwnam1 none
```

If the group is created by use of the `snmpvacm` command, then the storage type would be `nonvolatile`. Objects created by use of either the `snmpvacm` or `snmpusm` commands have storage type of `nonvolatile`.

EXAMPLE 4–5 Checking That Users Exist in the VACM Tables

Using the information in Example 4–3 and Example 4–4, check that the SNMPv3 user, `user2`, created in Example 4–2, exists. Validate the access entries already created for `user2`, by checking and setting values for the user. Use the `snmpget` and `snmpset` commands, once with encryption and once with no encryption. This method illustrates that the access entry for `user2` is the minimum security level required, defined as `auth`= 2. The method also illustrates that `priv` can be used as well, since that level is more secure.

Use the `snmpget` command to check that the new user exists, with the `DES` option set for encryption. A context, `-n fileX`, is specified:

EXAMPLE 4–5 Checking That Users Exist in the VACM Tables *(Continued)*

```
# snmpget -v3 -u user2 -a MD5
-A my_password -l authPriv -x DES -X my_password
-n fileX localhost 1.3.6.1.4.1.42.2.2.4.4.6.1.1.0
```

This command validates one of the access entries that you set up for `user2`. The options that are associated with use of the `snmpget` command are described in the `snmpcmd`(1M) man page.

The `snmpget` command retrieves the following information:

```
SNMPv2-SMI::enterprises.42.2.2.4.4.6.1.1.0 = INTEGER: 111
```

In this returned output, 111 is an integer that is stored in the specified OID.

Similarly, `snmpget` command can be used to check that the new user exists, without the `DES` option set. If you do not set the `DES` option, no encryption is requested. This example shows that the user, `user2`, can execute an operation not in a context:

```
# snmpget -v3 -u user2 -a MD5
-A my_password -l authNoPriv localhost 1.3.6.1.2.1.1.3.0
```

The `snmpget` command retrieves the following information about system uptime:

```
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (5375) 0:00:53.75
```

Try to set a new value for `sysLocation`:

```
# snmpset -v3 -u user2 -a MD5
-A my_password -l authPriv -x DES -X my_password
localhost 1.3.6.1.2.1.1.6.0 s "new val"
```

In this command, the `s` indicates “string”. The OID is `sysLocation`. The value being added to the `sysLocation` is `new val`.

Note that `user2` has full access rights to the context (`authPriv`) for DES. The password is *my_password*. The following is returned:

```
SNMPv2-MIB::sysLocation.0 = STRING: new val
```

Confirm these settings with the `snmpget` command:

```
# snmpget -v3 -u user2 -a MD5
-A my_password -l authPriv -x DES -X my_password localhost 1.3.6.1.2.1.1.6.0

SNMPv2-MIB::sysLocation.0 = STRING: new val
```

Try the same command without the DES encryption set:

```
# snmpset -v3 -u user2 -a MD5
-A my_password -l authNoPriv localhost 1.3.6.1.2.1.1.6.0 s "new val2"
```

The same result is successfully returned:

EXAMPLE 4–5 Checking That Users Exist in the VACM Tables *(Continued)*

```
SNMPv2-MIB::sysLocation.0 = STRING: new val2

# snmpget -v3 -u user2 -a MD5
-A my_password -l authNoPriv localhost 1.3.6.1.2.1.1.6.0

SNMPv2-MIB::sysLocation.0 = STRING: new val2
```

This output shows that the user has write access to MIB-II.

If `user2` has been defined in the `snmptrapd.conf` file, then start the SNMP trap daemon using the `snmptrapd` command:

```
# /usr/sfw/sbin/snmptrapd
```

Also, use the `snmpinform` command to send an INFORM-PDU trap. The `snmpinform` command validates that the user, `user2` or `user2`, can generate notifications. Notifications can be generated if you perform a cold start. A cold start generates a notification, or a "trap." The user can see this trap in the `/var/log/snmpd.log` file.

