Solstice Enterprise Agents 1.0 User Guide

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

Part No: 816–1322–10
May 2002

020115@3062

Contents

Preface

The *Solstice Enterprise Agents User Guide* is an introduction to the product and explains how to install, configure, and administer the Solstice Enterprise Agents runtime environment and associated subagents.

Who Should Use This Book

This document is targeted towards System Administrators.

Before You Read This Book

If you have just acquired the Solstice Enterprise Agents product, this should be the first book you read. It gives an overview of the product architecture functions, features, and components.

How This Book Is Organized

This document contains the following chapters:

Chapter 1 is an overview of the product.

Chapter 2 explains how to install Solstice Enterprise Agents on your Solstice machine.

Chapter 3 explains how the Master Agent works.

Chapter 4 provides configuration information.

Chapter 5 provides an overview of the Desktop Management Interface.

Chapter 6 explains how SNMP accesses DMI components and gives details on how to use mapping to communicate between SNMP and DMI protocols.

Chapter 7 explains the function of the command line interface and gives examples of its use.

Appendix A provides a list of error messages and their numbers.

The Glossary identifies acronyms and abbreviations and gives definitions of terms used in this book.

Related Books

The additional Sun publications listed below contain information related to this user guide.

- *System Administration Guide: Basic Administration*

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`.

Typographic Conventions

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

TABLE P–1 Typographic Conventions

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`
`AaBbCc123`	What you type, contrasted with on-screen computer output	`machine_name%` **`su`** `Password:`
AaBbCc123	Command-line placeholder: replace with a real name or value	To delete a file, type **`rm`** *filename*.
AaBbCc123	Book titles, new words, or terms, or words to be emphasized.	Read Chapter 6 in *User's Guide*. These are called *class* options. You must be *root* to do this.

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

What Is a Solstice Enterprise Agent?

1.1 Overview of Enterprise Agent

The Simple Network Management Protocol (SNMP) has been widely used in enterprise networks to effectively manage systems, network devices, and networks. The widespread use of SNMP has raised many issues relating to managing systems and networks. One of the benefits of SNMP is how quickly solutions may be created to support the increasing numbers of networking components and applications.

Within SNMP networks, the number of entities (systems, components, and applications) that need to be managed is growing rapidly. There is a need to respond to the industry's demand for more flexible and dynamic management of multiple devices.

The initial network management solution, that is based on SNMP, allowed developers to create one monolithic agent per system/device listening on a single port (port 161). It was soon discovered that this SNMP solution had many constraints and was not flexible enough to effectively manage all the devices necessary.

New technology was needed to produce multiple agents by different people, that could manage different components and applications separately within a device. This resulted in the new extensible agent technology or Master/subagent technology. Based on this technology, Sun provides a solution named Solstice Enterprise Agent (SEA).

The agents consist of Master Agent and subagents. The Master Agent receives the SNMP-based management requests from the managers and sends responses to these management requests. The responses are sent after retrieving the appropriate values from the respective subagents. The subagents provide management of different components based on Management Information Based (MIBs or MIFs) specifically designed for components/applications.

The Enterprise Agent also allows you to integrate and use SNMP-based legacy agents.

In subsequent chapters, the roles of the Master Agent and subagents are discussed in detail.

1.2 Enterprise Agent Components

The SNMP based component of the SEA product consists of various components.

Figure 1–1 illustrates the architecture of the SEA.

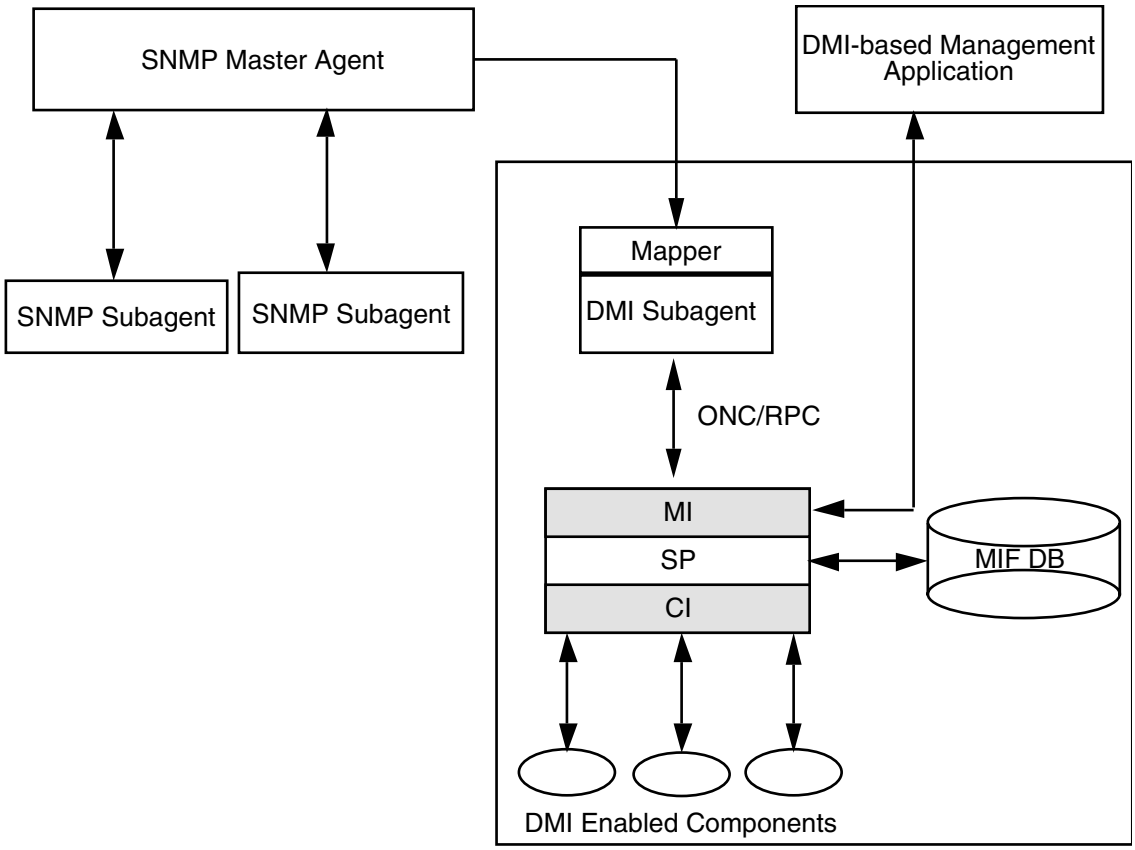

FIGURE 1–1 Architecture of the SEA

The following is a description of each of the components associated with the SEA product.

1.2.1 SNMP Master Agent

The Master Agent is an entity or process on a managed node that exchanges SNMP protocol messages with the Managers (for example, Domain Manager, Enterprise Manager, and HP Openview).

The Master Agent listens on port 161.

1.2.2 Subagents

Subagents are zero or more processes that have access to the management information and provide manageability to various applications/components within a system. These subagents interact with the Master Agent using SNMP. These subagents do not interact with the managers directly.

1.2.3 Agent/Subagent Software Development Toolkit

The Software Development Toolkit has multiple components. It includes agent/subagent libraries, a MIB compiler, and example subagents. The MIB compiler parses a MIB and creates stub files. The stub files consist of functions that you modify and enhance appropriately to provide manageability of the respective component or application.

1.2.4 Legacy SNMP Agents

Legacy SNMP Agents are SNMP-based and work as monolithic entities in a system. The Enterprise Agent allows the integration of legacy SNMP agents. The legacy agents are those agents already in released products from Sun or other companies.

1.2.5 Mapper

The Enterprise Agent technology also allows you to integrate DMI 2.0 functionality. This is accomplished through the mapper, that acts as a subagent. The mapper receives requests from the Master Agent and converts them into appropriate DMI requests, that are then sent to the DMI Service Provider. When the mapper receives the response back from the DMI Service Provider, it converts this response into the SNMP response and forwards it to the Manager through the Master Agent.

1.2.6 Terminology

Following are additional terms you need to be familiar with when using the SEA product.

Registration occurs when the Master Agent receives information from a subagent and then provides the subagent with management of a MIB subtree.

A *subtree* is indicated by a single oid. The Master Agent has no understanding of what this subtree is without any MIB specification. The subtree may actually be an entire MIB (e.g., 'host'), a full instance (e.g., hrDeviceDescr.42), or may not even be a subtree named in any MIB specification.

An *oid* range is the range of oids implied by a subtree. For instance, the subtree 1.2.3 carries an implied range of 1.2.3 up to but not including 1.2.4.

A *duplicate registration* is an attempt by one subagent to register a subtree that exactly matches a subtree already registered by another subagent.

An *overlapping registration* is an attempt by one subagent to register a subtree that is contained within, or contains, a subtree already registered by another subagent.

Dispatching is the communication of a management request from the Master Agent to one or more subagents. Dispatching is performed according to the Master Agent's current view of registered subtrees, and an explicitly stated algorithm.

Additional terms are described in this guide's glossary.

1.3 SNMP Functionality

The Master Agent receives SNMP requests from the system managers and sends responses to these requests, after determining appropriate values from the subagents. The subagents provide management of different components based on the Management Information Base (MIB) specifically designed for such components/applications. Each subagent, when invoked dynamically, registers with the Master Agent. During registration, it informs the Master Agent of the MIB subtree it manages. For more information, refer to Chapter 3.

The SEA technology provides a software development kit that allows you to create, release, and install subagents. Additionally, the SEA allows you to integrate and use SNMP-based legacy agents.

1.4 What is DMI?

The Desktop Management Interface (DMI) is a set of interfaces and a service provider that mediate between management applications and components residing in a system. The DMI is a free-standing interface that is not tied to any particular operating system or management process.

Sun provides DMI based functionality for management of the Sun platforms (hardware and software) and software applications running on these platforms. The DMI subagent is one type of subagent included in the SEA product. By using DMI, you may manage various elements within most systems (for example, PCs, workstations, routers, hubs, and other network objects).

DMI has four elements:

- A format for describing management information (MIF)
- A Service Provider entity
- Two sets of APIs
  - An interface between management applications and the Service Provider
  - An interface between the Service Provider and component instrumentation
- A set of services (using ONC/RPC) for facilitating remote communication

For more information, refer to Chapter 5.

1.5 Master Agent

The Master Agent acts as the primary interface between the network manager and the subagents. The requests received from the manager are parsed by the Master Agent. If necessary, the original requests are broken into multiple requests. The original request is distributed by the Master Agent based on the manageability provided by each subagent. The request is then forwarded to the appropriate subagents, which provide a response to each request. After collecting all the responses from each subagent, the final response is sent to the network manager.

Only one Master Agent presides over the Master/subagent model. The Master Agent acts as a request scheduler and dispatcher for all subscribed subagents. In addition, the subagents send traps to the Master Agent, that are then forwarded to the manager.

Figure 1–2 illustrates the Master Agent as it relates to the architecture of SEA.

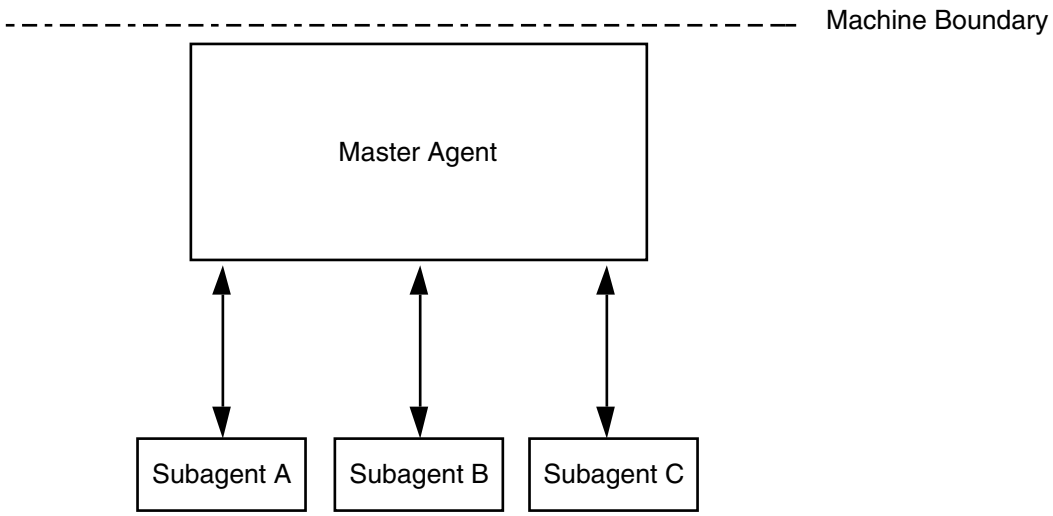

FIGURE 1–2 Architecture of the Master Agent

CHAPTER 2

Installing Enterprise Agents

2.1 Installing Enterprise Agents Overview

Installing Solstice Enterprise Agents (SEA) on the Solaris operating system follows Standard Solaris conventions, as described in the following sections.

The SEA package is divided into two major parts:

- SNMP
- DMI

2.2 Platforms and Packages

This product is supported on the following platforms:

- Sparc (Solaris 2.4, 2.5, and 2.6)
- IA (Solaris 2.4, 2.5, and 2.6)

The runtime product includes the following unique packages:

- SUNWsacom
- SUNWsasnm
- SUNWsadmi
- SUNWmibii
- SUNWsasdk (optional package for agent development)

The following sections describe the details of each package.

2.2.1 SUNWsacom

The SUNWsacom package contains all the configuration files corresponding to the other three packages. The files in this package are installed in the `/root` and `/var` file systems. These files include configuration files and other common files.

2.2.2 SUNWsasnm

The SUNWsasnm package includes:

- `snmpdx` – Master Agent executable
- `init.snmpdx` – Startup script file
- `snmpdx.mib` – Master Agent MIB file
- Libraries

2.2.3 SUNWsadmi

Within SUNWsadmi, the package includes:

- `dmispd` – DMI Service Provider (SP) executable
- DMI libraries – See Table 2–3
- `snmpXdmid` – DMI/SNMP mapper executable
- `init.dmi` – Startup script file
- `ciinvoke` – Script to invoke CI agents, from `init.dmi`
- Script to invoke CI agents – Invoked by `ciinvoke`
- `dmi_cmd` and `dmiget` – Command line MI utilities

2.2.4 SUNWmibii

The SUNWmibii package contains the `mibiisa` MIB II subagent. The subagent provides MIB II functionality. The functionality provided by MIB II subagent is the same as that provided by the `snmpd` agent released with Domain Manager.

2.2.5 SUNWsasdk

This package contains SNMP Toolkit and the DMI Toolkit to build the subagents.

