BEA Tuxedo®

SNMP Agent Administration Guide

Version 10.0
Document Released: September 28, 2007

Contents

1. BEA SNMP Agent Introduction

2. BEA SNMP Agent Architectural Models

3. Setting Up BEA SNMP Agent on a Managed Node

4. Integrating BEA SNMP Agent with a Management Framework

5. Setting Up the BEA SNMP Agent Integrator

6. Using Multiple SNMP Agents

7. Using the BEA SNMP Agent Integrator for Polling

8. BEA SNMP Agent Integrator Commands

9. Configuration Files

A. SNMP Information

CHAPTER 1

BEA SNMP Agent Introduction

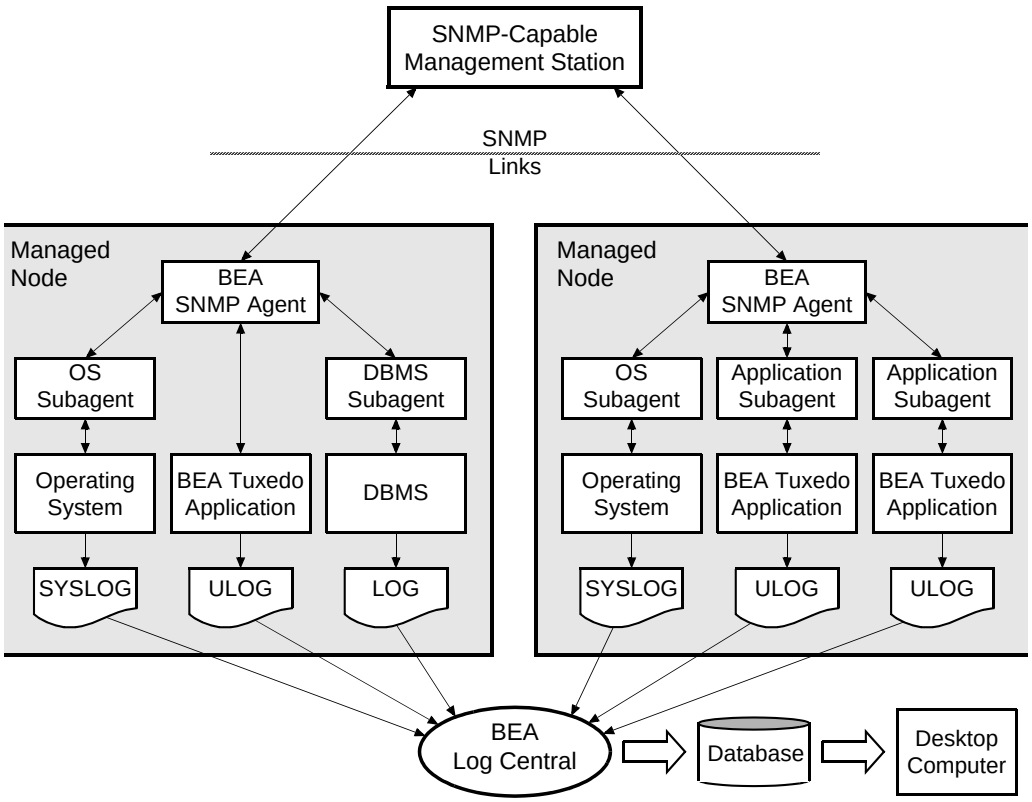
The following sections provide an overview of BEA SNMP Agent for BEA Tuxedo, the Simple Network Management Protocol (SNMP), the management information base (MIB), and the BEA SNMP Agent components:

About BEA SNMP Agent

BEA SNMP Agent for BEA Tuxedo, hereafter referred to as "BEA SNMP Agent 9.1" or simply "BEA SNMP Agent," enables SNMP-compliant network management frameworks to manage BEA Tuxedo systems and applications. BEA SNMP Agent complies with the Simple Network Management Protocol version 1 (SNMPv1) specification.

As shown in the following figure, BEA SNMP Agent provides the SNMP links from Tuxedo applications to SNMP-capable management stations (consoles). Also shown in the figure is that BEA SNMP Agent allows multiple SNMP agents and subagents—from any vendor—to operate on the same machine.

Figure 1-1 BEA SNMP Agent for BEA Tuxedo



BEA SNMP Agent provides the SNMP link from Tuxedo applications to SNMP-compliant network management frameworks (for example, HP OpenView) for monitoring, control, and alarm notification, as follows:

- Monitoring

  SNMP monitoring allows any supported SNMP-capable management station to monitor the state of the Tuxedo application. Also, appropriate actions can be triggered when a variable crosses a predefined threshold.

- Control

  An SNMP-capable management framework can modify the value of Tuxedo control and configuration parameters.

- Alarm notification

  The Tuxedo system generates certain system-wide events in case of exceptions (such as a Tuxedo node going down). These events are “trapped” and sent to the management framework.

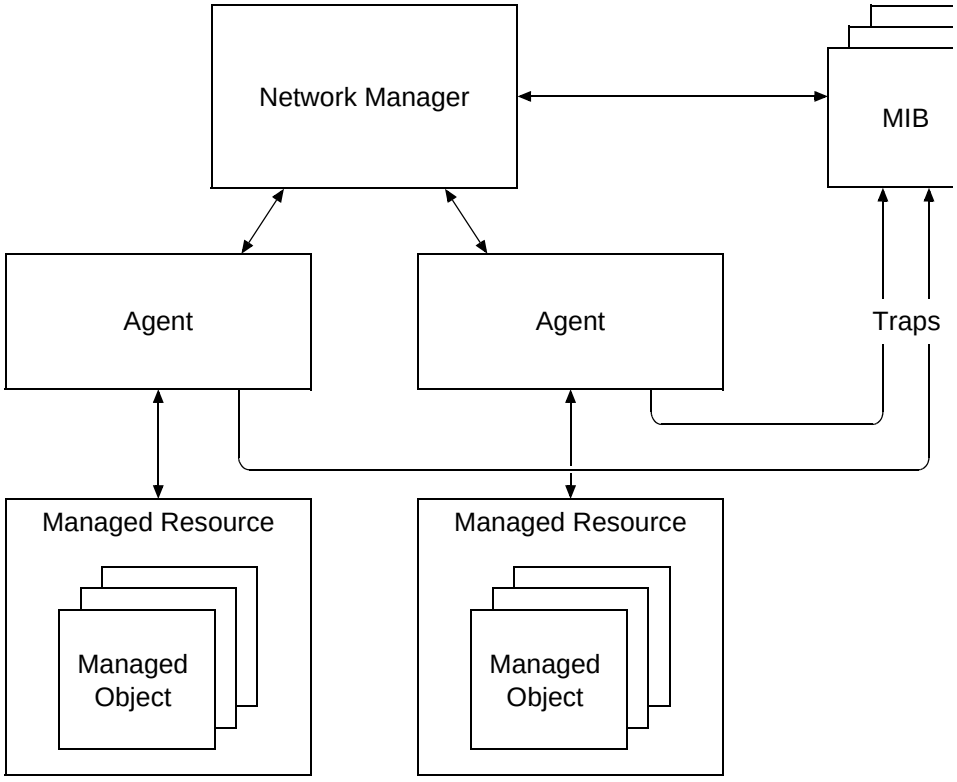
About SNMP

SNMP is an open network management standard for networks based on the Internet network protocols (TCP/IP). The basic SNMP standard for system management is defined in the Network Working Group (NWG) RFC 1157.

SNMP provides a standard system for classifying system information about hardware, software, and other aspects of a distributed client/server system. SNMP network and systems management is based on the manager/agent model described in the network management standards defined by the International Organization for Standardization (ISO).