```
# /usr/sfw/sbin/snmpinform  -v3 -u user2 -a MD5
 -A my_password -l authNoPriv localhost 42 coldStart.0
```

For more information, see the `snmptrapd.conf`(4) and `snmpinform`(1M) man pages.

Troubleshooting Problems With VACM Tables

When creating VACM table entries, ensure that you configure access rights correctly for your users and user groups. Incorrectly configured access rights can lead to access being denied to key users.

Avoid creating large numbers of groups of users. Large numbers of groups can be difficult to administrate. Troubleshooting problems when you have created very large numbers of different user groups can become unmanageable.

When working with VACM tables, return values can include the following messages:

`noSuchContext` — This value is returned if the System Management Agent does not find the `contextName` of a particular message in the `vacmContextTable`. Access is denied. Check the context table entries. Ensure that these entries are correctly configured. Has each user got a context? For more information, see "Context Table" on page 49.

`noSuchGroupName` — Thisvalue is returned if the `msgSecurityModel` specifier and associated `SecurityName` are not found in the `vacmSecurityToGroupTable`. Access is denied. Check the security to group table entries. Ensure that these entries are

	correctly configured. Has each user got a group name? Have users been entered into the table correctly? For more information, see "Security to Group Table" on page 50.
`notInView`	This value is returned if the MIB view does not contain the OID searched for. Access is denied. For more information, see "View Tree Family Table" on page 52.
`noAccessEntry`	This value is returned if an access right is not found. Access is denied. Have you correctly set up the mask? Although each group can have multiple access rights, only the most secure access right is selected. Is the *vacmAccessContextMatch* parameter set to *exact*? If the *vacmAccessContextMatch* parameter is set to exact, the `contextName` must be an exact match. Try setting the *vacmAccessContextMatch* value to *prefix* if appropriate. For more information, see "Access Table" on page 55.

Note – Badly configured VACM tables can subject the network to unauthorized, possibly malicious access. Ensure that you check your VACM table configurations in a test environment before implementing these configurations on your network devices.

For more information on VACM, see RFC 3415 at `http://www.ietf.org/rfc/rfc3415.txt`.

The MIB definitions for VACM can be found at `/etc/sma/snmp/mibs/SNMP-VIEW-BASED-ACM-MIB.txt`.

Creating and Managing Users

This section provides procedures that explain how to create users with security. Several methods are available to create users in the System Management Agent. After you first install the System Management Agent, the default configuration is for new users to be SNMPv1 and SNMPv2c users.

Note – The agent is not configured to create SNMPv3 users by default. To create SNMPv3 users in the System Management Agent, first you need to edit the main `/etc/sma/snmp/snmpd.conf` file. For more information, see the `snmpd.conf`(4) man page.

The first procedure in this section, "To Create a New SNMPv3 User" on page 61, shows you how to create the first, initial new user. Additional users are cloned from this initial user, so that the initial user's authentication and security types can be inherited. These types can be changed later. In cloning, secret key data for the user is set. You must know the passwords for the initial user and later users that you set up. You can only clone one user at a time from the initial user that you set up.

▼ To Create a New SNMPv3 User

The `net-snmp-config` command used in this procedure adds a line to the `/etc/sma/snmp/snmpd.conf` file, giving the initial user read and write access to the agent.

1. **Stop the System Management Agent.**

   ```
   # /etc/init.d/init.sma stop
   ```

2. **To create the new user, use the `net-snmp-config` command.**

   ```
   # /usr/sfw/bin/net-snmp-config --create-snmpv3-user
   -a "my_password" newuser
   ```

   This command causes a new user to be created, named `newuser`, with a password equal to `my_password`. The new user creation uses both MD5 and DES, which are described in "Authentication Protocol Algorithms" on page 43.

   By default, when creating a user using the `net-snmp-config` command, these settings are created unless otherwise specified:

   ```
   auth protocol = MD5security level = rwuser auth
   ```

3. **Check whether this command has been successful.**

   Start the System Management Agent:

   ```
   # /etc/init.d/init.sma start
   ```

   Check whether the new user exists.

   ```
   # snmpget -v 3 -u newuser -l authNoPriv
   -a MD5 -A my_password localhost sysUpTime.0
   ```

Note – Passwords must contain at least eight characters.

Giving the new user read and write access is not always useful. If you want to reduce or change the access rights of the new user, edit the `/etc/sma/snmp/snmpd.conf` file. For more information, see the `snmpd.conf`(4) man page.

▼ To Create a New User Using System Prompts

1. **Stop the System Management Agent.**

   ```
   # /etc/init.d/init.sma stop
   ```

2. **To create the new user, named `newuser`, with a password equal to `my_password`, use the `net-snmp-config` command interactively.**

   ```
   # ./net-snmp-config --create-snmpv3-user

   Enter a SNMPv3 user name to create:
   ```

3. **Provide the appropriate user name, in this case:**

   ```
   newuser

   Enter authentication pass-phrase:
   ```

4. **Type the appropriate pass-phrase, in this case:**

   ```
   my_password

   Enter encryption pass-phrase:
   ```

5. **To reuse the authentication pass-phrase, press Return.**

   ```
   adding the following line to /var/sma_snmp/snmpd.conf:
   createUser newuser MD5 "newuser_pass" DES
   adding the following line to /etc/sma/snmp/snmpd.conf:
   rwuser newuser
   ```

   By default, when creating a user using the `net-snmp-config` command, these settings are created unless otherwise specified:

   `auth protocol = MD5`

   `security level = rwuser auth`

6. **Check whether the new user exists.**

   ```
   # snmpget -v 3 -u newuser -l authNoPriv
   -a MD5 -A my_password localhost sysUpTime.0
   ```

Note – Passwords must contain at least eight characters.

Giving the new user read and write access is not always useful. If you want to reduce or change the access rights of the new user, edit the `/etc/sma/snmp/snmpd.conf` file. For more information, see the `snmpd.conf`(4) man page.

▼ To Create Additional SNMPv3 Users With Security

The preferred method of creating a new user in secure SNMP is to clone the initial user that you originally set up. This procedure copies the user you set up in "To Create a New SNMPv3 User" on page 61. This procedure uses the `snmpusm` command described in "Using USM for Authentication and Message Privacy" on page 42. For more information, see the `snmpusm`(1M) man page.

1. **Check whether the System Management Agent is running.**

   ```
   # /etc/init.d/init.sma status
   ```

   If the agent is not running, start it.

   ```
   # /etc/init.d/init.sma start
   ```

2. **Create a new user using the `snmpusm` command.**

   ```
   # snmpusm -v 3 -u newuser -a MD5 -A my_password -l
   authNoPriv localhost create lee newuser
   ```

   This command creates a user named "`lee`". This new user has the same password `my_password`, as the source user, named "`newuser`", that you created in "To Create a New SNMPv3 User" on page 61.

3. **Change the new user's password.**

   ```
   # snmpusm -v 3 -u lee -a MD5 -A my_password -l
   authNoPriv localhost passwd my_password lee_password
   ```

   This command gives the user `lee` a new password, `lee_password`. The default `auth type` is MD5.

4. **Create associated VACM entries either by directly editing the `/etc/sma/snmp/snmpd.conf` file or by using the `snmpvacm` command.**

   If you are directly editing the `snmpd.conf` file you must first stop the agent.

   ```
   # /etc/init.d/init.sma stop
   ```

5. **Assign access to `lee`.**

   - To give `lee` read and write access, add a new `rwuser` line to the `snmpd.conf` file.

     ```
     rwuser lee
     ```

   - To give `lee` read-only access, add a new `rouser` line to the `snmpd.conf` file.

     ```
     rouser lee
     ```

   If you do not specify a security level, the System Management Agent defaults to `authNoPriv`. For more information, see the `snmpd.conf`(4) or `snmpvacm`(1M) man pages.

6. **Check whether this procedure has been successful.**

Restart the System Management Agent.