SNMP Toolkit contains:

- `mibcodegen`
- Header files
- Standard MIB files
- Sample MIB and Example Code

DMI Toolkit includes:

- miftomib compiler
- libdmi
- libci
- libmi
- sp.mif
- Sample files for CI and MI

2.3 Installation Procedure

Packages are installed using the `pkgadd` command and are removed by the `pkgrm` command.

Note – The `snmpXdmid` must be configured properly in the Master Agent configuration files.

▼ 2.3.1 Remove Existing Packages

Caution – Before starting your installation procedure, be sure you log in as root. Then follow the procedures below.

If these packages don't exist, you don't have to remove them.

▼ 2.3.2 Add New Packages

With the addition of each package, you receive both the English and French copyrights, then a series of prompts that you can either answer specifically or accept the default.

1. **Add SUNWmibii:**

   ```
   pkgadd -d . SUNWmibii
   ```

2. **Add SUNWsasnm:**

   ```
   pkgadd -d . SUNWsasnm
   ```

3. **Add SUNWsadmi:**

   ```
   pkgadd -d . SUNWsadmi
   ```

4. **Add SUNWsacom:**

   ```
   pkgadd -d . SUNWsacom
   ```

5. **Add SUNWsasdk**

   ```
   pkgadd -d . SUNWsasdk
   ```

   When you add the `SUNWsadmi` package, the `dmispd` and `snmpXdmid` processes start upon reboot of the system.

   The SUNWsacom package consists of two script files; `init.dmi` and `init.snmpdx`. The SNMP daemon, `snmpdx`, is invoked by `init.snmpdx`. The `snmpdx` process automatically starts the `mibiisa` daemon.

   The `dmispd` process is invoked through the RC script files when the system is booted, as is `snmpdx`. The `snmpXdmid` mapper process is invoked after the `dmispd` has been invoked.

Note – When deleting or adding packages, be sure you follow the order described above.

2.4 SNMP Default Software Locations

Table 2–1 shows a list of default software locations within SNMP.

TABLE 2–1 Default SNMP Locations

Label	Directory
SEA_SNMPLibrary_Directory	/usr/lib
SEA_SNMPConfiguration_Directory	/etc/snmp/conf
SEA_SNMPMibs_Directory	/var/snmp/mib

Table 2–2 provides the list of SNMP component names and their default locations.

TABLE 2–2 SNMP Package Components

Component Name	Label/Directory	Description
snmpdx	/usr/lib/snmp	Master Agent executable
mibiisa	/usr/lib/snmp	MIB II snmp daemonpwd
snmp_trapsend	/usr/sbin	Utility for sending traps
mibcodegen	/usr/bin	Code generator executable
snmpdx.rsrc	SEA_SNMPConfiguration_Directory	Master Agent resource file
snmpdx.reg	SEA_SNMPConfiguration_Directory	Registration file for agents
snmpdx.acl	SEA_SNMPConfiguration_Directory	Master Agent access control file
snmpd.conf	SEA_SNMPConfiguration_Directory	SNMPD configuration file
mibiisa.reg	SEA_SNMPConfiguration_Directory	MIB II subagent registration file
mibiisa.rsrc	SEA_SNMPConfiguration_Directory	MIB II agent resource file
snmpdx.st	SEA_SNMPRun_Time_Directory	Master Agent status file
libssasnmp.so.1	SEA_SNMPLibrary_Directory	SSA SDK SNMP library
libssagent.so.1	SEA_SNMPLibrary_Directory	SSA SDK Agent library
enterprises.oid	SEA_SNMPConfiguration_Directory	Default enterprise-name OID map
sun.mib	SEA_SNMPMibs_Directory	Sun MIB
snmpdx.mib	SEA_SNMPMibs_Directory	Snmpdx MIB

2.5 DMI Default Software Locations

Table 2–3 provides a list of default software locations for the DMI portion of the product.

TABLE 2–3 Default Locations for DMI

Label	Location
SEA_DMILibrary_Directory	/usr/lib
SEA_DMIConfiguration_Directory	/etc/dmi/conf
SEA_DMIRunTime_Database_Directory	/var/dmi/db
SEA_DMIRunTime_MAP_Directory	/var/dmi/map
SEA_DMIMif_Directory	/var/dmi/mif

Table 2–4 provides a list of DMI component names and their default locations.

TABLE 2–4 DMI Package Components

Component name	Label/Directory	Description
snmpXdmid	/usr/lib/dmi	Mapper executable
dmispd	/usr/lib/dmi	DMI Service Provider executable
dmi_cmd	/usr/sbin	DMI command utility
dmiget	/usr/sbin	DMI command utility
snmpXdmid.conf	SEA_DMIConfiguration_Directory	Mapper configuration file
dmispd.conf	SEA_DMIConfiguration_Directory	DMI SP configuration file
map files	SEA_DMIRunTime_MAP_Directory	Map files
libdmi.so.1	SEA_DMILibrary_Directory	SSA SDK DMI generic library
libci.so.1	SEA_DMILibrary_Directory	SSA SDK CI library
libdmimi.so.1	SEA_DMILibrary_Directory	SSA SDK MI library
sp.mif	SEA_DMIMif_Directory	MIF files
ciinvoke	/etc/dmi/ciagent	DMI component interface invocation script

CHAPTER 3

SNMP-Based Master/Subagent

3.1 SNMP-Based Master/Subagent Overview

The Master Agent is the main component of Solstice Enterprise Agent technology. It runs as a daemon process and listens to User Datagram Protocol (UDP) port 161 for SNMP requests. The Master Agent also opens another port to receive SNMP trap notifications from various subagents. These traps are forwarded to various managers, as determined by the configuration files.

3.1.1 Invoking the Master Agent

When the system is initially booted, a system startup script file invokes the Master Agent if the `/etc/snmp/conf/snmpdx.rsrc` configuration file contains neither blank lines nor comment lines. Upon the Master Agent's invocation, it reads its various configuration files and appropriate actions are performed by activating subagents, determining the subtree OID for various subagents, populating its own MIBs. The Master Agent provides the following functionality:

- Invokes subagents
- Communicates with subagents
- Registers subagents
- Sends requests to subagents
- Receives responses from subagents

- Traps notifications from subagents

Note – If you have `snmpd` (part of Domain Manager) or some other SNMP agent using port 161 running, the Solstice Enterprise Agents cannot run.

3.1.2 Invoking the Subagent

A subagent can be invoked in the following ways:

- By the Master Agent - the Master Agent may invoke all agents with the agent resource file existing under `SEA_SNMPConfiguration_Directory`. The agents are invoked as specified in the Master Agent resource file. If the Master Agent successfully invokes a subagent, it creates a row entry in the subagent table and fills in the MIB variables with appropriate values for that subagent. A subagent is then registered with the Master Agent and is then ready to receive SNMP requests from the Master Agent.
- Manually or automatically at boot time - system administrators/users may manually invoke subagents that have no agent resource file; or, startup scripts can invoke the subagents when the system is booted. Such agents can be invoked only after the Master Agent has been invoked. Additionally, these agents must have been created using the Solstice Enterprise Agent Software Development Kits (SDKs) and linked with the appropriate libraries. This allows the subagents to register dynamically with the Master Agent.

3.1.3 Communicating With the Subagent

Any communication from subagents to the Master Agent is done through UDP port 161. The topic of sending traps to the Master Agent from the subagents is discussed in “3.1.6 Trap Notification” on page 28.

Note – The Master Agent communicates on separate ports for each subagent.

The Master Agent also checks to be sure that registered subagents are up and running, based on the following conditions:

- Time-out mechanism for each Get, Get Next, and Set request is sent to each subagent by the Master Agent.
- No activity between the Master Agent and the subagents. The Master Agent (based on `watch_dog` time as defined in the agent resource file) is used to determine whether a particular subagent is active. This is accomplished by sending an SNMP Get request to the subagent if there has been no activity between the subagent and

the Master Agent for some specific configured period of time.

3.1.4 Registering the Subagent

To register the subagent, the Master Agent binds it to the MIB. The Master Agent then determines its present location, using one of the following methods:

- Static method – The Master Agent reads agent resource files. This resource file contains an entry for each subagent.
- Dynamic method – The Master Agent receives the information from the subagents. The subagent sends a SET request containing the MIB objects needed to register with the Master Agent, through the use of the registration API.

The binding policy relates to the registration of SNMP object identifiers (OIDs). It involves decision-making on the part of the Master Agent when dispatching SNMP requests to various subagents. The Master Agent supports the binding policy, as shown in Table 3–1.

TABLE 3–1 Binding Policy

Type of Registration	Method of Registration
Individual variable registration	A subagent can manage individual variables
Row registration	A subagent can manage each row or multiple rows
Table registration	A subagent can register full and partial tables; partial table registration means that some columns of a table can be registered; for example, if a table has columns c1–c5, a subagent can then register a partial table managing c3 and c5 columns (only) of that table
Duplicate registration	NOT allowed
Overlapping registration	In the case of overlapped registration, the Master Agent dispatches requests on the basis of best OID match

3.1.5 Sending Requests

The Master Agent supports the forwarding of SNMP requests (Get, Get Next, and Set) in two modes. The mode is indicated by providing an optional argument with the command-line invocation. These modes are:

- Group mode – Multiple variables can be included in each request from the Master Agent to the subagents. This results in one send-request per agent.
- Split mode – Each variable in the incoming request results in one send-request to each subagent.

Following is a list of allowable send requests.

- `GET` and `GETNEXT` – Sends requests to subagents.
- `SET` – The set is implemented in a multi-phase fashion. First, all the `varbinds` in the set request are retrieved using a Get request. If successful, a Set request is sent to the respective subagents. If the Set request succeeds, a `SUCCESS` response is sent to the manager. If the Set request does not succeed, another Set request is sent to the subagents, with original values. This cancels the failed Set request. Any other SNMP requests being processed are completed before the Set request can be initiated.
- `GET RESPONSE` – Receives responses from subagents.
- `TRAP` – Receives notifications from subagents.

3.1.6 Trap Notification

The subagents send the traps to the Master Agent; the Master Agent the decides which managers will receive the trap. This decision is configurable.

3.2 Description of the Subagent

Subagents do not interact with the network manager directly. Instead, they communicate with the Master Agent. The management responsibility of the subagent can be offered to the Master Agent; however, it is up to the Master Agent to accept or ignore the offers.

After the subagents have registered with the Master Agent, they wait for the SNMP requests from the Master Agent. If a request is received, it sends the appropriate response. Additionally, the subagents can send SNMP traps.

The subagent is composed of four major components: the SNMP driver, Service API stack, subagent application, and MIB database. The Master Agent is responsible for receiving and sending the SNMP Protocol Data Units (PDUs). It also encodes and decodes the PDUs. The SP (Stack) dispatches the received requests appropriately or calls the applicable call back functions.

The system interface module is the implementation of the call back) of all the variables managed by the subagent. The MIB compiler automatically generates this information.

Except for the system dependent interface and the MIB database, the other components are reusable by other subagents and are provided in a library.

3.2.1 Establishment

This step involves the effort by the subagents to announce their presence and describe how to communicate with them. The transport used is UDP. The port on which the subagents listen for the SNMP request is configurable, as mentioned in "4.4 Agents Access Control File" on page 36 on "4.4 Agents Access Control File" on page 36. This establishment can also be done dynamically through a handshaking protocol.

3.2.2 Maintenance

The subagent periodically checks for existence of the Master Agent. The dynamic subagents (not invoked by the Master Agent) periodically determine whether the Master Agent is active. The runtime library reregisters the subagent's subtree with the Master Agent in the event of any interruptions.

3.2.3 Termination

Termination involves the effort to inform the Master Agent that the dynamic agent is about to exit. The Master Agent can then remove the row entry from its subagent table in the memory.

3.3 Using the Master Agent

`snmpdx [-h] [-p` *port_number*`] [-r` *filename*`]`

`[-a` *filename*`] [-c` *dirname*`] [-i` *filename*`] [-o` *filename*`] [-y]`

`[-m GROUP|SPLIT] [-d` *debug_level*`]`

The command line arguments are shown in Table 3–2.

TABLE 3–2 Master Agent Command Line Arguments

Argument	Description
`-a` *filename*	A full path (default) of the access control file is: `/etc/snmp/conf/snmdx.acl`; see "4.4 Agents Access Control File" on page 36 for more information

TABLE 3–2 Master Agent Command Line Arguments *(Continued)*

Argument	Description
`-c` *dirname*	The full path of the (default) directory containing the agent resource files; the default directory is: `/etc/snmp/conf`
`-d` *debug_level*	Used for debugging purposes; depending on the *debug_level* (0–4), it prints a specific amount of information; the default *debug_level* is 0
`-h`	Command usage
`-i` *filename*	The full path of the PID used by the Master Agent for recovery after a crash; it contains tuples of the UNIX process ID, port number, resource name, and agent name (default files) is: `/var/snmp/snmpdx.st`
`-o` *filename*	This file contains the tuple (*enterprise_name*, *oid*); for example, (`Sun Microsystems,1.3.1.6.1.4.32`); the file (default) is used as a base for lookup in the trap-filtering and forwarding process is: `/etc/snmp/conf/enterprises.oid`
`-m GROUP | SPLIT`	The forwarding of SNMP requests mode; the default is `GROUP`; see "3.1.5 Sending Requests" on page 27 for a description of the two modes
`-p` *port_number*	The port number; the default port number is 161; for example, `-p 1234`
`-r` *filename*	The full path of the resource file name used by the Master Agent; stores information about the subagents that the Master Agent invokes and manages; the default resource file is `/etc/snmp/conf/snmpdx.rsrc`; see "4.2 Agents Resource Configuration Files" on page 31 for more information
`-y`	A recovery indicator signals when invoking the Master Agent process and then invokes the recovery module; the recovery process discovers which subagents in the previous session are still active; those subagents not active are re-spawned by the Master Agent

CHAPTER 4

Configuring Enterprise Agents

4.1 Configuring Enterprise Agents Overview

The following files are considered for configuration:

- Master Agent resource configuration file
- Agents registration file
- Agents access control file
- Master Agent status file

4.2 Agents Resource Configuration Files

The Agents Resource Configuration files are used exclusively by the Master Agent. As soon as the Master Agent becomes active, it reads these files. These files store information for all those agents that the Master Agent may manage. Each entry in the configuration files also includes methods for invoking these subagents. Although a

subagent might not have a configuration file, the subagent may dynamically register with the Master Agent when it becomes active. More information about dynamic invocation and registration of subagents is described in “3.2 Description of the Subagent” on page 28.

Each agent may have its own resource configuration file when it opts for invocation by Master Agent and for static registration. This file contains information about the registration file associated with the subagent, in addition to the other information related to invoking the subagent. The agents registration file is described in “4.2 Agents Resource Configuration Files” on page 31.