In the model, shown in the following figure, a network manager exchanges monitoring and control information about network and system resources with distributed software processes called *agents*.

Figure 1-2 Manager/Agent Model



Any network or system resource that is manageable through this exchange of information is a *managed resource*. A managed resource could be a software resource, such as a message queue or a BEA Tuxedo application, or a hardware resource, such as a router or NFS file server.

SNMP enables you to correlate fault and performance data collected by different sources. For example, certain database inserts might fail because the filesystem on which the database resides has become full. This, in turn, might cause a BEA Tuxedo service to fail.

For a management framework to correlate this failure information with other aspects of system information and thus enable pro-active management of the system, all pieces of management information need to be available from the same management console. To achieve this level of correlation, a standardized method of communicating management information is required.

SNMP provides that standardized method. SNMP provides a unified way of representing information about the manageable features of the heterogeneous components of large distributed systems because it is an open network management standard for networks.

Agents function as “collection devices” that typically gather and send data about the managed resource in response to a request from a manager. In addition, agents often have the ability to issue unsolicited reports to managers when they detect certain predefined thresholds or events on a managed resource. In SNMP terminology, these unsolicited event reports are called *trap notifications*.

SNMP Management Information Base

An SNMP manager relies upon the management information base (MIB), a database that contains definitions and information about the properties of managed resources and the services the SNMP agents support. When a new agent is added to extend the manager’s domain, the manager must be provided with a new MIB component that defines the manageable features of the resources managed through that new agent.

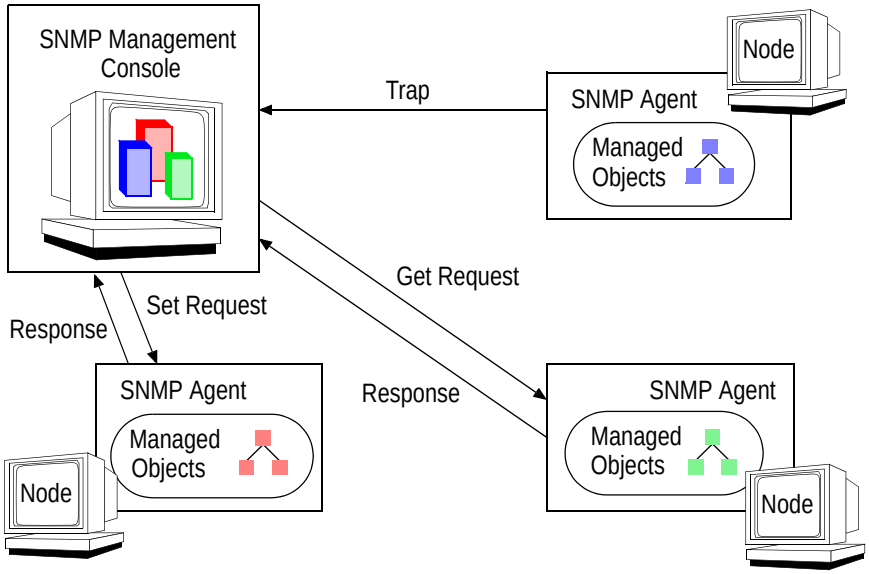
The manageable features of resources, as defined in an SNMP-compliant MIB, are called *managed objects* (also termed management variables or variables). Examples of managed (or MIB) objects include the state of a BEA Tuxedo domain, the number of users currently logged on to a BEA Tuxedo system, Tuxedo processes, and Tuxedo application variables. When the heterogeneous components of an enterprise’s distributed systems are defined within a common MIB on the management framework, a unified perspective and single access point is provided for managing system and network resources.

The data types and the representations of resources within a MIB, as well as the structure of a particular MIB, are defined in a standard called the Structure of Management Information (SMI). This standard is described in the NWG RFC 1155.

A formal language, known as the ISO Abstract Syntax Notation One (ASN.1), is used to describe MIB data independently of any encoding technique used. SNMP uses a subset of the ASN.1 language to represent a MIB.

The following figure shows an example SNMP installation that provides system administrator access to management information from a management console. Management commands are issued to SNMP agents to collect the values of various management variables (as defined in the platform’s MIB).

Figure 1-3 SNMP Management/Agent Interaction from a Management Console



Within most management frameworks, you can set up conditions to generate alarms based on defined event criteria. The criteria typically consist of changes in the values of certain attributes of the managed resources. These attributes are represented as MIB objects. You can also define the action to be taken when a specified event occurs, such as when a particular threshold is crossed.

The BEA SNMP MIB supports a full range of BEA Tuxedo system and application events. These system and application events are transmitted as enterprise-specific traps. See the *BEA Tuxedo SNMP Agent MIB Reference* for more information about the BEA SNMP MIB.

SNMP Protocol

SNMP protocol is based on request/response commands. A management framework sends a Get or GetNext command to request values of MIB variables from an agent, or a Set request to modify the value of a variable. Once the data is collected, management frameworks can present views or graphs of the information or take action in response to the information provided by SNMP agents.

Typically, management frameworks save the collected data to a repository for historical reporting. They also commonly include various tools and utilities to analyze the management

data. The management framework enables you to automate responses to event-based operations, change access privileges, update application information, and tune application parameters.

Benefits of Network Management Integration

Because BEA Tuxedo applications are part of an overall organization or business middleware solution, integrating them with SNMP enables you to effectively manage all your large-scale applications using the SNMP-compliant network management tool of your choice. Since most management frameworks support SNMP, BEA SNMP Agent for Tuxedo applications can be integrated into virtually every management framework. Examples of such management frameworks include:

- HP OpenView Network Node Manager
- IBM Tivoli NetView
- Evidian OpenMaster
- CA Unicenter TNG
- BMC Patrol
- Solstice

With these management frameworks, you can manage and control systems, databases, applications, and user access from a centralized management console. The tools available from the management framework enable you to automate and delegate routine and complex system tasks.

Using SNMP to manage BEA Tuxedo applications provides the following benefits:

- Movement towards a single management console, providing integrated systems management of Tuxedo-based applications.
- Hooking of Tuxedo applications into popular management frameworks, thereby making the management of Tuxedo applications more effective by providing a whole-system perspective instead of piecemeal solutions.
- Investment preservation in standard, SNMP-compliant management frameworks.