```
# /etc/init.d/init.sma restart
```

Check whether your new user exists.

```
# snmget -v 3 -u lee -a MD5 -A lee_password -l
 authNoPriv localhost sysUpTime.0
```

Managing SNMPv1 and SNMPv2c Users With SNMPv3 Security

For SNMPv1 and SNMPv2c users, community string is used for security. The standard Net-SNMP token, `com2sec`, is provided with the SMA. The `com2sec` token enables you to map a host name and community string pair, for SNMPv1 or SNMPv2c, to a security name. In this case, the security level is `noAuthNoPriv`. For information on the `noAuthNoPriv` security level and on other security levels, see "Where USM Security Information Is Contained" on page 45.

Proxy Statements and Security

In the System Management Agent, proxying is supported for SNMPv1 and SNMPv2c users only. For more information, see "Proxy Handling for Solstice Enterprise Agents Requests" on page 67.

Creating and Managing Groups

Creating a large number of groups in SNMP causes management and administration of these groups to become very complex. If you create a large number of groups, troubleshooting these groups very difficult.

Note – When groups or views are created by editing the `snmpd.conf` file, the storage type is permanent. If you edit the `snmpd.conf` file instead of using the `snmpvacm` command, entries for groups are permanent. You can delete the entries only by removing them from the `snmpd.conf` file.

Follow the examples provided in "Using VACM for Access Control" on page 46 for creating and managing groups.

CHAPTER 5

Migrating From Other Agents

This chapter provides information and procedures on how to migrate handling of processes and tasks from other management agents to the System Management Agent. Migration of applications is explained in the *Solaris System Management Agent Developer's Guide*. Migrating your applications to the SMA from any other agents you are using within the Solaris OS is not urgent. The exception to this rule is that of the Solstice Enterprise Agents software.

This chapter contains information on these topics:

Migration From Solstice Enterprise Agents Software

Support for the Solstice Enterprise Agents software is to be discontinued in a future Solaris release. The Solstice Enterprise Agents software master agent is `snmpdx`, located at `/usr/lib/snmp/`. Its functions are to be replaced by the System Management Agent master agent, `snmpd`. This agent is located in `/usr/sfw/sbin/`. For this reason, any Solstice Enterprise Agents subagents that developers have created must at some point be migrated to use the System Management Agent.

You can run the Solstice Enterprise Agents software and associated subagents concurrently with the SMA provided that SMA has been configured to load the `seaProxy` module. This purpose of this module is explained in "Proxy Handling for Solstice Enterprise Agents Requests" on page 67.

The Solstice Enterprise Agents software includes a subagent, `mibiisa`, that implements MIB-II and the `sun.mib`. In the System Management Agent, the functionality of `mibiisa` is implemented by the MIB-II portion of the System Management Agent.

Note – In this Solaris release, the Solstice Enterprise Agents `mibiisa` subagent is disabled. All SNMP requests that are intended for `mibiisa` are handled by the MIB-II implementation in the System Management Agent.

▼ To Prevent The System Management Agent Initializing at Boot Time

During the boot of the Solaris software, the SMA starts by default *after* the `snmpdx` has started. If you do not want to migrate your agent to the SMA, stop the SMA if it is running and edit the startup scripts. This editing prevents the System Management Agent from automatically starting at reboot.

1. **Open the `snmpd.conf` file.**

   This file is located at `/etc/sma/snmp/snmpd.conf`

2. **Edit the `snmpd.conf` file.**

   Instructions are included within the file for your convenience:

```
#####################################
# SECTION: Admins who want to disable the snmpd daemon from
# starting at boot time.
# Change DISABLE=NO to DISABLE=YES
# DO NOT DELETE
# DO NOT UNCOMMENT
# DISABLE=NO
#
# end ADMIN
```

   Change the NO for the DISABLE flag to YES, to prevent the `snmpd` daemon from starting at boot time.