The following example shows the grammar for the resource configuration files.

```
<ResourceFile> : Resource | Environment Resource
<Resource> : “resource” “=” “{“ ResourceList “}”
<ResourceList> : /*empty*/ | ResourceList ResourceItem
<ResourceItem> : “{“ StringList “}”
<Environment> : “environment” “=” “{“ EnvironmentList “}”
<EnvironmentList> : /*empty*/ | EnvironmentList
EnvironmentListItem
<EnvironmentListItem> : EnvironmentToken “=” Number
<EnvironmentToken> : “poll-interval” | “Max-agent-time-out”
<Number> : Integer
<StringList>: StringItem | StringList StringItem
<StringItem> : StringToken “=” QuotedString
<StringToken> : “registration_file” |“policy” | “command”
|’type”|”user”
<QuotedString> : “”” AlphanumericString ”””
```

The following example shows the `snmpdx.rsrc` and `mibiisa.rsrc` files.

Descriptions of the variables used in the configuration file follow the example. The comment lines begin with the # character.

```
snmpdx.rsrc
environment =
{
poll-interval = 5 # This is in seconds
max-agent-time_out = 10000000 # This is microseconds
}

mibiisa.rsrc
resource =
{
{
registration_file = /etc/snmp/conf/mibiisa.reg
security = “/etc/snmp/conf/snmpd.conf
type = “legacy”
policy = “spawn”
command = “/usr/lib/bin/mibiisa -p $PORT”
}
}
```

4.2.1 Environment Group

The environment group controls the behavior of the Master Agent. This group contains the following two variables:

- `poll-interval` - this field contains values in seconds and indicates that the Master Agent performs activities other than receiving/sending SNMP messages after the specified interval; it finds out if there is a change in the resource file (discovering if all the agents are responding) and performs other routine housekeeping activities
- `max-agent-time-out` - the value of this field is specified in microseconds; it signifies the maximum allowed time-out a subagent may request during registration; for example, when the Master Agent sends a request to a subagent, it waits for a time_out to receive the response; this time-out may be specified in the registration file or by using dynamic registration; if an agent sets this time-out too high, it creates problems for the Master Agent and other agents; to avoid such a problem, the Master Agent must specify a maximum value for the Master Agent to wait for a response from the subagent; this maximum value of time-out is specified using this variable.

4.2.2 Resource Group

The variables in the resource group are related only to subagents. The previous example configuration file contains two entries. Each entry represents a subagent and may have the following variables with some value assigned:

- `type` - this field has two values: `legacy` and `dynamic`.
- `registration_file` - this field specifies the registration configuration file for each subagent. The Master Agent reads the various entries in this file and creates appropriate entries in its MIB table. The details of this file are explained in "4.3 Agents Registration File" on page 34. This entry is mandatory for all legacy-type agents. If the value for this variable does not contain a full path, the executable checks the default directory `/etc/snmp/conf`.
- `policy` - this field has two values: `load` and `spawn`. The value `load` specifies the Master Agent to read this registration entry and create a row entry in its MIB table. If it finds the value `spawn`, the Master Agent invokes the respective subagent, as stated in the `command` field of that entry.
- `command` - this is the name of the subagent executable. The `command` may contain the full path, or if the full path is not mentioned, the executable checks the default directory `/usr/lib/bin`. The `command` may use a `$PORT` macro to provide the port number from which the subagent receives SNMP requests. `$PORT` is assigned a value by the Master Agent in the registration file of each subagent. The `$PORT` macro is necessary because the legacy agents or subagents may take different arguments for the port option (such as `-p`, `-n`, `-port`).
- `user` - this subagent is run according to the user specified in this entry.

4.3 Agents Registration File

Each agent has its own agent registration file. This provides the Master Agent and each subagent file with its own file version. The registration file contains information pertinent to each agent. It also includes the name of the agent, the subtree OIDs managed by the respective agent, request time out, and the preferred port number. The following example shows the format of each entry in this file.

```
<Config> : <Macro> <Agents>
<Macro> : "macros" "=" "{" <MacrosList>"}"
<MacrosList> : <MacrosList> <MacroItem> | empty
<MacroItem> : label "=" <SubidList>
<SubidsLis> : <SubidsList> "." <Subid> | <Subid>
<Subid> : "mib2" | "sun" | "enterprise" | identifier | number
<Agents> : "agents" "=" "{" <AgentList> "}"
<AgentList> : <AgentList> <AgentItem> | <AgentItem>
<AgentItem> : "{" <Name> <SubtreesTables> <TimeOut> <WatchDogTimer> <Port>"}"
<Name> : label "=" quotestring
<SubtreesTables> : <SubtreesTables> | <Subtrees> | <Tables>
<Subtrees> : "subtrees" "=" "{" <SubtreesList> "}"
<SubtreesList> : <SubtreesList> "," <SubtreeItem> | <SubtreeItem> | empty
<SubtreeItem> : <SubidsList>
<Tables> : "tables" "=" "{" <TableList>"}"
<TableList> : <TableList> <TableItem> | empty|
<TableItem> : "{" <Table> <Columns> <Indexs> "}"
<Table> : "table" "=" <SubidsList>
<Columns> : "column" "=" <Range>
<Range> : "[" number "]" | number
<Index>s : "indexs" "=" <Range>
<TimeOut> : "timeout" "=" number
<WatchDogTimer>: "watch-dog-time" "=" number
<Port> : "port" = number
```

The registration file name may have any extension. The `reg` extension is recommended. The following is an example of an actual subtree file.

```
macros = {
applicationTable = mib-2.27
sun = enterprise.42
}
agents = {
{
name = "ExampleAgent"
subtrees = { mib-2, sun }
tables = {
{ #begin table
table = applicationTable
columns = [ 2 -15 ]
indexes = [ 2 -3 ]
} #end table
} #end of tables
```

```
timeout = 20000 # Optional. Each SNMP request time out. This is
in microseconds.
watch_dog_time = 300 # This is in seconds
port = 4000 # Optional
}
} #end of agents
```

This configuration file consists of two groups of information:

- macros – contains macros for use in the agents group.
- agents – contains a number of variables and further groups *tabled* within its definition.

The variables used in the `agents` group are as follows:

- `name` - this variable names the subagent. Agent names must be unique, though multiple agents invoked as separate processes may have the same executable. The Master Agent uses the agent name as a key in the agent table MIB.
- `subtrees` - Subtrees contain a list of the subtree OIDs that are managed by this particular agent. A subagent may manage multiple trees. In the previous example of an actual subtree file in use, the agent named `ExampleAgent` manages subtrees `mib-2` and `sun`.
- `tables` - the subagent registration files may be configured to manage full or partial MIB tables. The tables group contains the `table` name, the `column` numbers, and particular row numbers (`indexes`). In the previous actual subtree file example, the `ExampleAgent` manages columns 2–15 of rows 2–3 of the application table in addition to subtrees `mib-2` and `sun`.
- `timeout` - the timeout variable is registered with the Master Agent. The Master Agent waits for the `timeout` microseconds to receive a response to its SNMP requests. Each agent specifies its own timeout, though this timeout may not be greater than the `max_agent_time_out` defined in the Master Agent resource configuration file previously defined.
- `watch_dog_time` - the Master Agent uses this timeout to determine if the subagent is active. The Master Agent polls the subagent only if there has been no activity between the Master Agent and the subagent for the `watch_dog_time` interval.
- `port` - the port is the number that the subagent is waiting for in order to receive SNMP requests from the Master Agent. This variable is optional. Normally, the subagents do not assign a value to this variable. If this variable is missing from the configuration file, the Master Agent attempts to find an unopened port, then invokes the respective subagent with this port number. If the port is assigned a value, the Master Agent invokes the subagent with this particular port number. The subagent is invoked according to the `command` variable mentioned in the Master Agent resource file.

Note – Dynamic agents developed using Solstice Enterprise Agents libraries do not need to be invoked by the Master Agent. For such agents, there may not be any entry in the Master Agent resource file. Such agents open any available port by themselves.

4.4 Agents Access Control File

The agents access control file is a configuration file that stores SNMP-related community information. Every subagent and the Master Agent may have its own access control file. This file name may have any extension, although the extension `acl` is recommended. This file must be stored in the `/etc/snmp/conf` directory.

The following is an example of the grammar for the access control configuration file.

```
<snmp_security> : <acls> <trap_block>
<acls> : /*empty*/ | "acl" "=" {<acls_list> }
<acls_list> : /*empty*/ | <acls_list> <acl_item>
<acl_item> : {<communities_stmt> <acl_access> <hosts> }
<communities_stmt> : "communities" "=" <communities_set>
<communities_set> : <communities_set> , <community_elem> |
<community_elem>
<community_elem>: alphanumeric_string
<acl_access> : "access" "=" <acl_access_type>
<acl_access_type> : read-only | read-write
<hosts> : "managers" "=" <hosts_list>
<hosts_list> : <hosts_list> , <host_item> | <host_item>
<host_item> : alphanumeric_string
<trap_block> : "trap" "=" { <traps_list> }
<traps_list> : /*empty*/ | <trap_list> < trap_item>
<trap_item> : { <trap_community_string> <trap_interest_hosts>
<enterprise_list> }
<trap_community_string> : "trap-community" "=" alphanumeric_string
<trap_interest_hosts_list> : <trap_interest_hosts_list> ,
<trap_interest_host_item> |
<trap_interest_host_item>
<trap_interest_host_item> : alphanumeric_string
<enterprise_list> : /*empty*/ | <enterprise_list> <enterprise_item>
<enterprise_item> : { <enterprise_stmt> <trap_number_stmt> }
<enterprise_stmt> : "enterprise" "=" quouted_alphanumeric_string
<trap_number_stmt> : "trap-num" "=" <trap_number_list>
<trap_number_list> : <trap_number_item>
<trap_number_item> : <trap_range>
<trap_range> : integer - integer | integer
```

The following is an example of the access control list file.

```
acl = {
{
communities = public, private
```

```
access = read-only
managers = hubble, snowbell, nanak
}
{
communities = jerry
access = read-write
managers = hubble, telescope
}
}
trap = {
  {
        trap-community = SNMP-trap
        hosts = hubble, snowbell
        { enterprise = "Sun"
          trap-num = 1, 2-5
        }
        {
          enterprise = "3Com"
          trap-num = 4 }
        }
        {
        trap-community = competitor-trap
        hosts = hp_server, ibm_server, sgi
        {
        enterprise = "sun"
        trap-num = 1,3 }
        {
        enterprise = "snmp"
        trap-num = 1-32
        }

    }

}
```

The access control list file contains the following two groups of configuration variables.

- `acl` - this group of variables consists of multiple triplets that include community names, access rights, and names of hosts from accepted SNMP requests (only if the requests include the configured communities). In the previous access control list file example, only `GET` and `GET_NEXT` SNMP requests that include public and private community names are accepted from hosts `hubble`, `snowbell`, and `nanak`. This group may contain multiple triplets.

  A Master Agent may have the appropriate communities and the access rights to receive SNMP PDUs. However, if the same SNMP PDU is forwarded to a subagent, it may reject the PDU that does not have the proper rights to receive such a PDU (or does not include the proper community). A subagent might have access rights and the community to receive an SNMP PDU, but such a PDU may never reach the subagent if the Master Agent does not have the appropriate community strings and the access rights.

- `trap` - this group of variables consists of information for sending and or /forwarding traps received from the subagents. The Master Agent uses the information specified in this group for forwarding the traps. This information specifies the names of the hosts to send the configured trap numbers. The trap PDUs contain the specified trap community. These traps are generated primarily by the subagents and are then sent to the Master Agent.

4.5 Master Agent Status File

The Master Agent status file contains information for the various subagents spawned by the Master Agent. The Master Agent uses this file exclusively. The Master Agent dynamically adds information to this file, and therefore you should not edit it manually. Whenever the Master Agent spawns a subagent process, it creates an entry in this file. The purpose of this file is for Master Agent recovery, in case the Master Agent dies or is killed. When the Master Agent restarts, the entries in this file indicate the subagent(s) it previously created and the corresponding port numbers. The Master Agent reads each entry in this file and compares it with the entries in the Master Agent resource configuration file. If the entry is not found in the resource file, the Master Agent kills that process. Whenever the entries are present in both files, the Master Agent attempts to access the subagent through the port.

4.5.1 MIB Issue

Three tables under Sun-specific enterprise MIB OID are defined that facilitate the Solstice Enterprise Agents technology. The following table example is meant to provide manageability of all subagents. The information in this table provides the identity of the subagent. This table contains the subagent name, and the subagent port number. An example of the subtree's OIDs managed by each subagent is not shown.

4.5.1.1 Sample MIB

The following is an example of a MIB containing all types of MIB variables. The MIB also includes tables. When this MIB is run through `mibcodegen`, it generates the appropriate MIB database and the stub code to build a subagent for this MIB.

```
DEMO-MIB DEFINITIONS ::= BEGIN

   IMPORTS
       OBJECT-TYPE, Counter32, Gauge32
         FROM SNMPv2-SMI
     DisplayString, TimeStamp
         FROM SNMPv2-TC;
```

```
   mib-2        OBJECT IDENTIFIER ::= { mgmt 1 }
   sun  OBJECT IDENTIFIER ::= { enterprises 42 }
   demo OBJECT IDENTIFIER ::= { sun 1000 }

--
-- Some objects
--
demoString OBJECT-TYPE
       SYNTAX DisplayString
       MAX-ACCESS read-write
       STATUS current
       DESCRIPTION
         "A read-write object of type String."
       ::= {demo 1}

demoInteger OBJECT-TYPE
       ::= {demoTable 1}
DemoEntry ::= SEQUENCE {
       demoEntryIndex
           INTEGER,
       demoEntryString
           DisplayString,
       demoEntryInteger
           INTEGER,
       demoEntryOid
           OBJECT IDENTIFIER }
SYNTAX INTEGER {
         up(1),
         down(2) }
       MAX-ACCESS read-write
       STATUS current
       DESCRIPTION
         "A read-write object of type Integer."
       ::= {demo 2}

   demoOid OBJECT-TYPE
       SYNTAX OBJECT IDENTIFIER
       MAX-ACCESS read-write
       STATUS current
       DESCRIPTION
         "A read-write object of type Oid."
       ::= {demo 3}
-- A table composed of some columns

   demoTable OBJECT-TYPE
       SYNTAX SEQUENCE OF DemoEntry
       MAX-ACCESS not-accessible
       STATUS current
       DESCRIPTION
           "A table."
       ::= {demo 10}

   demoEntry OBJECT-TYPE
       SYNTAX DemoEntry
```

```
       MAX-ACCESS not-accessible
       STATUS current
       DESCRIPTION
         "An entry in the table demoTable."
       INDEX {demoEntryIndex}
 demoEntryIndex OBJECT-TYPE
       SYNTAX INTEGER (1..2147483647)
       MAX-ACCESS not-accessible
       STATUS current
       DESCRIPTION
         "An index to uniquely identify the entry."
       ::= {demoEntry 1}

   demoEntryString OBJECT-TYPE
       SYNTAX DisplayString
       MAX-ACCESS read-write
       STATUS current
       DESCRIPTION
         "A read-write column of type String."
       ::= {demoEntry 2}

demoEntryInteger OBJECT-TYPE
       SYNTAX INTEGER {
         up(1),
         down(2) }

       MAX-ACCESS read-write
       STATUS current
       DESCRIPTION
         "A read-write column of type Integer."
       ::= {demoEntry 3}

   demoEntryOid OBJECT-TYPE
       SYNTAX OBJECT IDENTIFIER
       MAX-ACCESS read-write
       STATUS current
       DESCRIPTION
         "A read-write column of type Oid."
       ::= {demoEntry 4}
demoTrap TRAP-TYPE
        ENTERPRISE sun
        VARIABLES { demoInteger, demoString, demoOid}
        DESCRIPTION
        " Trap for testing."
        ::= 2
demoColdLinkTrap TRAP-TYPE
        ENTERPRISE snmp
        DESCRIPTION
        " Trap for testing."
        ::= 0

   END
```

CHAPTER 5

Using DMI

5.1 Using DMI Overview

This chapter provides an introduction to the Desktop Management Task Force (DMTF) special interest group, manageability, and the Desktop Management Interface (DMI).