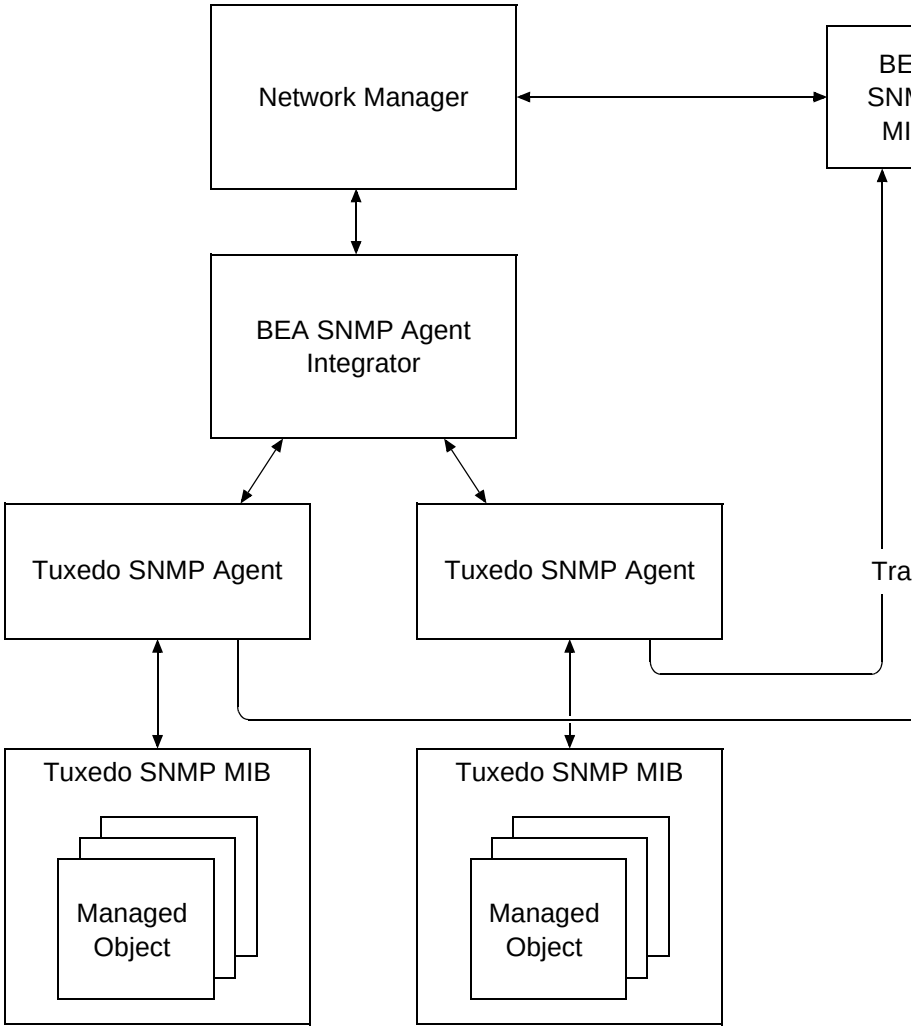
BEA SNMP Agent Components

BEA SNMP Agent consists of the following major components:

- Tuxedo SNMP agent
- BEA SNMP Agent Integrator
- SNMP MIB files
- Configuration files
- Commands and utilities

The following figure shows how the Tuxedo SNMP agent, the BEA SNMP Agent Integrator, and the BEA SNMP MIB work together to incorporate Tuxedo information within an SNMP management framework.

Figure 1-4 BEA SNMP Agent Components



Note: The BEA SNMP Agent component files reside in the *tux_prod_dir*/bin directory and the *tux_prod_dir*/udataobj/snmp/etc directory, where *tux_prod_dir* represents the directory in which the BEA Tuxedo 9.1 distribution is installed.

Tuxedo SNMP Agent

The Tuxedo SNMP agent, named `tux_snmpd`, responds to requests from SNMP managers and generates SNMP trap notifications for Tuxedo system and application events. The `tux_snmpd` executable translates management information for Tuxedo applications into a form compatible with SNMP, and stores the information in a local SNMP MIB for later access by the SNMP manager (a front-end process running in the SNMP network management framework).

The `tux_snmpd` executable can be started as an SNMP agent or an SNMP multiplexing (SMUX) subagent. The SMUX protocol is defined in RFC 1227.

BEA SNMP Agent Integrator

The BEA SNMP Agent Integrator, named `snmp_integrator`, provides both SNMP master agent and SMUX master agent capabilities. As such, the `snmp_integrator` executable enables multiple SNMP agents and SMUX subagents from any vendor to coexist on the same managed node (machine) and appear as a single SNMP agent to the SNMP manager.

SNMP MIB Files

BEA SNMP Agent provides the following two SNMP MIB files.

MIB Filename	Description
`bea.asn1`	Defines the SNMP MIB, a translation of the Tuxedo MIB, that makes the features of the Tuxedo components recognizable and thus manageable within an SNMP network management framework. Used by the SNMP network management framework to set up that portion of its local MIB (on the management station) required to manage Tuxedo applications.
`mib.txt`	Provides a textual description of the content of the SNMP MIB similar to the `bea.asn1` file—in fact, `mib.txt` is created from the `bea.asn1` file. Used by `tux_snmpd` at startup to set up its local SNMP MIB on the managed node.

The SNMP MIB, created from the `bea.asn1` and `mib.txt` files, makes the features of the following components manageable within an SNMP network management framework:

- Tuxedo 9.1 components
- Tuxedo 9.0 components
- Tuxedo 8.1 components
- Tuxedo 8.0 components
- Tuxedo 7.1 components
- Tuxedo 6.5 components

You can use BEA SNMP Agent 9.1 and the SNMP MIB to manage Tuxedo 9.1, 9.0, 8.1, 8.0, 7.1, and 6.5 applications. You cannot use BEA SNMP Agent 9.1 and the SNMP MIB to manage WebLogic Enterprise 5.1 or earlier applications.

For more information about the BEA SNMP MIB, see *BEA Tuxedo SNMP Agent MIB Reference*.

Configuration Files

BEA SNMP Agent provides the following two configuration files.

Configuration Filename	Description
`beamgr.conf`	Also known as the "BEA SNMP Agent configuration file." Contains the user-defined operational configurations read by `tux_snmpd` and `snmp_integrator` at startup.
`beamgr_snmpd.conf`	Also known as the "BEA SNMP Agent passwords configuration file." Contains the user-defined password configurations (SNMP community names, SMUX password) read by `tux_snmpd` and `snmp_integrator` at startup. The default read-only community name is `public`, and the default read-write community name is `iview`. The default SMUX password is no password.

For more information about the BEA SNMP Agent configuration files, see *"Configuration Files" on page 9-1*.

Commands and Utilities

The BEA SNMP Agent software provides the following commands to control `tux_snmpd` and `snmp_integrator` processes.

Command Name	Description
`stop_agent`	UNIX only*: Terminates the specified `tux_snmpd` and/or `snmp_integrator` processes.
`reinit_agent`	Windows and UNIX: Causes the specified `tux_snmpd` and/or `snmp_integrator` processes to re-read the local BEA Tuxedo configuration files. On UNIX systems, this command must be run with `root` permissions.
`show_agent`	Windows and UNIX: Lists the names and process IDs (PIDs) of the specified `tux_snmpd` and/or `snmp_integrator` processes.

* On a Windows system, a BEA SNMP agent (`tux_snmpd`, `snmp_integrator`) process can *only* be stopped as a Windows service; command-line shutdown is *not* supported.

The BEA SNMP Agent software provides the following utilities to install and test `tux_snmpd` and `snmp_integrator` processes:

Utility Name	Description
`instsrv`	Windows system only: Installs `tux_snmpd` or `snmp_integrator` as a Windows service.
`snmpget`	Reports information about managed objects in the SNMP MIB.
`snmpgetnext`	Returns the next consecutive managed object in the SNMP MIB.
`snmptest`	Selectively performs `Get` and `Set` operations on any SNMP MIB object.
`snmptrap`	Sends an SNMP `TrapRequest` message to a host.
`snmptrapd`	Receives and logs SNMP `TrapRequest` messages sent on a local machine to the SNMP trap port.
`snmpwalk`	Traverses the object identifier (OID) tree using the SNMP `GetNextRequest` message to query managed objects.

For more information about the BEA SNMP Agent commands and utilities, see *"BEA SNMP Agent Integrator Commands" on page 8-1*.