```
#####################################
# SECTION: Admins who want to disable the snmpd daemon from
# starting at boot time.
# Change DISABLE=NO to DISABLE=YES
# DO NOT DELETE
# DO NOT UNCOMMENT
# DISABLE=YES
#
# end ADMIN
```

Proxy Handling for Solstice Enterprise Agents Requests

You do not have to migrate those SNMP subagents developed using the Solstice Enterprise Agents software to the SMA if the SMA has been configured to load the `seaProxy` module. The `seaProxy` module allows the Solstice Enterprise Agents software and associated subagents to run concurrently with the SMA.

Note – The System Management Agent has been specifically customized to allow you to proxy incoming requests from the SMA to the Solstice Enterprise Agents software. This customization is a point of difference between the System Management Agent and the standard Net-SNMP implementation, version 5.0.9, on which the SMA is based.

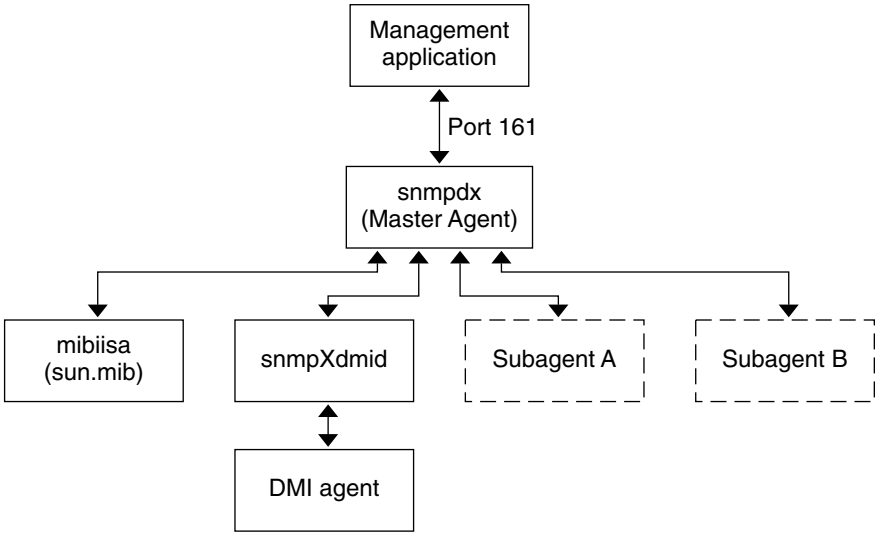

FIGURE 5–1 Routing of Requests and Responses in the SEA Software

After the System Management Agent has been installed, requests that would have originally been handled solely by the Solstice Enterprise Agents software are handled differently.

- Requests for MIB-II that would previously have gone directly to `snmpdx` are now handled by `snmpd`, the System Management Agent master agent.
- If the `seaExtensions` module has been loaded, requests for `sun.mib`, the `mibiisa` Solstice Enterprise Agents subagent MIB, are handled by this `seaExtensions` module. The MIB that describes this module is the SUN-SEA-EXTENSIONS-MIB.

- If the `seaProxy` module has been loaded, requests for `snmpdx.mib`, the Solstice Enterprise Agents software master agent MIB, are handled by this `seaProxy` module. The MIB that describes this module is the SUN-SEA-PROXY-MIB. For more information on the `seaProxy` module, see "Enabling the `seaProxy` Module" on page 68.

These statements are not true if you have prevented the System Management Agent from initializing at boot time. For more information, see "To Prevent The System Management Agent Initializing at Boot Time" on page 66.

A `proxy` token is available to specify that any incoming requests under a particular OID are proxied to another host. See the `snmpd.conf`(4) man page for a description of this proxy statement.

Enabling the `seaProxy` Module

On arrival at port 161, incoming requests that are intended for the Solstice Enterprise Agents software are received by the SMA. If a proxy exists for the request, the request is passed to the `snmpdx` daemon. From the `snmpdx` daemon, the request is passed to the Solstice Enterprise Agents software subagents. The `seaProxy` module generates *dynamic proxies*, which are not found in `snmpd.conf`. Dynamic proxies are based on static and dynamic Solstice Enterprise Agents subagent registrations. The `seaProxy` module uses Solstice Enterprise Agents subagent registration details to generate dynamic proxies.

To enable the `seaProxy` module provided with the System Management Agent, verify that in the `/etc/sma/snmp/snmpd.conf` file, for systems running on x86 platforms, the following line is configured:

```
dlmod seaProxy /usr/sfw/lib/libseaProxy.so
```

To enable the `seaProxy` module provided with the System Management Agent, verify that in the `/etc/sma/snmp/snmpd.conf` file, for systems running on SPARC platforms, the following line is configured:

```
dlmod seaProxy /usr/sfw/lib/sparcv9/libseaProxy.so
```

When the `seaProxy` module loads, the `seaProxy` module immediately begins collecting information from the Solstice Enterprise Agents subagent. For this reason, among others, the `snmpd` daemon must start up after the `snmpdx` daemon. If the `snmpd` daemon starts up before `snmpdx` daemon, the SMA re-reads the Solstice Enterprise Agents software subagent registration table. The `snmpdx` daemon can be running before the `snmpd` daemon if, for example, you stop and restart the `snmpd` daemon.

The `seaProxy` module uses the information in the software subagent registration table to generate proxies for those Solstice Enterprise Agents software subagents that have already registered.

The `seaProxy` module does not generate proxies for the `mibiisa` subagent.

Proxy Statements for Incoming Requests

This section describes proxy statements for requests from the System Management Agent that are intended for the Solstice Enterprise Agents software.

When dynamic proxies have been generated, the System Management Agent proxy mechanism handles the forwarding of those requests to `snmpdx`. The `seaProxy` module generates dynamic proxies for any Solstice Enterprise Agents subagents that have to register with `snmpdx`. Therefore, Solstice Enterprise Agents subagents can still be used with the SMA. Note that support of the Solstice Enterprise Agents software, including `snmpdx`, is for a limited transitional time. Migrate as early as possible those subagents that you implemented with the Solstice Enterprise Agents, to use the System Management Agent.

Migration from Solstice Enterprise Agents software to the System Management Agent is done through the AgentX subagent. If you have Solstice Enterprise Agents modules that you specifically want to migrate to the System Management Agent, see the *Solaris System Management Agent Developer's Guide*. This contains information on migrating modules, and explains the demo modules that are shipped with the System Management Agent. One of these demo modules is specifically designed to illustrate the migration process for Solstice Enterprise Agents modules.

If both the System Management Agent and the Solstice Enterprise Agents software are running, `snmpd`, the SMA master agent, should occupy port 161. During the boot process, the SMA startup script, `init.sma`, obtains an anonymous port. The `init.sma` script configures `snmpdx` to run on this port through the port entry in the Solstice Enterprise Agents configuration file, `snmpd.conf`, at `/etc/snmp/conf/`. After the change, the last few lines of the `/etc/snmp/conf/snmpdx.reg` file contain the new port number.

In this example, the new port number is 16161. The last few lines of the `/etc/snmp/conf/snmpdx.reg` file also contain other details:

```
agents =
{
  {  name = "relay-agent"
     subtrees = { sun.2.15 }
     timeout = 900000000
     port = 16161
  }
}
```

When Solstice Enterprise Agents subagents such as the DMI subagent start, they send requests to port 161 with a "private" community string. This "private" community string must be defined in the System Management Agent configuration file that was read at startup. Otherwise, Solstice Enterprise Agents subagents do not register successfully and die.

The SMA checks that a proxy statement is generated for the OID of the incoming request. The SMA performs this check if the "private" community string that the Solstice Enterprise Agents subagents hold in their requests is defined in the SMA

configuration file that was read at startup. Once these strings are verified, the SMA changes the port of the incoming request to the port configured as described in this section. In this example, the port that is configured is port 16161.

Note – After the `seaProxy` module has been enabled, you do not need to restart the Solstice Enterprise Agents software master agent, `snmpdx`, after restarting the SMA master agent, `snmpd`.

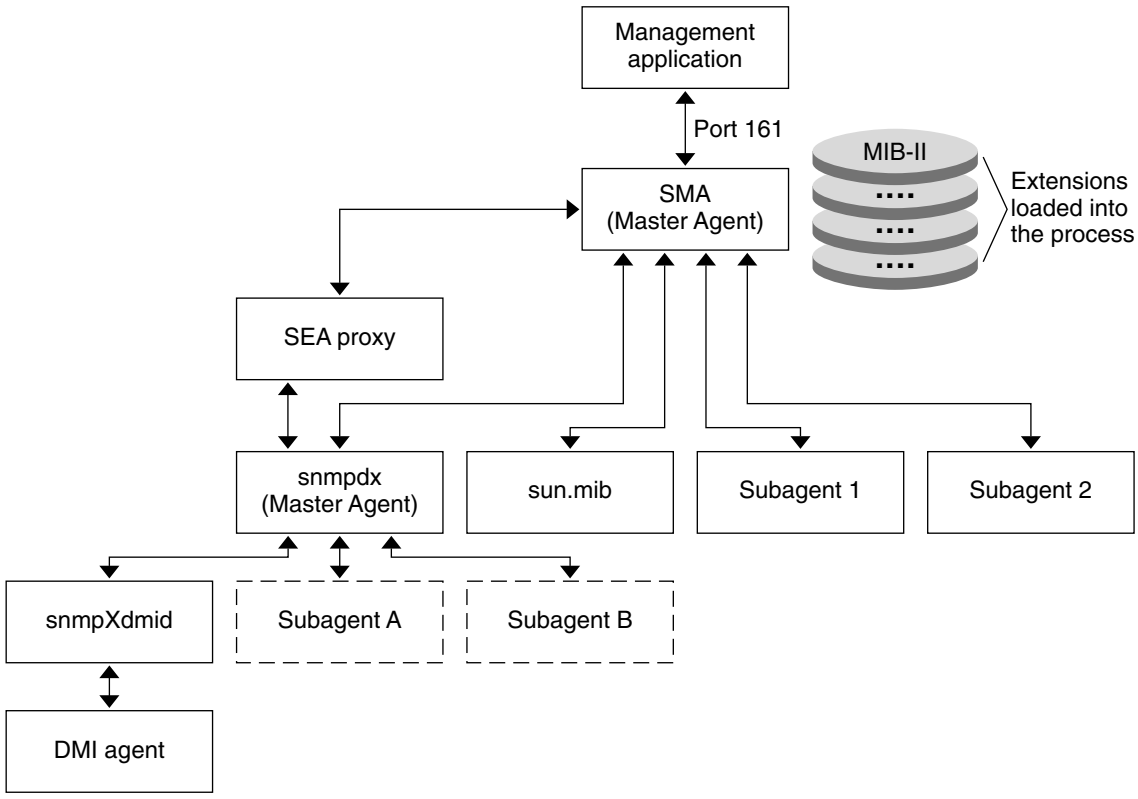

FIGURE 5–2 Routing of Requests With both SEA and SMA Present, Using the `seaProxy` Module and Proxy Statements

Migration From the Sun Fire Management Agent

The Sun SNMP Management Agent for Sun Fire™ and Netra™ Systems is a standalone SNMP agent that supports the following servers:

- Sun Fire v210 server.
- Sun Fire v240 server.
- Sun Fire v250 server.
- Sun Fire v440 server.
- Netra 240 server.
- Netra 440 server.

The Sun SNMP Management Agent for Sun Fire and Netra Systems provides SNMP–based access to hardware inventory and environmental monitoring. For more information, see the SNMP Management Agent Guide for the Sun Fire and Netra Systems. If you are running your Solaris 10 Operating System on any of the above servers, you should migrate from the Sun SNMP Management Agent for Sun Fire and Netra Systems to the SMA.

As with the System Management Agent, the Sun SNMP Management Agent for Sun Fire and Netra Systems is an SNMP agent that also uses a daemon that is named `snmpd`. If these two separate `snmpd` daemons are both running, make sure when stopping the `snmpd` daemon used by the Sun SNMP Management Agent for Sun Fire and Netra Systems, that you are stopping the desired daemon. The System Management Agent's `snmpd` daemon is located at `/usr/sfw/sbin/snmpd`. If you start the `snmpd` daemon manually from `/usr/sfw/sbin`, then the `init.sma` script does not work. In this case, you must kill the daemon manually:

```
# ps -edf | grep snmpd
```

Note – If the SMA `snmpd` daemon is not started by use of the `init.sma` script, then the daemon can *not* be shutdown by the `init.sma` script. When starting the `snmpd` daemon, use the `init.sma` script.

You must migrate to the System Management Agent from the Sun SNMP Management Agent for Sun Fire and Netra Systems. You can not run the Sun SNMP Management Agent for Sun Fire and Netra Systems on port 161 or port 162.

The `masfcnv` Migration Script

This section provides a procedure to migrate the configuration of the Sun SNMP Management Agent for Sun Fire and Netra Systems to the SMA. The procedure uses the `masfcnv` script. This script is designed specifically for migrating to the SMA SNMP agent from the Sun SNMP Management Agent for Sun Fire and Netra Systems.

The `./masfcnv` migration script is located at `/usr/sfw/lib/sma_snmp`. The `./masfcnv` migration script performs the following functions:

- The script migrates USM (SNMP) user names and passwords. The script checks that USM user names migrated from the Sun SNMP Management Agent for Sun Fire and Netra Systems to the SMA do not already exist in the SMA. If a duplicate exists, you need to determine whether the identified user should be treated as the same user in the SMA. For information on USM within the SMA, see "Using USM for Authentication and Message Privacy" on page 42.

  You must decide whether you want to migrate the Sun SNMP Management Agent for Sun Fire and Netra Systems key that is associated with that user. The alternative is to continue to use the existing System Management Agent key for that user. The Sun SNMP Management Agent for Sun Fire and Netra Systems only supports MD5 based keys. The SMA supports additional authentication schemes such as SHA and encryption (DES) for SNMP requests. A migrated user is therefore be unable to use these additional capabilities until the necessary keys have been configured. However, access based on MD5 authentication is available to such users. For more information on authentication and encryption, see "Authentication Protocol Algorithms" on page 43.

- The script uses the `snmpd.conf` template file that is located at `/usr/sfw/lib/sma_snmp/`. The script uses this template file to create a new `snmpd.conf` agent configuration file. This new `snmpd.conf` agent configuration file is specifically for the Sun SNMP Management Agent for Sun Fire and Netra Systems. This new `snmpd.conf` agent configuration file is installed at `/etc/opt/SUNWmasf/conf/`. The Sun SNMP Management Agent for Sun Fire and Netra Systems uses this new `snmpd.conf` agent configuration file to modify the SMA main configuration file. The Sun SNMP Management Agent for Sun Fire and Netra Systems also uses its agent configuration file to modify the SMA persistent storage file at `/var/sma_snmp/snmpd.conf`.

  For more information on SMA configuration files, see "Configuration Files and Scripts" on page 28.

- The script replaces Sun SNMP Management Agent for Sun Fire and Netra Systems configuration files by a default configuration. This default configuration sets up the Sun SNMP Management Agent for Sun Fire and Netra Systems as an AgentX subagent.

- The script makes back ups of the changed configuration files. Configuration file back ups are made by appending the extension `.bak.n` to the filename where `n` is an optional number.

- The script replaces the existing Sun SNMP Management Agent for Sun Fire and Netra Systems startup script in `/etc/init.d` with a new script.

- The script migrates the VACM configuration. The Sun SNMP Management Agent for Sun Fire and Netra Systems configuration related to the OID space used by the SUN MIB is migrated automatically. VACM configuration can be related to other OIDs. For example, VACM information can be related to the system branch in MIB-II. If VACM information is related to other OIDs, you must confirm if migration is required. For more information on VACM, see "Using VACM for Access Control" on page 46.
- The script migrates trap destinations from the Sun SNMP Management Agent for Sun Fire and Netra Systems to the SMA. Entries originally configured for both agents do not result in duplicate entries in the migrated configuration.
- The script migrates community strings from the Sun SNMP Management Agent for Sun Fire and Netra Systems to the SMA. You are advised if an identical string is configured for both agents.

After migration, the SMA provides SNMP access on its standard ports 161/162. The SMA provides access on other ports if you configure it. The SMA also provides SNMP access on the ports previously used by the Sun SNMP Management Agent for Sun Fire and Netra Systems. All ports provide access to the same set of OIDs. These OIDs include OIDs used by the SUN-PLATFORM-MIB as used by the Sun SNMP Management Agent for Sun Fire and Netra Systems. You can configure additional access controls to limit the visibility of data on a user basis.

If you are migrating user names and passwords from the Sun SNMP Management Agent for Sun Fire and Netra Systems, the `engineID` used by the SMA must be the same as that previously used by the Sun SNMP Management Agent for Sun Fire and Netra Systems. USM, used by SNMPv3, embeds the `engineID` into the keys used for authentication. If you have configured the SMA to use a different `engineID` to that of the Sun SNMP Management Agent for Sun Fire and Netra Systems, you must determine which `engineID` to use. If the `engineID` is different to that originally used by the Sun SNMP Management Agent for Sun Fire and Netra Systems, reset those passwords used by migrated users. For more information on the USM, see "Using USM for Authentication and Message Privacy" on page 42.

For further information on the `masfcnv` script, see the `masfcnv`(1M) man page.

Note – In all cases, the Sun SNMP Management Agent for Sun Fire and Netra Systems agent runs independently of the Solstice Enterprise Agents' executable, `snmpdx`. If you stop the Sun SNMP Management Agent for Sun Fire and Netra Systems agent, you do automatically stop the Solstice Enterprise Agents software. You must migrate to the System Management Agent from the Solstice Enterprise Agents software. For more information, see "Migration From Solstice Enterprise Agents Software" on page 65.