5.2 What is DMTF?

The DMTF was formed in May of 1992 as a cooperative effort of eight companies: Digital Equipment Corporation™, Hewlett-Packard™, IBM™, Intel™, Microsoft™, Novell™, Sun™, and SynOptics™. The objective of the DMTF was to provide a simple solution for desktop manageability.

The DTMF created a standard interface that handles communication between any management application and all the manageable products on -- or attached to -- a desktop PC or server.

This standard is called the DMI. For more information on the DMI, refer to the *DMI Specification Version 2.0* at http://www.dmtf.org

The DMI is:

- Independent of any specific operating system, hardware platform, or management protocol
- Easy for vendors to adopt
- Scalable, to accommodate a wide range of products from very simple to very complex and extensible
- Mappable to existing management and remote protocols

5.3 DMI Functionality

DMI functionality for the SEA includes the following:

- Dynamic installation and removal of component instrumentations and management application
- Responsibility for all runtime access to the MIF data
- Assurance that at least one group (the component ID group) is in each MIF file
- Responsibility for launching the component instrumentation, if necessary
- Slicing commands. When a management application requests more than one attribute value from a component in a single command, the SP sends commands to the component instrumentation for each attribute
- Assurance that commands serialize to a component instrumentation and ensure that commands are allowed to run to completion. Multiple requests for a particular component instrumentation must be queued
- Provide event/indication subscription and filtering
- Forward indications based on subscription and filters to each registered management application and timestamp incoming indications before forwarding them
- Send indications to all registered management applications that subscribe to receive indications when components are installed or removed from the MIF database
- Appear to management applications as a component with ID 1 (one). As a component, it must support the standard ComponentID group. Additionally, the DMI SP must support the subscription and filter standard group. Also, similar to a component, it may define additional groups beyond the ComponentID group

5.4 Architecture of DMI

Customers may use the DMI-based solution included in the SEA product in several ways. For example, it may act as another SNMP subagent. Additionally, DMI-based management applications may be written to directly interact with SP.

In the SNMP subagent mode, the SNMP requests are mapped to DMI requests and are communicated with DMI SP. In the direct mode, the management applications may directly interact with the SP using DMI.

Figure 5–1 illustrates the overall architecture of how the DMI solution relates to the Enterprise Agents.

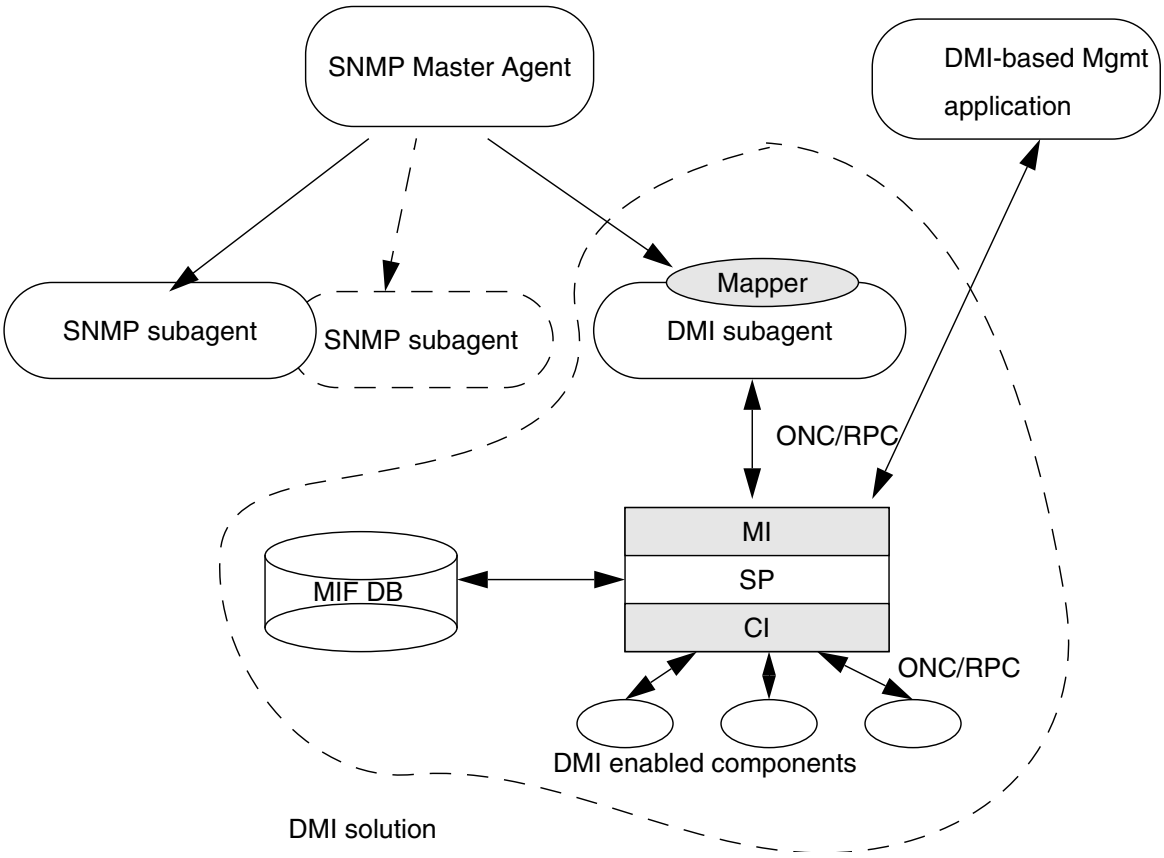

FIGURE 5–1 DMI and Enterprise Agents

5.4.1 DMI Service Provider

The DMI SP is the core of the DMI solution. Management applications and Component instrumentations communicate with each other through the SP. The SP coordinates and arbitrates requests from the management application to the specified component instrumentations. SP handles runtime management of the Component Interface (CI) and the Management Interface (MI), including component installation, registration at the MI and CI level, request serialization and synchronization, event handling for CI, and general flow control and housekeeping.

Figure 5–2 illustrates the elements that exist within a single system, or are directly attached. The management application may be used as a DMI browser.

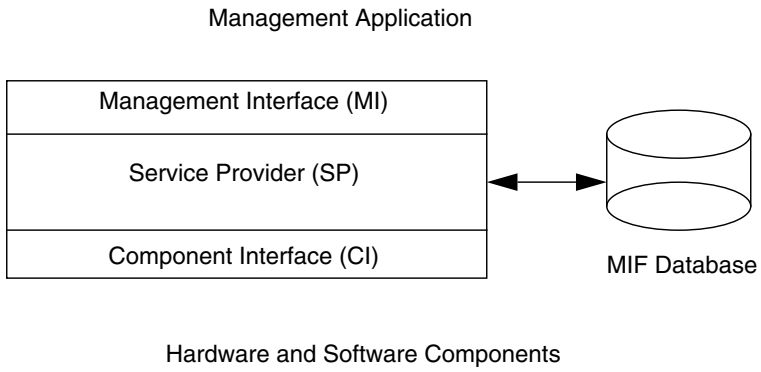

FIGURE 5–2 DMI Service Provider

The DMI SP consists of four modules:

- SP process
- MI (an interface used by the management application to communicate with DMI SP)
- Component Interface (the SP interface for component instrumentation code use)
- MIF database associated with the SP

5.4.1.1 Management Interface

The MI functionality for the SEA includes the following:

- A management application communicates with the DMI through the MI. A management application requests information about components in the system by issuing the DMI Get, Set, and List commands. It also registers with the SP for event notifications and proper filtering capabilities.

- When an event is generated, the DMI SP looks into its subscription and filtering table. DMI SP forwards the event in case it passes through the filtering process. The event is then forwarded to all management applications that have subscribed with the SP to receive such events.

5.4.1.2 Component Interface

The components communicate with the DMI through the CI. The components' subagents are created by end users to manage the respective components (devices/applications, etc.). The following functions are provided by the CI:

- Registration – Components register with the DMI SP.
- Sending Events – Component instrumentation sends an indication block to the SP for processing. The DMI mapper translates the DMI indication an SNMP trap.
- Components respond to the DMI SP requests to Get and Set various attributes instrumented by the component subagent.

5.4.1.3 MIF Database

The MIF database functionality for the SEA includes the following:

- There is one MIF database associated with the SP. Each component has a MIF file to describe its manageable characteristics. When the component is initially installed into the system, the MIF is added to the MIF database. SP controls all access to the MIF database.
- No MIF modification mechanism the to MIF database is provided. If the MIF needs to change, it must be uninstalled and modified using regular text editor tools, and then installed again.
- Upon installation and removal of a MIF from the MIF database, the SP must issue an indication to all registered management applications.

5.4.2 Invoking DMI Service Provider

After installing, a script file invokes the DMI SP at boot time if the `/etc/dmi/conf/dmispd.conf` configuration file contains neither blank lines nor comment lines. The DMI SP is invoked using the following options:

`dmispd [-h] [-c` *config_dir*`] [-d` *debug_level*`]`

TABLE 5–1 Invoking DMI SP Arguments

Argument	Definition
`-h`	**Displays the command line usage**
`-c` *confid_dir*	The full path of the directory containing the `dmispd.conf` configuration file
`-d` *debug_level*	In debug mode, the process does not run as a daemon and it displays trace messages on the display screen; depending on the `debug_level` (1-5), it prints a specific amount of information

5.5 DMI API Libraries

The DMI API library provided with the SEA package is a C library containing procedures for developing management applications using DMI. The library also provides a component interface for users to create subagents, including component instrumentation for the management of components. Additionally, DMI APIs simplify the processes of installing components in the MIF database and invoking the SP's component interface.

5.6 MIF-to-MIB Compiler

This utility translates DMI MIF files to SNMP format and generates a *map* configuration file. The Network Management application (Sun Net Manager, Enterprise Manager, etc.) may use the MIB to manage the DMI-based component MIF. The mapper process uses the map file. The map file helps in mapping SNMP-based MIB variables to the DMI-based MIF attributes.

`miftomib "[`*mifname*`=] [`*value value*` ...]" `*mifpathname* `[`*mibpathname*`]`

TABLE 5–2 MIF-to-MIB Compiler Arguments

Argument	Definition
mifname	The name of the `mib` object generated
value	One or more integers separated by a space
mifpathname	The `mif` file

TABLE 5–2 MIF-to-MIB Compiler Arguments *(Continued)*

mibpathname	The `mib` file generated

5.7 Mapper

The mapper is an SNMP subagent acting as a DMI management application. It sends management requests to the DMI SP using the management interface. It also processes events from the component through the SP and passes them on to the Master Agent. Thus, the DMI-enabled component looks like any other SNMP-managed component. Figure 5–3 illustrates the mapper and component communication.

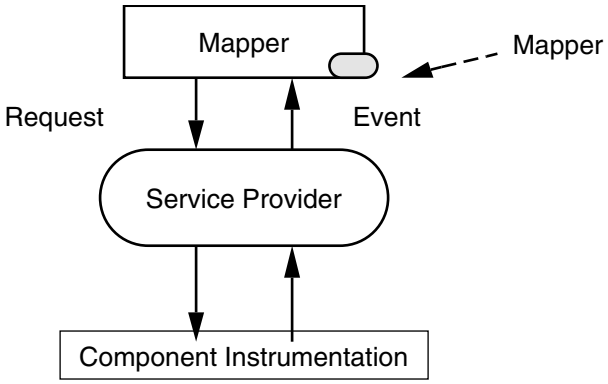

FIGURE 5–3 DMI Mapper and DMI Components

The following sections describe the subagent tasks during the life cycle.

5.7.1 Subagent Initialization/Reinstallation

- Registers itself with SP, so the SP may provide services, such as access to the MIF database and indications from components.
- Builds a translation table for mapping. This is done by reading the map files created for each component. The translation table is built when the mapper initially comes up. Later, when a new component is installed and registered with the SP, the DMI SP sends an event to the mapper. The mapper then adds the new map file entry into its mapping table.
- Establishes a connection between the Master Agent and subagent; it also registers itself with the Master Agent.

- Builds up the translation table used for SNMP/DMI mapping and translating SNMP OIDs to DMI objects dynamically.

5.7.2 Invoking the Mapper

The mapper is invoked using the following options:

`snmpXdmid -s` *hostname* `[-h] [-c` *config_dir*`] [-d` *debug_level*`]`

TABLE 5–3 Invoking DMI SP Arguments

Argument	Definition
`-h`	**Displays the command line usage**
`-s` *hostname*	The name of the host where the SP is running; by default it is the local host
`-c` *config_dir*	The full path of the directory containing the `snmpXdmid.conf` configuration file
`-d` *debug_level*	In debug mode, the process does not run as a daemon and it displays trace messages on the display screen; depending on the *debug_level* (1-5), it prints a specific amount of information

CHAPTER 6

Using SNMP With DMI

6.1 Using SNMP With DMI Overview

Solstice Enterprise Agents (SEA) technology enables management applications that communicate with SNMP to access DMI-enabled components through a DMI mapper called `snmpXdmid`. SNMP uses protocol data units to send information between management applications and agents distributed in the network. A standard MIB describes all objects that are managed by SNMP management applications. The agent programs supply or change the values of MIB objects, as requested by the management applications.