CHAPTER 2

BEA SNMP Agent Architectural Models

The following sections describe the agent/subagent architectural models supported by BEA SNMP Agent:

Architectural Models Identified

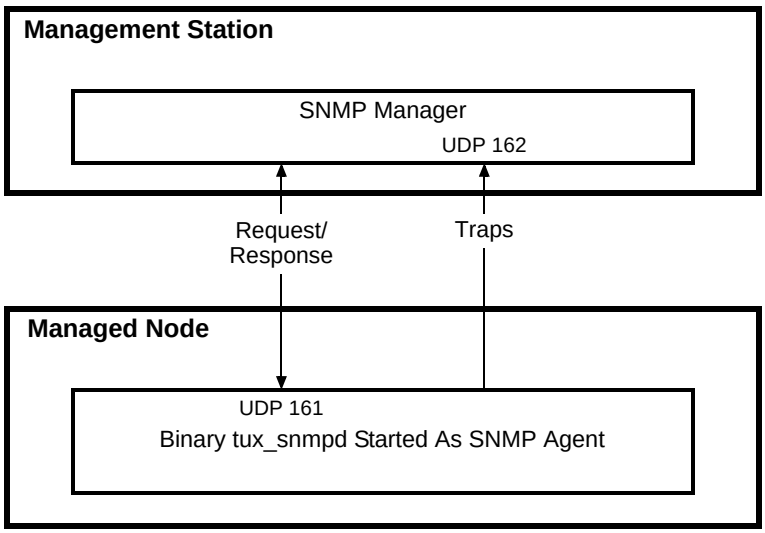
BEA SNMP Agent supports three architectural models: the simple SNMP agent model, the SNMP master agent/ subagent model, and the SMUX master agent/ subagent model. The latter two models, as well as the combination of the latter two models, are made possible by the SNMP master agent and SMUX master agent capabilities provided by the BEA SNMP Agent Integrator (`snmp_integrator`). The SNMP multiplexing (SMUX) protocol is defined in RFC 1227.

By providing the BEA SNMP Agent Integrator and supporting the SMUX protocol, BEA SNMP Agent is able to support multiple SNMP agents and subagents from any vendor to operate on the same machine and appear as a single SNMP agent to the SNMP management console.

Simple SNMP Agent Model

The simple SNMP agent model, shown in the following figure, applies when managing a single Tuxedo application (domain) running on the managed node.

Figure 2-1 Single SNMP Agent Running on a Managed Node—Example



With this model, the `tux_snmpd` process is started with the `-s` option in the startup command (`tux_snmpd -s`) to configure the process as an *SNMP agent*—as opposed to a *SMUX subagent*.

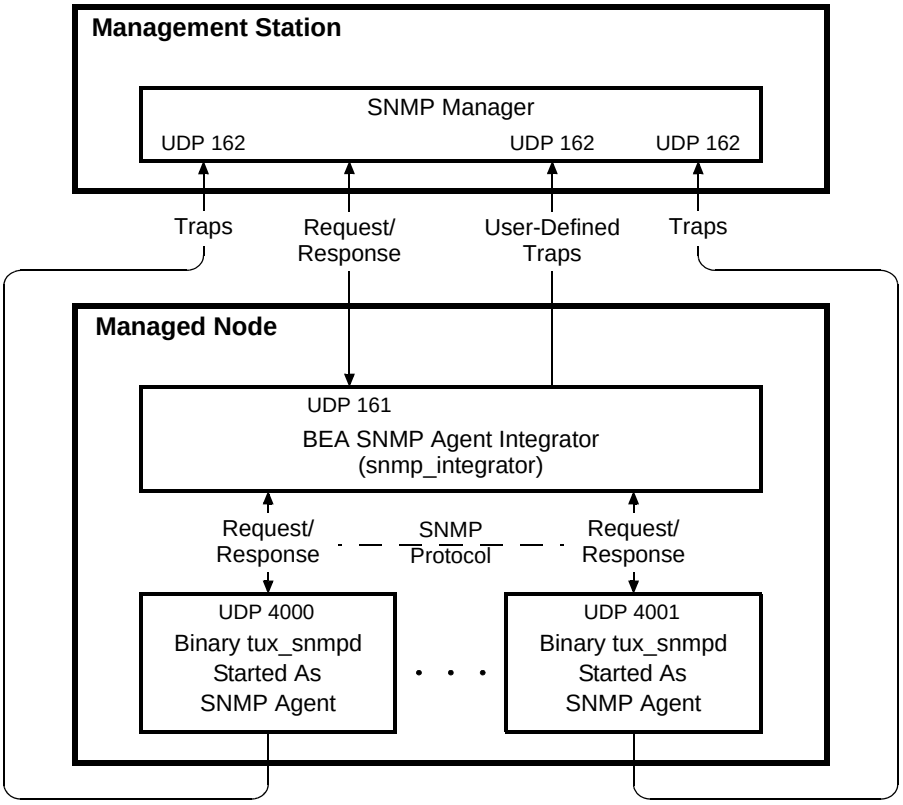
Including the `-c` option in the `tux_snmpd` startup command (`tux_snmpd -s -c`) enables users to run only one Tuxedo SNMP agent on the master node of a Tuxedo domain, to simplify management of multiple-node domains and to enable data collection from hardware platforms not directly supported by the BEA SNMP Agent software. When the `-c` option is specified, the agent becomes central for the entire Tuxedo domain and is enabled to gather information from all nodes involved in the domain.

SNMP Master Agent/ Subagent Model

The SNMP master agent/ subagent model, shown in the following figure, applies when managing different types of system or network resources on the managed node. This model does *not* apply

to managing multiple instances of the *same* type of resource; for example, you *cannot* use this model to manage multiple Tuxedo domains running on the same managed node.

Figure 2-2 Multiple SNMP Agents Running on a Managed Node—Example



Note: The SNMP agents must be started before starting the BEA SNMP Agent Integrator.

With this model, communication between the SNMP manager and the SNMP agents is handled by the BEA SNMP Agent Integrator *SNMP master agent* capability. The BEA SNMP Agent Integrator distributes the requests from the SNMP manager to specific SNMP agents, receives the responses from the individual agents, and forwards those responses back to the SNMP manager.

Starting tux_snmpd Processes as SNMP Agents

Each `tux_snmpd` process is started with the `-s` option to configure each `tux_snmpd` process as an *SNMP agent*—as opposed to a *SMUX subagent*. SNMP agents running under the BEA SNMP Agent Integrator are also known as *non-SMUX peer agents*.