▼ To Migrate From the Sun SNMP Management Agent for Sun Fire and Netra Systems to the SMA

1. **As root, stop both the System Management Agent and the `masfd` agents.**

   ```
   # /etc/init.d/init.sma stop
   # /etc/init.d/masfd stop
   ```

   Any other agents that have been configured as subagents of SMA also need to be stopped and restarted after the migration is complete.

2. **Perform a test migration to determine the effect of running the migration script.**

   A test migration is useful if you have made significant configuration changes to the System Management Agent.

   ```
   # cd /usr/sfw/lib/sma_snmp
   # ./masfcnv --dry-run -i -p enable --select-community=agent
   ```

   If this dry run completes successfully, the proposed SMA configuration files are be presented in the standard output. Review this output before proceeding. The configuration of the Sun SNMP Management Agent for Sun Fire and Netra Systems is migrated to the SMA by the `./masfcnv` migration script. If a conflict arises in the configuration, see the `masfcnv`(1M) man page for information on its resolution.

3. **Run the migration script.**

   ```
   # cd /usr/sfw/lib/sma_snmp
   # ./masfcnv -i -p enable --select-community=agent
   ```

4. **As root, restart both the System Management Agent and the Sun SNMP Management Agent for Sun Fire and Netra Systems.**

   ```
   # /etc/init.d/init.sma start
   # /etc/init.d/masfd start
   ```

   The Sun SNMP Management Agent for Sun Fire and Netra Systems is then reconfigured to run as a subagent under the System Management Agent. Any other agents that have been configured as subagents of the System Management Agent also need to be restarted after the migration is complete.

   After migration to the SMA from the Sun SNMP Management Agent for Sun Fire and Netra Systems, the Sun Fire hardware instrumentation becomes accessible to SNMP applications through the SMA. The SMA uses the same port that was previously used by the Sun SNMP Management Agent for Sun Fire and Netra Systems.

APPENDIX **A**

Tools and Man Pages

This appendix describes the various tools and man pages that are available in the System Management Agent.

Tools and Utilities Configuration File

In the System Management Agent, as in the standard Net-SNMP implementation, configuration of the available tools and utilities is done through the `snmp.conf` configuration file. The `snmp.conf` configuration file is located at `/etc/sma/snmp/`.

Before modifying the `snmp.conf` configuration file, read "Configuration Files and Scripts" on page 28. Ensure also that you read the `snmp_config`(4) and `snmp.conf`(4) man pages in that order.

The `snmp.conf` configuration file supports the directives listed in the `snmp.conf` man page. If you store sensitive information in this file, ensure that you set the permissions so that the file is readable only by the user.

Man Pages

This section lists all the man pages that are associated with the System Management Agent. The man pages are listed in tables, which are organized by the type of content:

- Table A–1
- Table A–2
- Table A–3

- Table A–4
- Table A–5

The following table lists man pages for general SNMP information.

TABLE A–1 Man Pages for General SNMP Topics

Man Page	Description
`netsnmp`(5)	Gives an overview of the Net-SNMP implementation included in the Solaris software. Also available as the `sma_snmp`(5) man page.
`snmpcmd`(1M)	Describes the common options for Net-SNMP commands.
`snmp_variables`(4)	Discusses the format that must be used to specify variable names to Net-SNMP commands.

The following table lists the man pages for Net-SNMP command tools.

TABLE A–2 Man Pages for SNMP Tools

Man page	Tool Description
`mib2c`(1M)	The `mib2c` tool uses nodes in a MIB definition file to produce two C code template files to use as a basis for a MIB module.
`snmpbulkget`(1M)	The `snmpbulkget` utility is an SNMP application that uses the SNMP GETBULK operation to send information to a network manager.
`snmpbulkwalk`(1M)	The `snmpbulkwalk` utility is an SNMP application that uses SNMP GETBULK requests to query a network entity efficiently for a tree of information.
`snmpget`(1M)	The `snmpget` utility is an SNMP application that uses the SNMP GET request to query for information on a network entity.
`snmpgetnext`(1M)	The `snmpgetnext` utility is an SNMP application that uses the SNMP GETNEXT request to query for information on a network entity.
`snmpinform`(1M)	The `snmpinform` command invokes the `snmptrap` utility, which is an SNMP application that uses the SNMP TRAP operation to send information to a network manager.

TABLE A–2 Man Pages for SNMP Tools *(Continued)*

Man page	Tool Description
`snmpnetstat`(1M)	The `snmpnetstat` command symbolically displays the values of various network-related information retrieved from a remote system using the SNMP protocol.
`snmpset`(1M)	The `snmpset` utility is an SNMP application that uses the SNMP SET request to set information on a network entity.
`snmptrap`(1M)	The `snmptrap` utility is an SNMP application that uses the SNMP TRAP operation to send information to a network manager.
`snmpusm`(1M)	The `snmpusm` utility is an SNMP application that can be used to do simple maintenance on an SNMP agent's User-based Security Module (USM) table.
`snmpvacm`(1M)	The `snmpvacm` utility is a SNMP application that can be used to do maintenance on an SNMP agent's View-based Access Control Module (VACM) table.
`snmpwalk`(1M)	The `snmpwalk` utility is an SNMP application that uses SNMP GETNEXT requests to query a network entity for a tree of information.
`snmpdf`(1M)	The `snmpdf` command is a networked version of the `df` command. `snmpdft` checks the disk space on the remote machine by examining the HOST-RESOURCES-MIB's `hrStorageTable` or the UCD-SNMP-MIB's `dskTable`.
`snmpdelta`(1M)	The `snmpdelta` utility monitors the specified integer valued OIDs. The utility reports changes over time.
`snmptable`(1M)	The `snmptable` utility is an SNMP application that repeatedly uses the SNMP GETNEXT or GETBULK requests to query for information on a network entity.