The DMI mapper `snmpXdmid` provides a mapping function that dynamically translates DMI information into the SNMP MIB format for communication with SNMP management applications. An SNMP management application may send requests to `snmpXdmid`, that in turn converts the SNMP requests into DMI requests. The conversion is performed again, in reverse, when the DMI response is returned. This enables the SNMP management application to participate in active management of DMI-enabled components.

This chapter describes how DMI and SNMP work together, and how to map data between SNMP MIB and DMI MIF.

Figure 6–1 illustrates how `snmpXdmid` works with other parts of the Solstice Enterprise Agents product.

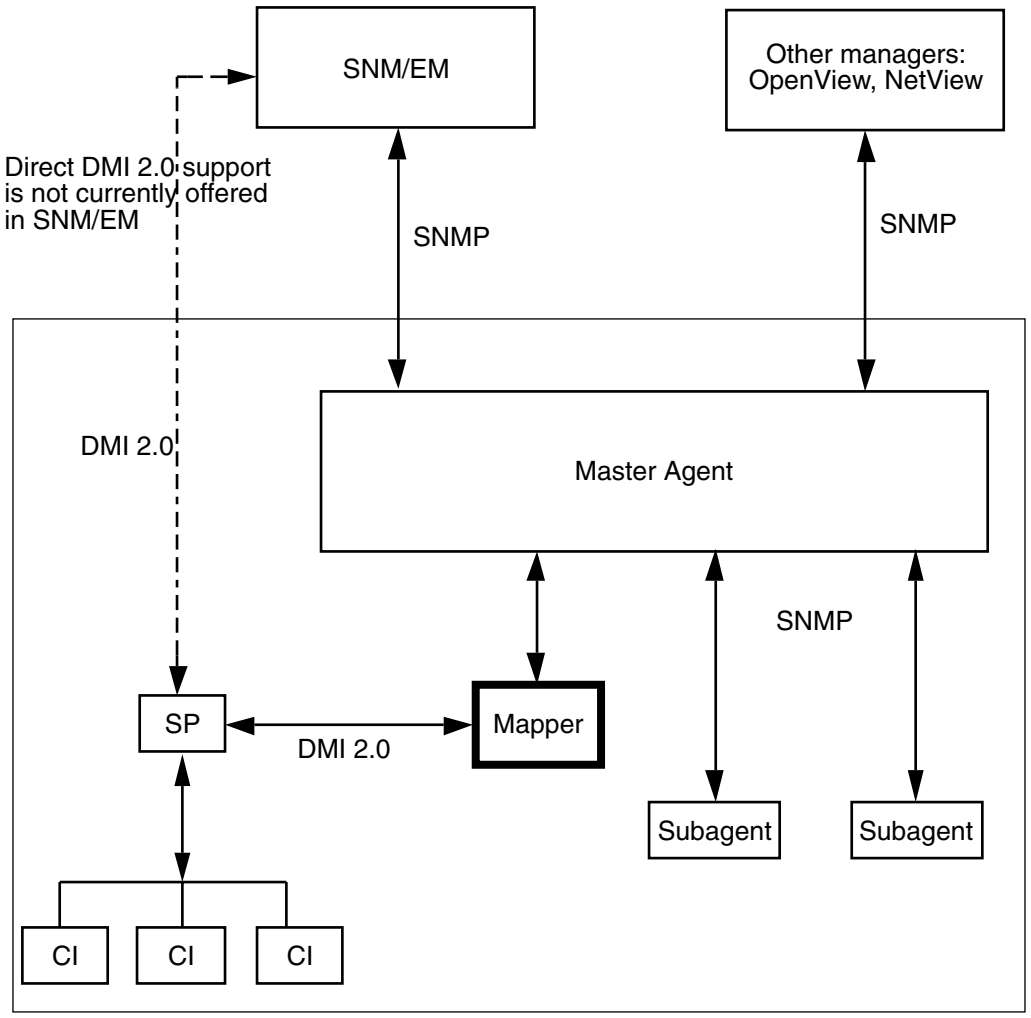

FIGURE 6–1 snmpXdmid Overview

6.2 SNMP Communication With the DMI

The communication path between an SNMP management application and the DMI consists of the SNMP Master Agent and the DMI mapper `snmpXdmid`, as described in the following sections.

6.2.1 SNMP Master Agent

The SNMP Master Agent handles SNMP requests and responses between management applications and registered subagents in your system. The `snmpd` executable file installed with Solstice Enterprise Agents replaces any `snmpd` file that may already be installed in your system.

The SNMP Master Agent communicates with subagents in the system through SNMP PDUs. A process acting as a subagent may register a MIB subtree with the SNMP agent. Any request received by the Master Agent for a registered variable is passed through `snmpdXdmid` to the subagent, then performs the request and returns a response to the Master Agent.

6.2.2 DMI Mapper

The DMI mapper handles requests from and responses to the SNMP Master Agent in the system. When it receives a request, such as an SNMP Get request for a specific MIB variable, the mapper translates the MIB variable into a corresponding DMI MIF attribute. Indications sent by DMI components in the system are then converted to SNMP traps and sent to the management application. Translation of MIB variables into MIF attributes and indications into traps are performed only for those components that the mapper has registered with the SNMP Master Agent.

6.3 Registering a Component With the SNMP Master Agent

To enable mapping of a component by `snmpXdmid`, the component's MIF file must correlate with an SNMP object identifier (OID) prefix. The OID prefix is a registration point that identifies the DMI component to the SNMP Master Agent.

The correlation of OID to component is performed by adding an entry for the component to the mapping file. The format of the `.MAP` file is explained in "6.6.3 MIF-to-MIB Mapping" on page 57. The `.MAP` files for the Solaris environment are located in the `/var/dmi/map` directory.

Note – For the DMI mapper to perform the mapping properly, make certain that the component name field in the mapping file entry corresponds exactly to the value of the name statement specified for the component in the MIF file. This includes such aspects as spacing and the character case in the string.

If any changes are made to the mapping file, one of the following actions must be taken for the DMI mapper to operate with current mapping information:

- If a new component is being registered after changes in the `.MAP` files, `snmpXdmid` re-reads the `.MAP` files automatically and no explicit action is required.
- If a component is not registered but only `.MAP` files are modified, then `snmpXdmid` must be stopped and restarted.

Note – If the same component name is repeated in the mapping file with different enterprise IDs, the DMI Mapper maps the component twice.

6.4 Running the DMI Mapper on Solaris

When running the DMI mapper `snmpXdmid` in a Solaris environment, the proper flow of SNMP information must be insured by:

- Adding an entry in the `snmpdx.acl` file for each host that receives the mapper traps.
- Making sure that the community name used by management applications for requests to the mapper allows write access. This is specified in the `snmpdx.acl` file. If you do not allow write access for the community name, Get and Set operations issued by management applications that use that community name will fail.
- Making sure the trap-ids from 0 through 16 are covered under Sun enterprise for traps in `snmpdx.acl`. This is handled automatically when `snmpXdmid` is installed on the system.

The `snmp.conf` file contains a description of the format for entries and provides instructions for adding entries. The `snmp.conf` file is located in the `/etc` directory.

6.5 How the DMI Mapper Works

The subagent registers the DMI components in the system with the SNMP Master Agent. Proper mapping on the part of the mapper requires that the MIF structure of a component be identified in a mapping file used by the subagent when it is processing requests from the SNMP agent. The mapping file contains a unique SNMP OID for the component. The OID is used as a registration point for the Master Agent.

To generate the mapping file, perform one of the following tasks:

- Create the file manually using a text editor.
- Use the `miftomib` utility to generate the mapping file and a corresponding SNMP MIB file for the component MIF.

The operational process of the subagent has the following states:

- Initialization
- Normal operation
- Exception reporting
- Termination

These states provide a general operational flow. The mapper `snmpXdmid` runs as a daemon. Normally, it waits for SNMP requests. On arrival of the request, it processes the request, returns the response, then continues to wait for further SNMP requests. This mapper also receives indications from the DMI SP. These are then forwarded by default as SNMP traps.

▼ 6.5.1 Initializing and Reinitializing the Subagent

The mapper normally starts at the system boot time through startup scripts. The mapper startup must be performed in the later stages of system startup. This enables the SNMP Master Agent and DMI (SP) to be initialized. Dynamic registration then takes place between `snmpXdmid` and both the Master Agent and the DMI SP.

1. **The mapper uses the Management Interface (MI) registration call to register itself with the Service Provider (SP) before any other management activity may be initiated.**

   This enables the SP to provide services (such as access to the MIF database). The mapper also adds an entry in the subscription table/filter table on the SP to receive indications.

2. **The mapper builds the translation table.**

   The translation table retains a Master Agent registration point for each unique pair of SNMP OID prefix and component names it finds in the `.MAP` files. When constructing these translation tables, the mapper searches all `.MAP` files in the `/var/dmi/map` directory.

   Identifiers for these components, along with group identifiers associated with each component, are retained for translation. Attributes of installed components not found in a `.MAP` file by the program are inaccessible by the SNMP agent. Whenever a component is installed or uninstalled, the translation tables are appropriately adjusted.

   The mapper re-reads all the map files when a new component is registered with the SP to insure that the translation tables are always current.

3. **The mapper connects to the Master Agent.**

Before the communication may take place for this connection, the mapper must establish a connection and register with the Master Agent.

Registration is achieved by using the following definitions:

- Subagent ID
- Agent status = ACTIVE
- Timeout value
- Subagent name
- Subagent address

4. **At this point, the agent registers the object ID for the MIB for which it is responsible.**

   The MIB OID prefixes that have been retrieved from the `.MAP` files are used for registration. OID prefixes associated with component names for which there are no installed components may also be registered.

   On initialization, if the DMI and its interfaces are functional, and the translation table is correct, a cold-start trap is sent to the agent. Refer to "6.6.1 Exception Reporting" on page 56 for more information on trapping.

   Reinitialization occurs upon detection of a severe or potentially severe error. This may be an error in communicating with either DMI or the Master Agent. On reinitializing DMI, Master Agent, and table initialization, a warm-start trap is sent to the Master Agent.

6.6 Converting SNMP Requests to DMI

When the SNMP Master Agent receives a request it determines whether the object is within the subtree registered the mapper. If it is, it then sends the mapper an SNMP request. Upon receiving a packet, the mapper parses the packet according to the following types:

- `GET`
- `GETNEXT`
- `SET`

Conversion of an SNMP request to DMI is shown in Table 6–1.

TABLE 6–1 SNMP Request to DMI Conversion

SNMP	Object	DMI	Data
{verb}	{noun}	{verb}	{noun}
get	OID	MI_cmds	ID and Key

TABLE 6–1 SNMP Request to DMI Conversion *(Continued)*

SNMP	Object	DMI	Data
getnext			comp ID/group ID/attr ID
set/commit/undo			

Access to the MIF database is enabled through a set of MI commands used to query for lists of components, groups or attributes, and to Get or Set the individual attributes.

When the mapper receives a GET request from the Master Agent, the mapper:

1. Determines whether the request is for the MIF or MIB, depending on whether it finds an entry in the translation table.
2. Parses the remainder of the OID, checking for validity.
3. Searches the translation table for presence of the specified group and component.
4. Translates the DMI table index (if specified) in the OID into a DMI key format.
5. Attempts to retrieve the object value from the DMI. The subagent then gives the object value returned to the agent.

In processing GETNEXT requests, the subagent must make certain that lexicographical ordering is maintained. The MIF database may undergo considerable searching before arriving at the attribute returned as an SMI object. If the value of the target attribute is unavailable because of a minor error (for example, specific to that attribute), a genErr prescribed by /RFC1448/ is returned. If the next object is not in the subagent's tree, the same object instance as in the request is returned and the value noSuchName is returned.

DMI attribute identifiers are encoded into unique OIDs. The INDEX clause is used to access DMI table objects and identifies a DMI table row. When DMI table attributes are examined during a GETNEXT, all row identifiers (key values) are examined to insure lexicographic order.

Handling of the SET request by the Master Agent is a bit more involved than the GET request. The normal command sequence is a GET followed by a SET command from the agent. In the event the agent may not successfully complete all the SETs for a given SNMP PDU, it sends the subagent another SET command with oi.d values obtained from the GET command. When it receives the SET command, the subagent makes some basic checks, such as confirming existence of the object and instance, correct value type, valid contents, and availability of memory needed for the operation. At this point, the call to the DMI with a DmiSetAttribute() to RESERVE the attributes is complete. The RESERVE is used to validate a SET without performing the SET. A RESERVE failure results in a genErr being returned to the Master Agent. Otherwise, the subagent performs DmiSetAttribute() with the SET option, to perform the actual set.

If another subagent has informed the SNMP agent it may not perform the SET of that PDU, the agent then gives a SET to this mapper, with older values.

6.6.1 Exception Reporting

The Master Agent enables the mapper to report traps to the agent by building a Master Agent trap PDU and sending it to the agent.

The trap is caused by an unsolicited message known as an INDICATION from the DMI SP. If these indications are from the component, they are called *events*. The mapper receives all indications after it has subscribed to the SP by adding an entry into its subscription table. The mapper also sets the filter conditions for receiving every type of event. Because the mapper is already waiting for requests from the agent, this routine is a separate thread. The subagent determines what OIDs are sent with the trap. The following list describes the different types of indications.

- DmiDeliverEvent This indication is generated by the component. The SP passes it to all the MI applications having their valid entries in the subscription and filter tables maintained by the SP. The snmpXdmid mapper matches the OID prefix to the component ID, generating the event for building the TrapOID. The groupId and attributes are also part of the event. The snmpXdmid mapper uses all relevant information in the event (apart from the OID prefix itself) to generate the TrapOID. The snmpXdmid generates a SNMP-specific trap, with specific trapID=14.
- DmiComponentAdded This indication is generated by the SP when a new component is registered. New components may be registered for existing .MAP files or for a totally new .MAP file. When this trap is received by snmpXdmid, it unregisters all the registered OIDs in the Master Agent, rereads all the .MAP files, and then goes through the process of reregistering all of the OIDs with the Master Agent. This helps to keep the subagent translation tables in sync with the .MAP files, especially since the .MAP files are generated external to the mapper. The mapper then generates an SNMP-specific trap with trapID = 7.
- DmiComponent-Deleted This indication is generated by the SP when an existing component ID is unregistered with the SP. The snmpXdmid mapper modifies the translation tables upon receiving this indication. By default, it generates an SNMP trap with trapID=8.
- DmiLanguageAdded This indication is generated by SP. It results in an SNMP trap with trapID=9.
- DmiLanguageDeleted This indication is generated by SP, resulting in an SNMP trap with trapID=10.
- DmiGroupAdded This indication is generated by SP when a new group is registered with it under a component ID. This results in the updating of translation tables in snmpXdmid. An SNMP trap is generated with trapID=11.
- DmiGroupDeleted This indication is generated by the SP when a group is unregistered with it. This results in the updating of translation tables, and an SNMP trap is generated with trapID=12.