Managing a Tuxedo Domain

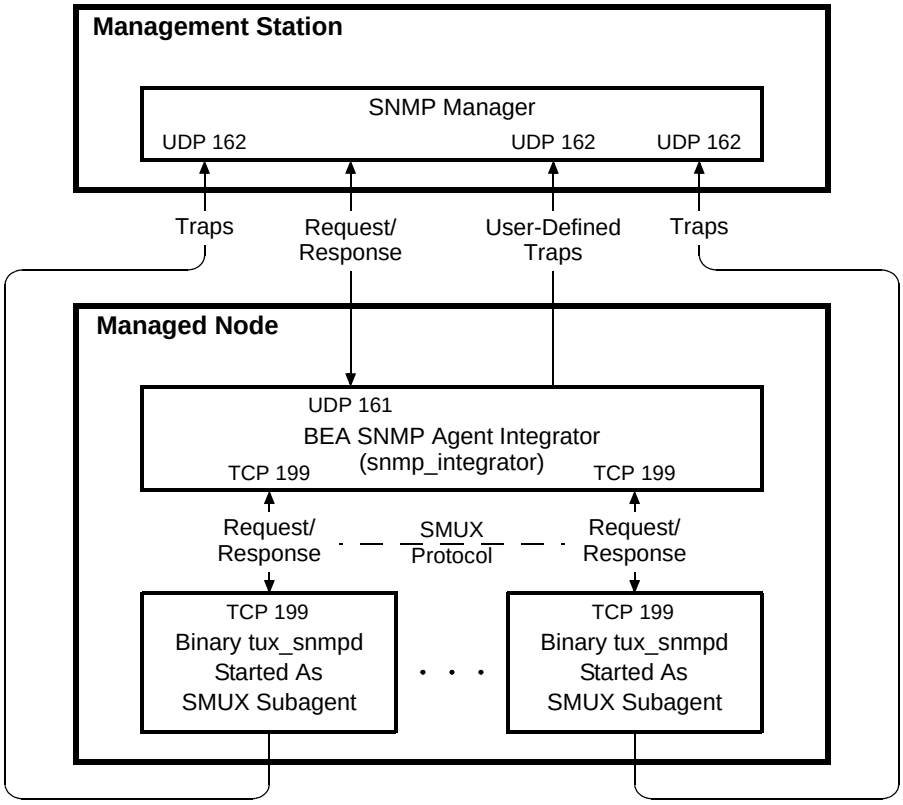
With the SNMP master agent/ subagent model, one and only one of the managed resources on the managed node may be a Tuxedo domain. The `tux_snmpd` process targeted to manage that domain may be started with the `-l` *`logical_agent_name`* option to assign the process a name other than the default name "tux_snmpd." The `-l` *`logical_agent_name`* option or the default name "tux_snmpd" allows an administrator to associate the `tux_snmpd` process with the Tuxedo domain that the process is to manage.

Also, as described in "Simple SNMP Agent Model" on page 2-2, if the managed node is the master machine for the Tuxedo domain, including the `-c` option in the startup of the associated `tux_snmpd` process enables that process to manage the entire domain.

SMUX Master Agent/ Subagent Model

The SMUX master agent/ subagent model, shown in the following figure, applies when managing different types of system or network resources on the managed node. This model *also applies* to managing multiple instances of the *same* type of resource; for example, you *can* use this model to manage multiple Tuxedo domains running on the same managed node.

Figure 2-3 Multiple SMUX Subagents Running on a Managed Node—Example



Note: The BEA SNMP Agent Integrator must be started before starting the SMUX subagents.

With this model, communication between the SNMP manager and the SMUX subagents is handled by the BEA SNMP Agent Integrator *SMUX master agent* capability. The BEA SNMP Agent Integrator distributes the requests from the SNMP manager to specific SMUX subagents, receives the responses from the individual subagents, and forwards those responses back to the SNMP manager.

Starting tux_snmpd Processes as SMUX Subagents

Each `tux_snmpd` process is started *without* the `-s` option to configure each `tux_snmpd` process as a *SMUX subagent*—as opposed to an *SNMP agent*. SMUX subagents use the SNMP

multiplexing (SMUX) protocol defined in RFC 1227 to communicate with the BEA SNMP Agent Integrator.

Managing Multiple Tuxedo Domains Concurrently

With the SMUX master agent/ subagent model, multiple Tuxedo domains running on the managed node may be managed concurrently. To do so, an administrator starts a SMUX subagent for each domain. And for each SMUX subagent, the administrator specifies the `-l` *`logical_agent_name`* option at startup to associate the `tux_snmpd` process with the Tuxedo domain that the process is to manage.

The SNMP manager running on the management station uses an address of the following form to send requests to a particular SMUX subagent:

`community_name``@`*`logical_agent_name`*

Where *`community_name`* is the read-only community name for `Get` requests, and the read-write community name for `Set` requests; and *`logical_agent_name`* is the name of the agent to which the SNMP request is intended. For example:

```
public@simpapp_agent
```

Also, as mentioned in "Simple SNMP Agent Model" on page 2-2, if the managed node is the master machine for a Tuxedo domain, including the `-c` option in the startup of the associated `tux_snmpd` process enables that process to manage the entire domain.

CHAPTER 3

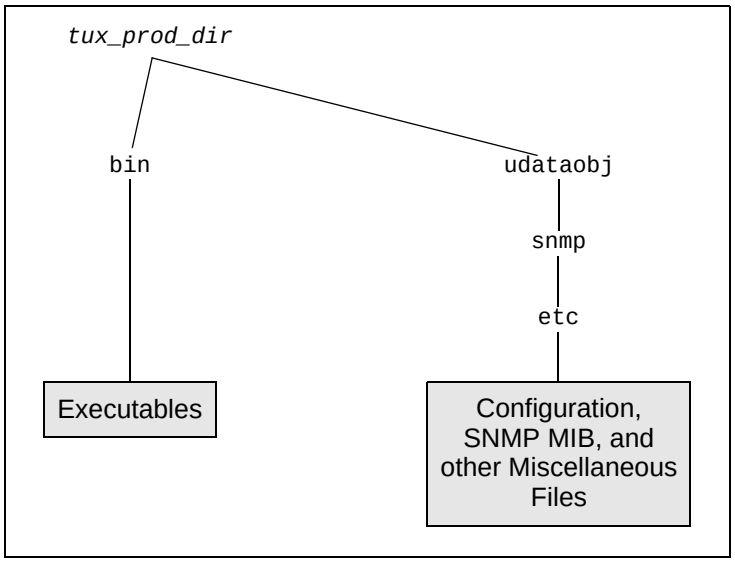
Setting Up BEA SNMP Agent on a Managed Node

To integrate BEA SNMP Agent into your management framework, you need to set up the BEA SNMP Agent software on the managed node and on the management framework. The following sections describe the procedure for setting up the BEA SNMP Agent on the managed node:

Directory Structure

The BEA SNMP Agent files reside in the directories shown in the following figure.

Figure 3-1 Directory Structure



Note: *tux_prod_dir* represents the directory in which the BEA Tuxedo 9.1 distribution is installed.

BEA SNMP Agent Configuration Files

BEA SNMP Agent provides the following two configuration files: `beamgr.conf` and `beamgr_snmpd.conf`. The `beamgr.conf` file, also known as the "BEA SNMP Agent configuration file," contains the user-defined operational configurations read by the Tuxedo SNMP agent (`tux_snmpd`) and the BEA SNMP Agent Integrator (`snmp_integrator`) at startup.

The `beamgr_snmpd.conf` file, also known as the "BEA SNMP Agent passwords configuration file," contains the user-defined password configurations (SNMP community names, SMUX password) read by `tux_snmpd` and `snmp_integrator` at startup. The default read-only community name is `public`, and the default read-write community name is `iview`. The default SMUX password is no password.

For more information about the BEA SNMP Agent configuration files, see *"Configuration Files" on page 9-1*.

BEA SNMP Agent Configuration

To configure BEA SNMP Agent, follow these steps:

1. Install the BEA SNMP Agent on the BEA Tuxedo server nodes to be managed.

   The Tuxedo SNMP agent `tux_snmpd` is installed one at a time. On a Windows system, if you do not install BEA Tuxedo first, you do not get the option to install `tux_snmpd`. For detailed information about how to install BEA SNMP Agent, see *Installing the BEA Tuxedo System*.

   Some attributes of Tuxedo resources are accessible globally (that is, no matter which Tuxedo node they are on) while others are accessible only by a BEA SNMP agent local to the same machine. To access managed objects that are only accessible locally, you must install BEA SNMP agents on each machine where these resources reside, or install a BEA SNMP agent on the master node and execute it with the `-c` option, which enables you to run the agent only on the master node but to still gather information from all machines.

2. Set up access to Tuxedo shared binaries.

   – On a Windows system:

     If BEA SNMP Agent is not installed in the same directory as the BEA Tuxedo application, make sure that the `bin` directory of the appropriate Tuxedo installation precedes any other Tuxedo installations in the `PATH` system environment variable. This directory order in `PATH` enables BEA SNMP Agent to have access to the correct Tuxedo dynamic link libraries (DLLs).