`snmptest`(1M)	The `snmptest` utility is a flexible SNMP application that can monitor and manage information on a network entity. The utility uses a command-line interpreter to enable you to send different types of SNMP requests to target agents.
`snmptranslate`(1M)	The `snmptranslate` utility is an application that translates one or more SNMP object identifier values from their symbolic, textual forms into their numerical forms. The application also translates one or more SNMP object identifier values from their numerical forms into their symbolic, textual forms.

TABLE A–2 Man Pages for SNMP Tools *(Continued)*

Man page	Tool Description
`snmpstatus`(1)	The `snmpstatus` command is an SNMP application that retrieves several important statistics from a network entity.

The following table lists the man pages associated with configuration files that are used by the Net-SNMP agent.

TABLE A–3 Man Pages for SNMP Configuration Files

Man Page	Description
`snmp_config`(4)	Provides an overview of the Net-SNMP configuration files included with System Management Agent.
`snmp.conf`(4)	The file `snmp.conf` defines how the Net-SNMP applications such as `snmpget` or `snmpwalk` operate.
`snmpd.conf`(4)	The file `snmpd.conf` defines how the Net-SNMP agent operates.
`snmptrapd.conf`(4)	The file `snmptrapd.conf` defines how the Net-SNMP trap-receiving daemon, `snmptrapd`, operates when the daemon receives a trap.
`snmpconf`(1M)	The `snmpconf` utility is a script that asks you questions. The script creates an `snmpd.conf` configuration file that is based on your responses.

The following table lists the man pages for daemons that are associated with Net-SNMP.

TABLE A–4 Man Pages for SNMP Daemons

Man Page	Description
`snmpd`(1M)	The `snmpd` daemon is the SNMP agent. The daemon binds to a port and awaits requests from SNMP management software.
`init.sma`(1M)	The `init.sma` utility is run automatically during installation and each time the system is rebooted. This utility manages the `snmpd`.
`snmptrapd`(1M)	The `snmptrapd` daemon is an SNMP application that receives and logs SNMP TRAP and INFORM messages.

TABLE A–5 Man Pages for Migration Scripts

Man Page	Description
`masfcnv`(1M)	The `masfcnv` migration script can be used to help you migrate an existing set of configuration files for Sun SNMP Management Agent for Sun Fire and Netra Systems to the SMA.

Glossary

agent	A software program, typically run on a managed device, that implements the SNMP protocol. The program also services the requests of a manager. Agents can act as proxies for some non-SNMP manageable network nodes.
Agent Extensibility Protocol (AgentX)	A subagent protocol that can communicate with a master SNMP agent.
configuration token	Tokens can be identifiers, keywords, constants, punctuation, or white space.
context	A collection of managed objects that are accessible by an SNMP entity. The name for a subset of managed objects.
DAQ	Data acquisition.
DES	Data Encryption Standard
legacy subagent	See proxy agent.
Management Information Base (MIB)	A virtual information store for managed objects. MIBs define the properties of a managed object within the device to be managed.
manager	A client application that accesses data from a managed device or system.
master agent	An agent running on a designated port.
MIB II	The current standard definition of the virtual file store for objects that can be managed by SNMP.
mib2c	A utility that compiles MIBs and generates a syntax template for MIB implementation.
Object Identifier (OID)	Every managed object, whether the object is a device or the characteristics of a device, has a name, a syntax, and an encoding. The name, an object identifier (OID), uniquely identifies the object. The

	OID is written as a sequence of integers separated by periods. For example, the sequence 1.3.6.1.2.1.1.1.0 specifies the system description within the system group of the management subtree.
PDU	Protocol Data Unit. This unit defines the type of SNMP message. A PDU contains control fields and an array of pairs. The control fields are dependent on the message type. The first element of each of the pairs in the array identifies management data. The second element of each of the pairs in the array specifies the value of this management data.
proxy agent	An agent that acts on behalf of a non-SNMP (foreign) network device. The management station contacts the proxy agent and indicates the identity of the foreign device. The proxy agent translates the protocol interactions it receives from the management station into the interactions supported by the foreign device.
MD5	The message digest function defined in RFC 1321.
net-snmp	The Open Source version of the SMA. SMA uses net-snmp base functionality and supports SNMP protocol versions 1, 2, and 3.
SHA1	Secure Hash Algorithm - Version 1.0. SHA is a cryptographic message digest algorithm.
Structure of Management Information (SMI)	An industry-accepted method of organizing object names so that logical access can occur. The SMI states that each managed object must have a name, a syntax, and an encoding. The name, an object identifier (OID), uniquely identifies the object. The syntax defines the data type, such as an integer or a string of octets. The encoding describes how the information associated with the managed objects is serialized for transmission between machines.
subagent	An agent that interacts with a master agent.
System Management Agent (SMA)	An management agent based on open source Net-SNMP but carrying some Sun modifications and additional tools and wrappers.
trap	A message sent to a manager that describes exceptions that occurred on a managed device.
USM	User-based Security Model. A standard for providing SNMP message-level security, described in RFC 3414 at `http://www.ietf.org/rfc/rfc3414.txt`. This RFC document also includes a MIB for remotely monitoring and managing the configuration parameters for the User-based Security Model.
VACM	View-Based Access Control Mechanism. A standard for controlling access to management information, described in RFC 3415 at `http://www.ietf.org/rfc/rfc3415.txt`. This RFC document also includes a MIB for remotely managing the configuration parameters for the View-based Access Control Model.

Index

T

U

V