- `DmiSubscriptionNotice` This indication is generated by the SP under two circumstances:
  - When the management application subscription for indication hits the warning timestamp. A flag indicates the warning aspect of the indication. An NMP trap with `trapID=15` is generated.
  - When the management application subscription for indication hits the expiration timestamp. An SNMP trap with `trapID=16` is generated.

6.6.2 Terminating the Subagent

Normally, the `snmpXdmid` daemon runs forever, unless explicitly killed or a disastrous situation is encountered (for example, lack of memory).

6.6.3 MIF-to-MIB Mapping

The mapping of MIF to MIB involves the assignment of an OID from the MIF. A translator (`miftomib.EXE`) is available to build a file extension of `.MAP` and a `.MIB` file from a MIF definition. The translator may be used as a tool to generate the `.MIB` and the `.MAP` files. Or, the map file may be generated manually, using a text editor. All `.MAP` files located in the directory under `/var/dmi/map` are scanned and included in the subagent translation table.

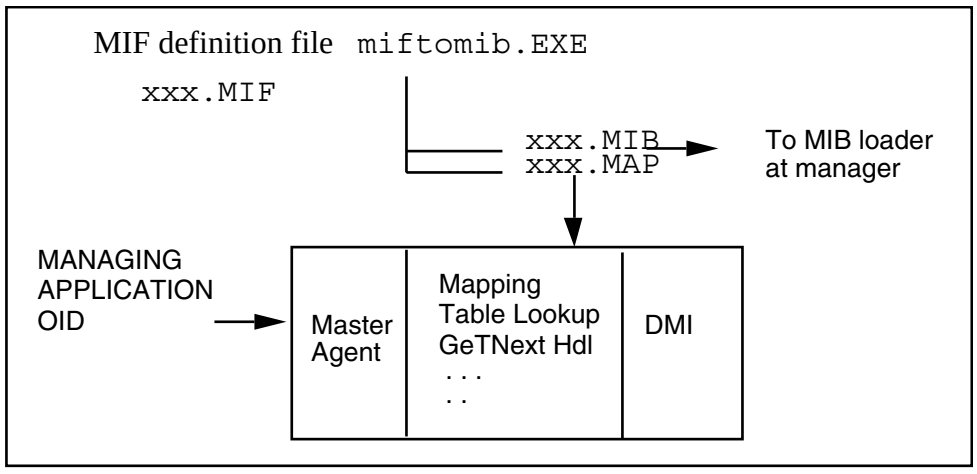

FIGURE 6–2 MIF-to-MIB Mapping

The map file format is shown in Table 6–2.

TABLE 6–2 Map File Format

OID Prefix	Component Name
"1.3.6.1.4.1.42.2000.2"	"Client Access/400 Family - Base Product"

The design approach is to map the DMI component ID, group ID, and attribute ID to MIB object IDs. The managing entity must predetermine the MIB definition to be used by the mapper to map to the MIF definition. Using a `miftomib` translator to produce the MIB mapping file facilitates the mapping.

The map file is generated externally to the mapper. A mechanism is required to notify the mapper of new or updated files on the system in order to enable the new MIF definitions to be dynamically added. This is accomplished by using the `DmiGroupAdded` indication, delivered by `dmispd`, when the mapper rereads all the `.MAP` files.

See Figure 6–3 for examples of the OID prefix and the completed OID layout. The mapping file is used by the mapper to register the OID with the SNMP agent relay and to correlate the OID with the component name. It contains the component ID and the group keys, in case the group contains tabular data.

The mapper puts no restriction on what the OID prefix may be. That is controlled by the network administrator. Any OID may be entered into a `.MAP` file, provided the SNMP Master Agent allows the subtree registration. In the case of failure, the `.MAP` file needs to be changed for the OID correction.

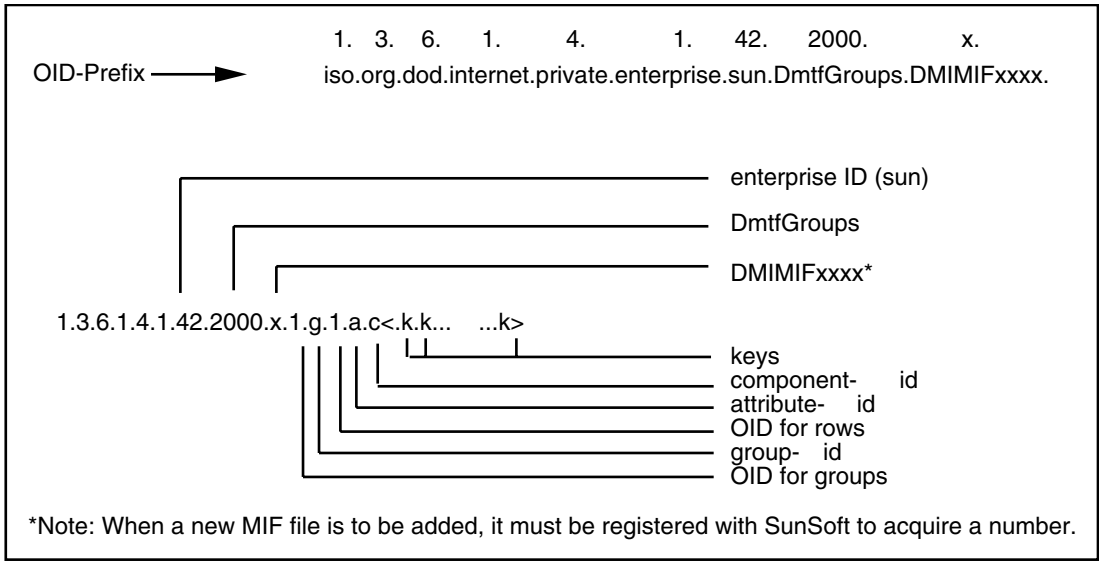

FIGURE 6–3 MIB OID Layout

Parts of the object identifier are shown below:

- Object Identifier: OID-prefix: `1.3.6.1.4.1.42.2000.x`
- OID for Groups: `.1`
- Object-type for Table: `.iGroupID`
- Object-type for Row: `.1`
- Attribute Identifier: `.iAttributeId`
- Instance Identifier: Indexes: `.iComponentId <keys>`

The example in Table 6–3 shows OIDs for client access attributes (no keys).

TABLE 6–3 OIDs for Client Access Attributes

Object Identifier: OID-prefix	OID for Groups	Object-type for Table	Object-type for Row	Attribute Identifier: Object-type for Column	Instance Identifier: Indexes
`1.3.6.1.4.1.42.2000.x`	`.1`	`.1`	`.1`	`.1`	`.3`

Retrieving objects via the `GetNext` operation must be specially handled, since MIB tables are traversed column-by-column and MIF tables are traversed row-by-row.

6.6.4 Specific Mapping Considerations

The DMI specification has reserved `ComponentId=1` for the DMI SP. The specification also defines the MIF file for the SP. The network administrator must take this into consideration when creating the `.MAP` files or when specifying the OID preference as a command-line parameter to the `miftomib` utility. Every MIF file must contain a standard group with ID 1.

TABLE 6–4 ComponentID Group Translated to DMI MIB

MIS Object Identifier and SYNTAX	MIF Data	Description	Notes
`DMIcompindex` INTEGER (1...217483647)	Component ID	Unique value for component	Assigned at installation by the SP, it communicates with this component until uninstalled; managing applications record this ID to request attributes later
`DMIcompManufacture` DisplayString (0...64)	Attribute "Manufacture"	Value assigned by the component provider	The name of the organization that produced this component

TABLE 6–4 ComponentID Group Translated to DMI MIB *(Continued)*

MIS Object Identifier and SYNTAX	MIF Data	Description	Notes
`DMIcompProduct` DisplayString (0...64)	Attribute "Product"	Value assigned by the component provider	Name of the component
`DMIcompVersion` DisplayString (0...64)	Attribute "Version"	Value assigned by the component provider	Version of the component
`DMIcompSerialnumber` DisplayString (0...64)	Attribute "Serial Number"	Value assigned by the component provider	Serial number of component
`DMIcompInstallation` Date	Attribute "Installation"	Value assigned by the SP at installation time	28-octet displayable string composed of date and time
`DMIcompVerify` Integer (0...7)	Attribute "Verify"	Verification level of this component at installation time	A request for this attribute causes the component to find out if it is still in the system and working properly

With this mapping, any MIF that is installed with the DMI has at minimum the `ComponentID` Group visible to management applications. All attributes within this group have read-only access. As MIFs become standard and are translated into MIBs, they become accessible but have additional anchor points in the `DmtfGroups` tree. For example, if the software MIF were defined, the translation would provide a `DMISW` MIB and be anchored as follows:

```
enterprise.sun.DmtfGroups.DMISW(2)
```

or

```
1.3.6.1.4.1.42.2000.3
```

Managing applications must be prepared for the same component to be visible in two different branches of the `DmtfGroups` tree.

Additional OID prefixes:

- `DMIHW (3)`
- `DMIPRINTER (4)`

6.7 The DMI Mapper Configuration File

The default configuration file `snmpXdmid.conf` is located in the directory `/etc/dmi/conf`. You may alternatively provide another path to this program as a command-line option for the location of the `snmpXdmid.conf` file.

6.7.1 WARNING_TIMESTAMP

`snmpXdmid` must subscribe to the DMI SP to receive indications. According to DMI 2.0 specifications, this subscription is valid only up to a particular timestamp. The SP issues a warning subscription notice before it ends the subscription. This timestamp indicates the time a subscription warning was issued.

The default value is:

```
WARNING_TIMESTAMP = 20101231110000.000000-420
```

6.7.2 EXPIRATION_TIMESTAMP

This indicates the time that a subscription actually expires. No indications are received after this time, unless they are re-subscribed. By default, a future timestamp is chosen so the subscription practically exists forever. This is done so that `snmpXdmid` always retains its indication subscription with the SP.

The default value is:

```
EXPIRATION_TIMESTAMP = 250101231120000.000000-420
```

6.7.3 FAILURE_THRESHOLD

This indicates the number of attempts by DMI SP in order to deliver the indications to `snmpXdmid`, in case it encounters errors before it drops the indication and clears the subscription entry.

The default value is:

```
FAILURE_THRESHOLD = 1
```

6.7.4 TRAP_FORWARD_TO_MAGENT

A non-zero value results in `snmpXdmid` generating an SNMP trap following an indication from the DMI SP. A zero value results in no SNMP trap generation following a DMI SP indication.

The default value is:

```
TRAP_FORWARD_TO_MAGENT = 1
```

6.8 Generating a MIB File

SNMP management applications typically require a MIB defining managed data. For each MIF file that you want to make accessible to an SNMP manager, you must generate an SNMP MIB that corresponds to the MIF file. The MIB must then be loaded at the management application that uses the definitions in the MIB to communicate with the DMI component. The management application may also make the MIB information available to browsers and other MIB-based applications.

To generate an SNMP MIB from a MIF file, use the `miftomib` utility at the command prompt. After you create the MIB file, you may transfer it to the SNMP management application.

CHAPTER 7

DMI Command Line Utilities

7.1 DMI Command Line Utilities Overview

The Desktop Management Interface(DMI) Protocol command line interface consists of two utilities that retrieve DMI Service Provider (SP) information:

- `dmi_cmd`
- `dmiget`

7.1.1 The dmi_cmd Utility

The `dmi_cmd` utility provides the ability to:

- Obtain version information about the DMI SP
- Set the configuration to describe the language required by the management application
- Obtain the configuration information describing the current language in use for the session
- Install components into the database
- List components in a system to determine what is installed
- Delete an existing component from the database
- Install group schemas to an existing component in the database

- List class names for all groups in a component
- List the groups within a component
- Delete a group from a component
- Install a language schema for an existing component in the database
- List the set of language mappings installed for a specified component
- Delete a specific language mapping for a component
- List properties for one or more attributes in a group

7.1.2 The dmiget Utility

The `dmiget` utility retrieves the table information of a specific component in the DMI SP.

7.2 Using the dmi_cmd Command

```
dmi_cmd [-s hostname]
-h |
-V |
-W config |
-X |
-CI mif_filename |
-CL [-c compId] [-r reqMode] [-d] [-p] [-m maxCount] |
-CD -c compId |
-GI schema_filename -c compId |
-GL -c compId -g groupId [-r reqMode] [-d] [-p] [-m maxCount] |
-GM -c compId [-m maxCount] |
-GD -c compId -g groupId |
-NI schema_filename -c compId |
-NL -c compId |
-ND -c compId -l language_string |
-AL -c compId -g groupId [-a attrId] [-r reqMode] [-d] [-p]

[-m maxCount]
```

The command line arguments for `dmi_cmd` are shown in Table 7–1.

TABLE 7–1 dmi_cmd Command Line Arguments

Argument	Description
`-s` *hostname*	Specifies the host machine that `dmisp` is running on; the default host is the local host
`-h`	Prints usage information

TABLE 7–1 dmi_cmd Command Line Arguments *(Continued)*

Argument	Description
`-V`	Retrieves version information about the DMI SP
`-W` *config*	Sets the configuration specified in *config* to `dmisp`
`-X`	Retrieves configuration information describing the current language in use
`-CI` *mif_filename*	Installs the component specified in *mif_filename*
`-CL` *[-c compId] [-r reqMode] [-d] [-p] [-m maxCount]*	Lists components
`-CD` *-c compId*	Deletes the component specified by *compId*
`-GI` *schema_filename -c compId*	Installs the group schema specified in *schema_filename*
`-GL` *-c compId -g groupId [-r reqMode] [-d] [-p] [-m maxCount]*	Lists the groups for a specified component
`-GM` *-c compId [-m maxCount]*	Lists class names for a specified component
`-GD` *-c compId -g groupId*	Deletes specified groups for a specified component
`-NI` *schema_filename -c compId*	Installs the language schema specified in *schema_filename*
`-NL` *-c compId*	Lists language mappings for a specified component
`-ND` *-c compId -l language_string*	Deletes language mapping for a specified component
`-AL` *-c compId -g groupId [-a attrId] [-r reqMode] [-d] [-p] [-m maxCount]*	Lists attributes for a specified component

Note – The values for *compId*, *groupId*, *attrId*, and *maxCount* are positive integers. The default value is 0.