   – On a UNIX system:

     Make sure the search path for shared libraries includes `$TUXDIR/lib`. The search path for shared libraries is:

     `SHLIB_PATH` on HP-UX, `LIBPATH` on AIX, and `LIBRARY_PATH` on all other UNIX systems.

3. Install the BEA SNMP Agent configuration file.

   – On a Windows system:

     Open a command-line shell and copy the BEA SNMP Agent configuration file `beamgr.conf` to the `C:\etc` directory:

     ```
     prompt> md c:\etc
     prompt> copy tux_prod_dir\udataobj\snmp\etc\beamgr.conf
               c:\etc
     ```

– On a UNIX system:

  Log in as `root` and copy the BEA SNMP Agent configuration file `beamgr.conf` to the `/etc` directory:

  ```
  prompt> su
  prompt> Password:
  prompt> cp tux_prod_dir/udataobj/snmp/etc/beamgr.conf /etc
  ```

4. Set your `PATH` to include the location of the BEA SNMP Agent executables. This step applies to both Windows and UNIX systems.

   All users of the installed BEA SNMP Agent products need to update their `PATH` environment variable to include the location of the BEA SNMP Agent executable files. The following is a UNIX example in C shell:

   ```
   prompt> set path = ( $PATH tux_prod_dir/bin )
   ```

5. If you are running the BEA SNMP agent as an SNMP multiplexing (SMUX) subagent, set your master agent timeout. The SMUX protocol is defined in RFC 1227.

   Configure the timeout of your SMUX master, if any (such as `snmp_integrator`), and of your SNMP manager, to at least 30 seconds. For `snmp_integrator`, you can set this timeout by adding an `INTEGRATOR_TIMEOUT` entry to the BEA SNMP Agent `beamgr.conf` configuration file as follows:

   ```
   INTEGRATOR_TIMEOUT 30
   ```

6. When BEA SNMP Agent is installed on a Windows system, ensure that a match exists between the TCP/IP host name and the computer name.

   Check that the host name specified in Start->Settings->Control Panel->Network->Identification is all UPPERCASE and matches the host name specified in Start->Settings->Control Panel->Network->Protocols->TCP/IP-> Properties->DNS, which should also be all UPPERCASE.

7. Specify the destination for traps.

   The default destination for SNMP trap notifications is `localhost`. To send traps to other destinations, use a text editor to modify the `TRAP_HOST` entry in the BEA SNMP Agent `beamgr.conf` configuration file to specify the host name of the target destination machine for SNMP trap notifications, and the port number and community name to use in sending traps.

   Typically, the destination is the host machine where the SNMP management framework is located. Some management frameworks use distributed trap daemons that “collect” SNMP

trap notifications for forwarding to management stations. In that case, the machine with the trap daemon should be the destination.

For more information, see "Configuration Files" on page 9-1.

8. Identify the domain to be managed.

   The identity of the BEA Tuxedo application to be managed can be specified in two ways. BEA SNMP Agent uses the following sources in the indicated order of precedence:

   a. The `TMAGENT` entry in the BEA SNMP Agent configuration file. This entry is of the form:

      `TMAGENT` *`logical_agent_name tuxdir tuxconfig_path`*

      For more information, see "Configuration Files" on page 9-1.

   b. `TUXCONFIG` and `TUXDIR` environment variables

9. Ensure that the BEA Tuxedo EventBroker is configured.

   BEA SNMP Agent cannot receive Tuxedo event notifications unless the Tuxedo EventBroker server (`TMSYSEVT`) is running. To enable forwarding of Tuxedo events as SNMP traps, ensure that the Tuxedo EventBroker servers are running. For information on the Tuxedo EventBroker, see "About the EventBroker" in *Administering a BEA Tuxedo Application at Run Time* and reference page `TMSYSEVT(5)` in *BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes Reference*.

10. If you are using only BEA SNMP agents, start the Tuxedo SNMP agents on the managed nodes where your Tuxedo resources reside. For more information, see "Starting BEA SNMP Agent" on page 3-8.

    If you are using the BEA SNMP Agent Integrator, follow the instructions in "Setting Up the BEA SNMP Agent Integrator" on page 5-1 and set up the BEA SNMP agents and then the BEA SNMP Agent Integrator.

11. Integrate BEA SNMP Agent with your SNMP management framework. See "Integrating BEA SNMP Agent with a Management Framework" on page 4-1.

BEA SNMP Agent Advanced Configuration

If you want to customize BEA SNMP Agent for tasks such as managing multiple BEA Tuxedo domains concurrently or using nondefault ports for communication with the system manager, perform the following additional steps:

1. Define logical agent names if you want to manage multiple Tuxedo domains concurrently.

To manage multiple Tuxedo domains on a managed node at the same time, add a `TMAGENT` entry to the BEA SNMP Agent configuration file for each agent. The `TMAGENT` entry is of the following form:

`TMAGENT` *`logical_agent_name tuxdir tuxconfig_path`*

To manage multiple domains on a managed node, run a separate Tuxedo agent for each domain being monitored. These agents must be run as SMUX subagents under the BEA SNMP Agent Integrator.

When multiple agents running as SMUX subagents are running on the same node, SNMP manager Set or Get requests to a particular agent must be addressed using a community of the form:

`community@logical_agent_name`

where *`logical_agent_name`* identifies the agent to which the SNMP request is forwarded. For example:

```
public@simpapp_agent
```

If only one agent is running on a node, *`logical_agent_name`* is optional in specifying the community in Set or Get requests.

2. Define Tuxedo event filters to be used.

   Tuxedo event filters can define a subset of Tuxedo events to be received by the agent for each domain being monitored. You can use `TMEVENT_FILTER` entries in the BEA SNMP Agent configuration file to define a subset of Tuxedo event notifications that are to be forwarded as SNMP trap notifications. For more information, see "Configuration Files" on page 9-1. MIB objects corresponding to Tuxedo event filters are described in "Core MIB" in *BEA Tuxedo SNMP Agent MIB Reference*.

3. Specify non-default SNMP communities and SMUX password.

   By default, an SNMP agent (such as the BEA SNMP Agent Integrator or `tux_snmpd` when running as an SNMP agent) uses `public` as the read-only community and `iview` as the read-write community when communicating with SNMP managers. To define additional community names, specify them in the BEA SNMP Agent passwords file. You can also use the passwords file to specify a password for the BEA SNMP Agent Integrator to use for authenticating connection requests from SMUX subagents.

   a. To set up the passwords file:

      On a Windows system:

      Open a command-line shell and copy the BEA SNMP Agent `beamgr_snmpd.conf` passwords file to `c:\etc`. For example:

```
prompt> copy tux_prod_dir\udataobj\snmp\etc\beamgr_snmpd.conf
          c:\etc
```

On a UNIX system:

Copy the BEA SNMP Agent `beamgr_snmpd.conf` passwords file to the `/etc` directory and make the copy readable and writable only by `root`. For example:

```
prompt> cp tux_prod_dir/udataobj/snmp/etc/beamgr_snmpd.conf
          /etc
prompt> chmod 600 /etc/beamgr_snmpd.conf
```

b. Modify the SNMP communities in this file. The keywords used in this file are:

`SMUX_PASSWD`

`COMMUNITY_RO`

`COMMUNITY_RW`

`DISABLE_SET`

c. If you want to set the agent to be read-only, specify a `DISABLE_SET` entry in the passwords file as follows:

```
DISABLE_SET YES
```

If there is no `DISABLE_SET` entry in the passwords file, the agent has both Set and Get capability.