The valid integer values for *reqMode* are:

- `1 (DMI_UNIQUE)`
- `2 (DMI_FIRST)`
- `3 (DMI_NEXT)`

Note – The default value for *reqMode* is 1 (`DMI_UNIQUE`). If an invalid *reqMode* is specified, the default value is used.

When listing components (`-CL`), groups (`-GL`), or attributes (`-AL`), use the `-d` option to display descriptions and the `-p` option to display the pragma string.

7.2.1 dmi_cmd Examples

7.2.1.1 Example 1

The following command lists the component ID, name, and description of up to 5 components, starting from component 3 in the `dmispd` running on the host `snowbell`, by using the default request mode (`DMI_UNIQUE`).

```
%dmi_cmd -s snowbell -CL -d -c 3 -m 5

Connecting to dmispd on the snowbell...

CompId:         4
Comp Name:      DMTF Developers -  Direct Interface Version
Description:    A list of the people who actually wrote the code.

CompId:         5
Comp Name:      DMTF Developers -  Direct Interface Version
Description:    A list of the people who actually wrote the code.

CompId:         6
Comp Name:      DMTF Developers -  Direct Interface Version
Description:    A list of the people who actually wrote the code.

CompId:         7
Comp Name:      DMTF Developers -  Direct Interface Version
Description:    A list of the people who actually wrote the code.

CompId:         8
Comp Name:      DMTF Developers -  Direct Interface Version
Description:    A list of the people who actually wrote the code.
```

7.2.1.2 Example 2

The following command lists the ID, name, storage, access mode, date type, and maximum size of all attributes in group 1 of component 1 in the `dmispd` running on local host, by using default request mode (`DMI_UNIQUE`). No description is displayed, and no limitation is set for max count.

```
%dmi_cmd -AL -g 1 -c 1

Connecting to dmispd on the localhost...

12 attrs listed for group 1 of comp 1

Attr Id:        1
Name:           Manufacturer
Storage:        MIF_COMMON
Access:         MIF_READ_ONLY
Type:           MIF_DISPLAYSTRING
maxSize:        64

Attr Id:        2
Name:           Product
Storage:        MIF_COMMON
Access:         MIF_READ_ONLY
Type:           MIF_DISPLAYSTRING
maxSize:        64

Attr Id:        3
Name:           Version
Storage:        MIF_COMMON
Access:         MIF_READ_ONLY
Type:           MIF_DISPLAYSTRING
maxSize:        64

Attr Id:        4
Name:           Serial Number
Storage:        MIF_SPECIFIC
Access:         MIF_READ_ONLY
Type:           MIF_DISPLAYSTRING
maxSize:        64

Attr Id:        5
Name:           Installation
Storage:        MIF_SPECIFIC
Access:         MIF_READ_ONLY
Type:           MIF_DATE
maxSize:        0

Attr Id:        6
Name:           Verify
Storage:        MIF_SPECIFIC
Access:         MIF_READ_ONLY
Type:           MIF_INTEGER
maxSize:        0

Attr Id:        7
Name:           ComponentId
Storage:        MIF_SPECIFIC
Access:         MIF_READ_ONLY
Type:           MIF_INTEGER
maxSize:        0
```

```
Attr Id:        8
Name:           ComponentName
Storage:        MIF_SPECIFIC
Access:         MIF_READ_ONLY
Type:           MIF_DISPLAYSTRING
maxSize:        256

Attr Id:        9
Name:           ComponentDesc
Storage:        MIF_SPECIFIC
Access:         MIF_READ_ONLY
Type:           MIF_DISPLAYSTRING
maxSize:        256

Attr Id:        10
Name:           GroupId
Storage:        MIF_SPECIFIC
Access:         MIF_READ_ONLY
Type:           MIF_INTEGER
maxSize:        0

Attr Id:        11
Name:           GroupName
Storage:        MIF_SPECIFIC
Access:         MIF_READ_ONLY
Type:           MIF_DISPLAYSTRING
maxSize:        256

Attr Id:        12
Name:           LanguageName
Storage:        MIF_SPECIFIC
Access:         MIF_READ_ONLY
Type:           MIF_DISPLAYSTRING
maxSize:        256
```

7.2.1.3 Example 3

The following command installs `namedir.mif` in `dmispd` running on `localhost`. The file `namedir.mif` is located in the directory specified in the configuration file.

```
%dmi_cmd -CI namedir.mif

Connecting to dmispd on the localhost...

"namedir.mif" is installed as comp 21.
```

7.2.1.4 Example 4

The following command uninstalls component 5 in `dmispd` running on `localhost`.

```
%dmi_cmd -CD -c 5

Connecting to dmispd on the localhost...
```

```
comp 5 is uninstalled.
```

7.2.1.5 Example 5

The following command displays the version of `dmispd` running on the `snowbell` machine.

```
%dmi_cmd -s snowbell -V

Connecting to dmispd on the snowbell...

dmispd version: Dmi2.0
description:    This is a DMI2.0 based on ONC RPC
```

7.3 Using the dmiget Command

`dmiget [-s` *hostname*`]`

`-h |`

`{-c` *compId* `[-g` *groupId*`] [-a` *attrId*`]}`

The command line arguments for `dmiget` are described in Table 7–2.

TABLE 7–2 dmiget Command Line Arguments

Argument	Description
`-s` *hostname*	Specifies the host machine that `dmisp` is running on; the default host is the local host
`-h`	Prints usage information
`-c` *compId*	Displays all the table information for a specified component
`-g` *groupId*	Displays the group information for a component specified with the `-c` argument
`-a` *attrId*	Displays the attributes for the component specified with the `-c` argument

7.3.1 dmiget Examples

7.3.1.1 Example 1

The following command displays the table information in group 2 of component 3.

```
%dmiget -c 3 -g 2

Connecting to dmispd on the localhost...

For group 2 of component 3:

Id: 10,        10
Id: 20,        developer1
Id: 30,        SunSoft
Id: 40,        Solaris 2.6

Id: 10,        20
Id: 20,        developer2
Id: 30,        SunSoft
Id: 40,        Solaris 2.6

Id: 10,        30
Id: 20,        developer3
Id: 30,        SunSoft
Id: 40,        Solaris 2.6
```

7.3.1.2 Example 2

The following command displays the table information for component 3.

```
%dmiget -c 3

Connecting to dmispd on the localhost...

For group 1 of component 3:

Id: 1,        SunSoft
Id: 2,        DMTF Demonstration
Id: 3,        Version 1.0
Id: 4,        1.00000
Id: 5,        1994 06 03 09 00 00
Id: 6,        0
Id: 7,        0
Id: 8,
Id: 9,
Id: 10,        0
Id: 11,
Id: 12,

For group 2 of component 3:

Id: 10,        10
Id: 20,        developer1
Id: 30,        SunSoft
Id: 40,        Solaris 2.6

Id: 10,        20
Id: 20,        developer2
```

```
Id: 30,        SunSoft
Id: 40,        Solaris 2.6

Id: 10,        30
Id: 20,        developer3
Id: 30,        SunSoft
Id: 40,        Solaris 2.6

For group 42 of component 3:

Id: 1,        Circus
Id: 2,        4.0a

Id: 1,        Disk Blaster
Id: 2,        2.0c

Id: 1,        Oleo
Id: 2,        3.0

Id: 1,        Presenter
Id: 2,        1.2
```

7.3.1.3 Example 3

The following command displays table information for attribute 20 in group 2 of component 3.

```
%dmiget -c 3 -g 2 -a 20 -s snowbell

Connecting to dmispd on the snowbell...

For group 2 of component 3:

Id: 20,            developer1
Id: 20,            developer2
Id: 20,            developer3
```

APPENDIX **A**

Error Messages

A.1 Messages by Name

Some of the error messages for Solstice Enterprise Agents (SEA) are listed in this section. They are worded to be fairly intuitive and easy to understand. Also, to make them easier to find, the messages are presented here in alphabetical order. Their number follows, to the right of the listing.

To see the messages listed by number, see Section A.2 "Messages by Number".

ADD Messages

ADD_XLATE_outOfMemory	738

AR CONNECT Messages

AR_CONNECT_AND_OPEN_connectFailed	102
AR_CONNECT_AND_OPEN_getHostNameFailed	101
AR_CONNECT_AND_OPEN_openFailed	103

All error messages are logged into the `/var/adm/messages` Syslog facility.

FIND Messages

GET Messages

ISSUE Messages

KEY Messages

A.2 Messages by Number

```
171     UNREG_BY_AR_failed
211     GET_PARSE_INSTANCE_notFullySpecified
219     GET_PARSE_INSTANCE_programmingError
221     GET_FIND_GC_PAIR_pairNotFound_groupMatch
222     GET_FIND_GC_PAIR_pairNotFound_noGroupMatch
241     GET_ATTRIBUTE_cnfBufAddressability
244     GET_ATTRIBUTE_tooBig
245     GET_ATTRIBUTE_genErr
246     GET_ATTRIBUTE_dmiBroke
247     GET_ATTRIBUTE_arBroke
251     GET_COMP_MIB_noSuchInstance
252     GET_COMP_MIB_noSuchInstanceOrObject
253     GET_COMP_MIB_noSuchObject
254     GET_COMP_MIB_tooBig
255     GET_COMP_MIB_genErr
256     GET_COMP_MIB_dmiBroke
257     GET_COMP_MIB_arBroke
321     GN_PARSE_INSTANCE_badOid
341     GN_FIND_GROUP_foundNoChange
342     GN_FIND_GROUP_notFound
363     GN_FIND_COMPONENT_notFound
371     GN_FIND_KEY_noKeyExists
372     GN_FIND_KEY_notFound
385     GN_GET_OBJECT_genErr
386     GN_GET_OBJECT_dmiBroke
388     GN_GET_OBJECT_outOfMemory
401     GET_NEXT_COMP_MIB_tooBig
405     GET_NEXT_COMP_MIB_genErr
408     GET_NEXT_COMP_MIB_outOfMemory
411     GET_KEY_VALUE_noneHigher
```

412 GET_KEY_VALUE_otherError

418 GET_KEY_VALUE_outOfMemory

421 TRANSLATE_DMI_KEY_TO_INTERNAL_otherError

428 TRANSLATE_DMI_KEY_TO_INTERNAL_outOfMemory

431 GET_DMI_KEY_SIZE_badAttributeType

441 GET_INTERNAL_KEY_SIZE_otherError

552 SET_ATTRIBUTE_badPacketType

555 SET_ATTRIBUTE_genErr

556 SET_ATTRIBUTE_dmiBroke

571 SET_CHECK_ILLEGAL_gotError

575 SET_CHECK_ILLEGAL_genErr

576 SET_CHECK_ILLEGAL_dmiBroke

581 SET_COMP_MIB_commitFailed

583 SET_COMP_MIB_noAccess

584 SET_COMP_MIB_noCreation

591 SET_COMP_MIB_notWritable

592 SET_COMP_MIB_wrongType

593 SET_COMP_MIB_wrongLength

594 SET_COMP_MIB_wrongValue

601 SET_COMP_MIB_resourceUnavailable

602 SET_COMP_MIB_badPacketType

605 SET_COMP_MIB_genErr

606 SET_COMP_MIB_dmiBroke

618 SET_ADD_TO_RESERVE_LIST_outOfMemory

711 CREAT_BUFFER_invalidOid

718 CREAT_BUFFER_outOfMemory

721 EXTRACT_OID_invalidOid

728 EXTRACT_OID_outOfMemory

738 ADD_XLATE_outOfMemory

756 CREATE_COMP_LIST_dmiBroke

```
758       CREATE_COMP_LIST_outOfMemory
766       CREATE_GC_PAIRS_dmiBroke
768       CREATE_GC_PAIRS_outOfMemory
776       GET_GC_PAIRS_dmiBroke
778       GET_GC_PAIRS_outOfMemory
821       PREPARE_FOR_DMI_INVOKE_resetSemFailed
833       DMI_REGISTER_badCmdHandle
841       DMI_UNREGISTER_failed
848       DMI_UNREGISTER_outOfMemory
871       KEY_FROM_AR_tooShort
872       KEY_FROM_AR_tooLong
873       KEY_FROM_AR_illegalKey
874       KEY_FROM_AR_otherError
878       KEY_FROM_AR_outOfMemory
891       XLATE_TYPE_noError_opaqueType
892       XLATE_TYPE_unexpectedType
893       XLATE_TYPE_mifUnknownType
894       XLATE_TYPE_otherError
900       ISSUE_LIST_COMP_failed
901       ISSUE_LIST_COMP_badLevelCheck
902       ISSUE_LIST_COMP_badCmdHandle
908       ISSUE_LIST_COMP_outOfMemory
910       ISSUE_LIST_GROUP_failed
911       ISSUE_LIST_GROUP_badLevelCheck
912       ISSUE_LIST_GROUP_badCmdHandle
918       ISSUE_LIST_GROUP_outOfMemory
920       ISSUE_LIST_ATTRIBUTE_failed
921       ISSUE_LIST_ATTRIBUTE_badLevelCheck
922       ISSUE_LIST_ATTRIBUTE_badCmdHandle
928       ISSUE_LIST_ATTRIBUTE_outOfMemory
```

930 ISSUE_LIST_DESCRIPTION_failed

931 ISSUE_LIST_DESCRIPTION_badLevelCheck

932 ISSUE_LIST_DESCRIPTION_badCmdHandle

938 ISSUE_LIST_DESCRIPTION_outOfMemory

940 ISSUE_GET_ATTRIBUTE_failed

941 ISSUE_GET_ATTRIBUTE_badLevelCheck

942 ISSUE_GET_ATTRIBUTE_badCmdHandle

948 ISSUE_GET_ATTRIBUTE_outOfMemory

950 ISSUE_GET_ROW_failed

951 ISSUE_GET_ROW_badLevelCheck

952 ISSUE_GET_ROW_badCmdHandle

958 ISSUE_GET_ROW_outOfMemory

960 BUILD_SET_ATTRIBUTE_failed

968 BUILD_SET_ATTRIBUTE_outOfMemory

970 ISSUE_SET_ATTRIBUTE_failed

971 ISSUE_SET_ATTRIBUTE_badLevelCheck

972 ISSUE_SET_ATTRIBUTE_badCmdHandle

978 ISSUE_SET_ATTRIBUTE_outOfMemory

981 FIND_ATTRIBUTE_ACCESS_notFound

982 FIND_ATTRIBUTE_ACCESS_otherError

986 FIND_ATTRIBUTE_ACCESS_dmiBroke

991 FIND_ATTRIBUTE_TYPE_notFound

992 FIND_ATTRIBUTE_TYPE_otherError

996 FIND_ATTRIBUTE_TYPE_dmiBroke

1008 TRACE_KEY_outOfMemory

Glossary

This glossary offers brief descriptions of terms that appear in the discussion of the Solstice Enterprise Agents documentation, either because they are used in the industry or because they have specific meanings in the Solstice environment.

agent Also called Network Management Agent. A module residing in a managed resource on a network, capable of reporting the status of the resource and/or responding to inquiries about it. Described in standards documents X.701 | ISO/IEC 10040. In a general sense, software running on a managed object that responds to and reports to the management application with current information about the object. See also *Proxy Agent*.

agent/subagent SDK The Software Development Kit has multiple components. It includes *agent/subagent* libraries, a MIB compiler, and sample subagents.

API Application Programming Interface. An API is a set of software routines that enables an applications developer to access and use the features of a product.

ARP Address Resolution Protocol. A procedure for finding the network hardware address corresponding to an internet address (RFC 826).

ASN.1 Abstract Syntax Notation One. A specification understood by network management protocols and used for encoding information between a manager and agents in a machine and network-independent manner.

attribute An attribute is the building block of MIF. An attribute describes a single characteristic of a manageable product, or component. For example, the clock speed of a processor chip is an attribute of that chip. A set of related attributes constitutes a MIF group.

child A subordinate object contained in an instance of a class and directly below that class instance. [C]

CI Component Interface. Describes access to management information and enables a component to be managed.

class The formal description of a set of objects. In the OSI world, objects with similar *attributes* and *behavior* are grouped into classes. In C++, the rules governing a set of data structures (that are said to be *instances* of the class) and the *methods* (also called *member functions*) that give access to an instance's data.

class instance A collection of attribute instance values that specifies one example of a class. For example, if the class comprised port information for a router port, you could specify an instance of the class by providing a router board and port number for a particular port. The information you provide to specify a class instance is called the *instance identifier*. Other related terms are instance string, Relative Distinguished Name (RDN), Index, and Named Object.

common group A MIF group that has been proposed to and accepted by the DMTF special interest group and that describes common attributes applicable to all, or most, manageable products. Examples of common groups include Field Replaceable Unit (FRU) and Operational State.

component Any hardware or software product that is part of or attached to a desktop system or server. For example, a modem, a printer, a network interface card, a spreadsheet program, and an operating system could all be considered components.

dispatching The communication of a management request from the Master Agent to one or more *subagents*. Dispatching is performed according to the Master Agent's current view of registered *subtrees* and an explicitly stated algorithm.

DCE Distributed Computing Environment. This is provided by OSF (Open System Foundation) DCE allows development of applications based on client-server architecture.

DMI Desktop Management Interface. The Desktop Management Interface is a set of interfaces and a service provider that mediate between management applications and components residing in a system. The DMI is a free-standing interface that is not tied to any particular operating system or management process.

DMTF Data Management Task Force. The Desktop Management Task Force was formed in May of 1992 as a cooperative effort of eight companies: Digital Equipment Corporation, Hewlett-Packard, IBM, Intel, Microsoft, Novell, SunSoft and SynOptics.

duplicate registration An attempt by one subagent to *register* a subtree that exactly matches a subtree already registered by another *subagent*.

entities Systems, components, and applications.

enumerations Enumerations are lists of possible values for a given attribute. They may be global or local. Global enumerations named may be used by

other attributes within a component. A local enumeration is unnamed and may only be used by the attribute containing it.

events Events consist of unsolicited information sent from a component to the Service Provider detailing an unusual circumstance or notable event. Events trigger indications from the Service Provider to management applications. Events may be sent, for instance, when an error occurs or when a new version of a piece of software is installed. Component manufacturers determine that events will be related to their product and what information will be passed about the event.

filter The use of a Boolean expression to test a set of attributes in order to select the objects where a network management command is addressed. Object instances that successfully pass the filtering tests become those where a management operation is performed. Defined by the CMIS specification (ISO/IEC 9595), filtering capabilities help reduce the network traffic overhead of a management protocol. See also *scoping*. This usage of "filter" is distinct from the usage in UNIX systems, where a filter is a program that accepts input from one stream and supplies output in another, so that it may be piped to other functions as needed. See Network Management Forum.

gateway A computer that interconnect two networks and routes packets from one to the other. A gateway has more than one network interface.

group A group is a set of related attributes for a given component. The DMTF group has standardized MIFs at the group level as well as at the component level.

indications Indications are information sent from the Service Provider to management applications when an event is received by the Service Provider or when a component is installed or removed from the MIF database. Indications triggered by events include information about the event and the component sending the event.

instance In C++, a piece of data whose structure is described by its membership in a class. Access to the data is provided only by the member functions defined by the class. For managed objects, a specific case or example of a managed object. For example, routers might be taken as an object class; one particular router would be an instance of that class.

instrumentation Instrumentation is the general name for programs that provide the values for attributes in the MIF database. Instrumentation is provided in two ways: by runtime programs (programs that are run by the Service Provider to retrieve or set the value at the time the action is requested by a management application) and by direct interface (programs that are always running and linked into the Service Provider to provide the value on request).

Internet A large collection of connected networks, primarily in the United States, running the Internet suite of protocols. The generic term "internet" refers to a collection of TCP/IP internetworks.

interoperability The capability of two or more systems to meet user requirements by communicating through specific mechanisms in a known environment.

IETF Internet Engineering Task Force. Source of *MIB*, SNMP.

IP address A 32-bit quantity used to represent a point of attachment in a TCP/IP-based Internet.

ISO International Standards Organization. Develops standards, by international agreement, over a wide range of technical areas.

keys A key attribute is the attribute used to find a specific row in a table of attributes when there is more than one instance of a set of attributes in a particular group. For instance, a computer system often has more than one serial port attached to it. To describe these serial ports, the Serial Port Group in the system's MIF file would be set up as a table, with one row in the table describing the specifics of a particular serial port. To access this information, one or more of the attributes (such as the I/O address) would be designated as the key. To find a specific serial port, the management application would ask for the row containing the proper I/O address.

Legacy SNMP agents SNMP-based agents that already exist in released products from Sun or outside companies. Solstice Enterprise Agents allows the integration of legacy SNMP *agents*.

managed node A network computer, router, hub, or other piece of equipment on the network that has object classes entered in the Solstice EM MIT and a network agent running on it.

managed object The representation of a network resource (or a set of resources). Note that in general a managed object is an abstraction that represents selected attributes of the resource it represents. The managed object resides within the MIS, where it represents a resource that is elsewhere. A managed object is characterized by:

- *attributes* visible at its boundary
- *management operations* that may be applied to it
- *behaviors* it exhibits in response to management operations
- *notifications* that it emits

A MIB or MIT entry that represents some aspect of a network node or line that is monitored and, in some cases, set, using Solstice EM services. The MIS manages the object by polling it, displaying the attribute values for current object instances of it, and in some cases changing the attribute values for instances of it.

managed object class	The formal description of a set of managed objects. A managed object is the collection of data that represents a managed resource. Specified in ITU Recommendation X.701 | ISO/IEC 10040.
management application	A management application is any program that retrieves and changes information about the manageable products on a desktop system. A management application talks to the Service Provider through the Management Interface (MI). For example, a remote network monitoring tool and a local control panel are management applications.
managing system	The system requesting information from and setting information in a network node running a network-management system.
Mapper	The integration of DMI 2.0 technology is done through the Mapper, that acts as a *subagent*. The Mapper receives the requests from the Master Agent and converts them into appropriate DMI requests that are sent to the DMI Service Provider. When the Mapper receives the response from the DMI Service Provider, it converts this response into the SNMP response and forwards it to the Manager through the Master Agent.
Master Agent	An entity/process on a managed node that exchanges SNMP protocol messages with the managers such as Domain Manager, Enterprise Manager, H-P Openview.
MetaData	The set of descriptions of the forms of data used to describe managed objects in a network (as distinct from the data itself).
MIB	Management Information Base. A hierarchical system for classifying information about resources in a network. By industry agreement, individual developers are assigned portions of the tree structure where they may attach descriptions specific to their own devices.
MIB module	A collection of managed objects.
network management agent	The implementation of a network management protocol (a program) that exchanges network management information with a network management station.
network management protocol	The protocol used to convey management information.
NMF	Network Management Forum. An association of vendors and developers of network hardware and software dedicated to the promotion of interoperable network management based on the use of OSI techniques.
OID	Object IDentifier. A number that identifies an object's position in a global object registration tree. An example is 1.3.6.1.4.1.45.1.3.2, that corresponds to ios.org.dod.internet.private.enterprise.synoptics.1.3.2, and identifies a Synoptics3000 concentrator. There may also be a MIB

	name for the object identifier (for example, `cisco` for a Cisco router). [S] In CMIP, one half of the Relative Distinguished Name (RDN) pair, that identifies an object's position in an *MIT*. See Name. [C] An Object IDentifier uses a system for describing an object's class by reference to a standard tree structure of descriptions. Each node of the tree is assigned a number, so that an object's identifier is a sequence of numbers. In Internet usage, the identifiers are shown as a string of numbers delimited by dots (for example, 0.128.45.12); in the OSI context (and in Solstice EM) the numbers are delimited by blanks and the entire sequence is surrounded by braces (for example, { 0 128 45 12 }).
OID range	The range of OIDs implied by a subtree. For instance, the subtree 1.2.3 carries an implied range of 1.2.3 up to but not including 1.2.4.
ONC/RPC	Open Network Computing/Remote Procedure Call.
OSF	Open Systems Foundation. UNIX consortium including Hewlett-Packard, IBM, and DEC, founded 1988. Sponsors of *DME*.
OSI	Open Systems Interconnection. General name for the set of network management conventions adopted by the International Standards Organization. An international effort (via ISO) to facilitate communications among computers of varying manufacturers and technology.
OSI/NMF	OSI Network Management Forum. An OSI group formed to develop and promulgate definitions and standards for the SNMP, PING, and CMIP protocols.
overlapping registration	An attempt by one subagent to *register* a subtree that is contained within or contains a subtree already registered by another subagent.
parent	An instance of the class containing a (*child*) object. [C]
poll	A periodic request for MIB or MIT object-class status information sent to a managed object. Configurable in some cases by the network administrator via Solstice EM Request Designer. SNMP tends to be poll-oriented, while CMIP tends to be event-oriented.
proprietary group	A proprietary group is a group of attributes that is specific to a particular product vendor and has not been proposed or standardized by the DMTF special interest group. Proprietary groups allow vendors to differentiate their product and demonstrate competitive advantages.
protocol	A set of rules used by computers to communicate with each other. A protocol is also the private language and procedures of an OSI layer.
registration	The act of a subagent informing the Master Agent that the subagent will provide management of a MIB *subtree*.

required group A required group is a group of attributes that are required to be included in a MIF file in order to be DMI-compliant. Currently, the only required group is the ComponentID group, that must be group 1 in any MIF file.

RFC Request for Comment. The series of documents that formalize protocols within the Internet (TCP/IP-based) community are referred to as RFC, the last phase in the formal standardization process before the document is made official. RFCs are published by the Internet Engineering Task Force (IETF).

router The term routing refers to the process of selecting a path to send packets over, and router is any computer able to make such a selection. Although both hosts and gateways do routing, the term router is commonly used for a device that interconnects two networks (See gateway).

SAP *Service Access Point*.The notional point where a service user and a layer entity meet so that services may be offered by the layer entity to the particular user.

severity enumeration A bit-mask so that multiple event severities may be selected for a filter entry.

sibling An object that shares a common parent class with the object in question. [C]

SMI Structure of Management Information.

SNM SunNet Manager. To export your subagent to Site/SunNet/Domain Manager (SNM), you need an SNM schema file.

SNMP Simple Network Management Protocol. Protocol for exchanging information between network managers and “agents’, processes within various managed objects that are able to report their status on request. The protocol was introduced as a simple interim solution, but is at present widely used in the Internet environment. It is a connection-less protocol, with the view of continuing to receive information from managed objects even when network performance is degraded and a connection-based reliable transport may fail.

SNMPD Simple Network Management Protocol Daemon.

standard group An MIF group that has been proposed to and accepted by the DMTF special interest group and that describes attributes applicable to all, or most all, products of a similar type, such as all printers or all network interface cards. Currently, standard groups are approved for PC Systems, have been proposed for Network Information Cards, and are in development for Printers, Servers, Software and Modems.

state A description of a managed object in a point in time with respect to a *request*. At any given moment, a request, reflecting the target managed

	object, is in some state defined in that request or is undergoing a *transition* between states. You may think of a state as a receptacle that holds transitions to other states. While in a state, a request repeatedly, at intervals determined by the state's *poll rate,* tests the *conditions* associated with each transition leading from that state. In addition to a poll rate, each state has a *severity* associated with it. It also has a name and a description. Between any two states, there is a single transition (one-way or two-way) with, potentially, multiple *conditions* associated with each transition. There is one required state, the ground (or init) state. The only requirement for this state is that it have a *severity* of "normal". Other states are of your choosing.
subagent	Process that has access to the management information and provides manageability to various applications/components within a system. A subagent communicates with the Master Agent using SNMP. A subagent does not communicate with a manager directly.
subnet	In Internet parlance, a logical partition of a network. OSI attaches a more restricted meaning: the portion of a network attached to the same physical medium.
subnet mask	A 32-bit quantity indicating bits in an IP address identify the physical network.
subtree	Indicated by a single OID, may be an entire MIB, a full instance, or even a subtree named in any MIB specification.
synthetic event	An event that is generated based on a composite analysis of various elements of state in the managed machine.
tables	A table of attributes is used when there is more than one set of attributes for a particular group. For instance, a computer system often has more than one serial port attached to it. To describe these serial ports, the Serial Port Group in the system's MIF file would be set up as a table, with one row in the table describing the specifics of a particular serial port.
TCP/IP	Transmission Control Protocol/Internet Protocol. The Internet suite of protocols is a group of protocols related to a common framework, or set of rules that defines how computers communicate with each other in an open (non-proprietary) system, typically a large communications infrastructure.
table	An SNMP term that describes a set of attribute values for object class instances. The rows represent the attributes and the columns represent class instances.
trap	In Internet jargon, notification of a problem that an agent sends to a management MIS of its own initiative rather than in response to a poll.

	SNMP formally defines seven types of traps and permits subtypes to be defined. OMNIPoint 1 uses the term “event report” rather than “trap”.
trap-directed polling	A hybrid form of trouble reporting where a single trap initiated by an agent is followed up by polls when the management MIS requests further information.
UDP	Universal Datagram Protocol. A connectionless protocol where SNMP is usually implemented.

Index

E

F

G

I

L

M

N

O

P

R

S

T

U

W