For more information, see "Configuration Files" on page 9-1.

4. Specify a SMUX password when using the BEA SNMP agent component as a SMUX subagent under a SMUX master agent, such as the BEA SNMP Agent Integrator.

   The environment variable `BEA_SMUX_PASSWD` specifies the password that the SNMP agent uses when registering with a SMUX master agent, such as the BEA SNMP Agent Integrator. This environment variable is required only if the SMUX master agent expects a password. If this environment variable is not set, a password is not specified by `tux_snmpd` when registering.

5. Define different port numbers.

   By default, BEA SNMP agents assume the following port numbers as specified by SNMP and SMUX standards:

```
snmp         161/udp
snmp-trap    162/udp
smux         199/tcp
```

If the default port assignments are not sufficient for your needs, you can define these services on other ports, or use the appropriate command-line options when starting SNMP agents to assign them to nondefault ports.

– On a Windows system:

  To modify or define the services, add the appropriate lines in the *root_directory*\system32\drivers\etc\services file. For example:

```
snmp        161/udp   snmp
snmp-trap   162/udp   snmp
```

  Consult your Windows system administrator for the default settings used for your SNMP-related services.

– On a UNIX system:

  To modify or define the services, perform these steps:

a. Determine if the NIS server is running. Use the `ypwhich` command to determine if an NIS server or map master is available. For example:

```
prompt> ypwhich
zort.kremvax.com
```

b. If an NIS server is available, use the `ypcat` command to determine if the services are available.

```
prompt> ypcat services | grep snmp
snmp-trap    162/udp   snmptrap
snmp         161/udp
```

c. If an NIS server is not available and services are provided on the local host, examine the `/etc/services` file.

```
prompt> cat /etc/services | grep snmp
snmp-trap    162/udp   snmptrap
snmp         161/udp
```

  To establish the SNMP services, refer to your UNIX system documentation as needed for instructions specific to your UNIX platform.

Starting BEA SNMP Agent

To manage multiple BEA Tuxedo domains, you can run multiple BEA SNMP agents on the same node. Each agent can manage only one domain. To manage multiple domains, you must have the BEA SNMP Agent Integrator running and the agents must be started as SMUX subagents.

On startup, a Tuxedo SNMP agent checks for a TMAGENT entry in the BEA SNMP Agent configuration file that matches its logical agent name. A TMAGENT entry provides a path to the Tuxedo domain to be monitored. If no matching TMAGENT entry is found, the agent connects to the Tuxedo domain specified in the TUXCONFIG and TUXDIR environment variables. The agent exits if the TUXCONFIG or TUXDIR environment variable is not defined and no appropriate TMAGENT entry is found in the BEA SNMP Agent configuration file. For more information, see "Configuration Files" on page 9-1.

BEA SNMP Agent Processes

The tux_snmpd binary is the Tuxedo SNMP agent that supports the Tuxedo SNMP MIB. For a description of the supported MIB groups and objects, see *BEA Tuxedo SNMP Agent MIB Reference*.

The BEA SNMP agent can run as an SNMP agent or as a SMUX subagent.

When the BEA SNMP agent starts up as an SNMP agent, it generates a coldStart trap. The destination host, port, and community used when sending traps are as specified in the TRAP_HOST entry in the BEA SNMP Agent beamgr.conf configuration file. For more information, see "BEA SNMP Agent Configuration" on page 3-3.

When running as a SMUX subagent, the BEA SNMP agent specifies a password to the SMUX master agent at the time of registration if the environment variable BEA_SMUX_PASSWD has been defined. In that case, the BEA SNMP agent uses the value of BEA_SMUX_PASSWD as the password; if BEA_SMUX_PASSWD has not been defined, the BEA SNMP agent does not specify a password to the master agent when registering.

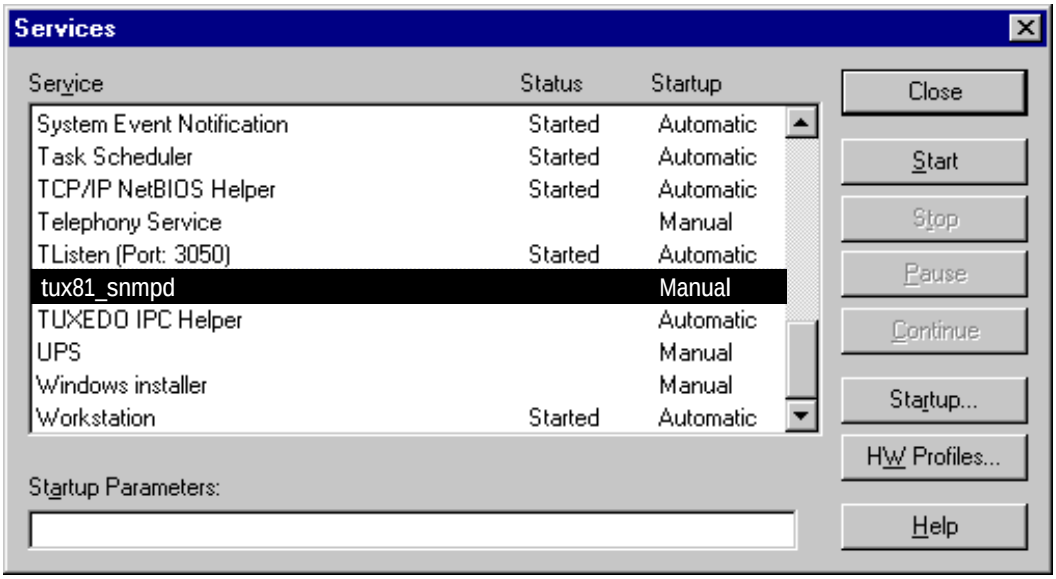
The tux_snmpd supports the MIB-II snmp group when running as the SNMP agent.

Starting BEA SNMP Agent on a Windows System

To start BEA SNMP agents on a Windows system, follow these steps:

1. On the Windows taskbar, choose Start->Settings->Control Panel->Services (or Start->Programs->Administrative Tools->Services on a Windows 2003 system) to display the Services window.
2. In the list of Services, locate and select the installed service named tux81_snmpd and click Start to start it, as shown in the following figure. There may be a short delay as the service is initiated.

Figure 3-2 Starting a Service



3. Install additional Windows services if you want to run multiple agents on a single node.

   The installation program for Windows installs the SNMP agent as a single Windows service. If you want to run multiple instances of the agent to monitor multiple Tuxedo domains, you need to install additional Windows services for the additional agents.

   To install additional Windows services for Tuxedo SNMP agents, open a command-line shell and run the following command for each additional Tuxedo SNMP agent:

   ```
   prompt> instsrv logical_agent_name
             tux_prod_dir\bin\tux_snmpd.exe
   ```

   Assign separate logical agent names to run multiple instances of the agent on the same node. To use multiple agents to monitor multiple Tuxedo domains, *`logical_agent_name`* is a string that associates an agent with a Tuxedo domain as defined by a `TMAGENT` entry in the BEA SNMP Agent `beamgr.conf` configuration file. For format information, see "BEA SNMP Agent Advanced Configuration" on page 3-5.

   This entry assigns the agent started with *`logical_agent_name`* to the indicated Tuxedo domain. See "Configuration Files" on page 9-1.

Windows Startup Options

Enter the desired startup options in the Startup Parameters field in the Services window.

`-d`
: Dumps the SNMP or SMUX packets received and sent by the agent to the Windows Event Log.

`-s`
: Specifies that the BEA SNMP agent is to run as an SNMP agent. If you do not specify this option, the BEA SNMP agent runs as a SMUX subagent. If a SMUX master agent (for example, `snmp_integrator`) is not running, you must provide `-s` as a startup parameter before selecting Start.

`-p` *`snmp_port`*
: The *`snmp_port`* option specifies the UDP port on which the BEA SNMP agent listens for incoming SNMP packets. The `-p` option enables you to run the BEA SNMP agent on a port other than the standard SNMP port 161. This option is meaningful only when the BEA SNMP agent is running as an SNMP agent.

`-r` *`smux_port`*
: Specifies the TCP port to connect to a SMUX master agent. (The default is port 199.) This option is meaningful only when `tux_snmpd` is running as a SMUX subagent.

`-m` *`hostname`*
: The name of the machine where the SMUX master agent, such as the BEA SNMP Agent Integrator, is running. This option is used only when you want `tux_snmpd` to register with a SMUX master agent on a remote machine.

`-c`
: Enables you to run the agent only on the master node but to still gather information from all machines. This results in a more manageable solution because it requires you to run one agent process per domain instead of one per node. In addition, it enables you to gather SNMP information from nodes with operating systems not supporting the current SNMP Agent.

  When using this option, you must ensure that only one agent is started on the domain; otherwise, the results are unpredictable.

Starting BEA SNMP Agent on a UNIX System

To start BEA SNMP agents on a UNIX system, enter the Tuxedo SNMP agent startup command at the command-line prompt:

```
tux_snmpd [-l logical_agent_name] [-d] [-n] [-s] [-p snmp_port]
[-r smux_port] [-m hostname] [-h] [-c]
```

UNIX Startup Options

The command line options are:

`-l` *`logical_agent_name`*

The *`logical_agent_name`* string associates an agent with a BEA Tuxedo domain as defined by a `TMAGENT` entry in the BEA SNMP Agent `beamgr.conf` configuration file. The logical agent name can be a maximum of 32 characters long. For format information, see "BEA SNMP Agent Advanced Configuration" on page 3-5.

Assign separate logical agent names to run multiple instances of the agent on the same node. If you do not specify the `-l` option, the BEA SNMP agent uses the name of the executable as the logical agent name.

`-d`

Dumps the SNMP or SMUX packets received and sent by the agent to standard output.

`-n`

If the agent/subagent is run with this option, it does not become a daemon. Use this option to start the BEA SNMP agent with the `init` command.

`-s`

Specifies that the BEA SNMP agent is to run as an SNMP agent. If you do not specify this option, the BEA SNMP agent runs as a SMUX subagent.

`-c`

Enables you to run the agent only on the master node but to still gather information from all machines. This results in a more manageable solution because it requires you to run one agent process per domain instead of one per node. In addition, it enables you to gather SNMP information from nodes with operating systems not supporting the current SNMP Agent.

When using this option, you must ensure that only one agent is started on the domain; otherwise, the results are unpredictable.

`-p` *`snmp_port`*

The *`snmp_port`* option specifies the UDP port on which the BEA SNMP agent listens for incoming SNMP packets. The `-p` option enables you to run the BEA SNMP agent on a port other than the standard SNMP port 161. This option is meaningful only when the BEA SNMP agent is running as an SNMP agent.

`-r` *`smux_port`*

Specifies the TCP port to connect to a SMUX master agent. The default is port 199. This option is meaningful only when the BEA SNMP agent is running as a SMUX subagent.

`-m` *`hostname`*
: The name of the machine where the SMUX master agent, such as the BEA SNMP Agent Integrator, is running. This option is used only when you want the BEA SNMP agent to register with a SMUX master agent on a remote machine.

`-h`
: Displays the syntax for the `tux_snmpd` command.

Stopping BEA SNMP Agent

On a Windows system, you stop BEA SNMP agents and the optional BEA SNMP Agent Integrator via the Services window. On a UNIX system, you stop BEA SNMP agents and the optional BEA SNMP Agent Integrator by entering the `stop_agent` command at the command-line prompt.

Stopping BEA SNMP Agent on a Windows System

To stop one or more BEA SNMP agents on a Windows system, follow these steps:

1. On the Windows taskbar, choose Start->Settings->Control Panel->Services (or Start->Programs->Administrative Tools->Services on a Windows 2003 system) to display the Services window.
2. In the list of Services, locate and select the installed service and click Stop to stop it.

Stopping BEA SNMP Agent on a UNIX System

To stop one or more BEA SNMP agents on a UNIX system, issue the following command:

prompt> `stop_agent` *`logical_agent_name`* `| all [`*`logical_agent_name`*`]`

For example,

prompt> `stop_agent tux_snmpd`

If you specify `all`, all SNMP agents are stopped. The name of the executable is the default logical agent name.

BEA Tuxedo Master and Non-Master Nodes

The Tuxedo SNMP agent can be installed on both Tuxedo master and non-master nodes. If the BEA Tuxedo application is down on the non-master node, SNMP Get requests addressed to the BEA SNMP agent on the non-master node may not have the latest information. For example, this

would be true if the requested information was updated on a master node after the application on the non-master node went down. Set requests to a non-master node are not permitted if the BEA Tuxedo application is down on the local node.

Some MIB groups in the Tuxedo SNMP MIB return values for all Tuxedo nodes, whereas other MIB groups return data only for the local node, as shown in the following table. Thus, if you want to manage objects whose values are local to a particular machine, you must install a copy of the BEA SNMP agent on that machine or start the BEA SNMP agent with the `-c` option on the master machine.

MIB Table/Group	Description
`tuxTwshTbl`	Run-time attributes of workstation handler (WSH) client processes.
`tuxTulogTable`	Run-time attributes of userlog files within an application.
`tuxTmsgTable`	Run-time attributes of the Tuxedo system message tables.
`tuxTqueueTable`	Run-time attributes of queues in an application.
`tuxTAppQTbl`	Attributes of application queues.
`tuxTAppQmsgTbl`	Attributes of messages stored in application queues.
`tuxTQspaceTbl`	Attributes of application queue spaces.
`tuxTQtransTbl`	Run-time attributes of transactions associated with application queue spaces.
`tuxTBridgeTbl`	Status and statistics pertaining to connections between machines making up an application.
`tuxTclientTbl`	Run-time attributes of active clients within an application.
`tuxTconnTable`	Run-time attributes of active conversations within an application.
`tuxTdeviceTbl`	Configuration and run-time attributes of raw disk slices or UNIX system files being used to store Tuxedo system device lists.
`tuxTsrvrTblExt`	Attributes of servers within an application. It is an extension of `tuxTsrvrTbl`.

MIB Table/Group	Description
tuxTranTbl	Run-time attributes of active transactions within the application.
tuxTsvcGrp	Configuration attributes of services within an application.
tuxLclIfQueueTable	Local run-time attributes of an interface for a particular CORBA server queue.
tuxLclInterfaceTable	Configuration and run-time attributes of CORBA interfaces for the local host on which BEA SNMP Agent is running.
tuxTAppQctrl	A control MIB that enables controlled access to all application queue-related MIB groups.

CHAPTER 4

Integrating BEA SNMP Agent with a Management Framework

The following sections explain how to integrate the BEA SNMP Agent into your management framework:

- Understanding the BEA SNMP Agent MIB Files
- Using BEA SNMP Agent with a Management Framework
- Integrating BEA Tuxedo Event Notifications

Understanding the BEA SNMP Agent MIB Files

BEA SNMP Agent provides the following two SNMP MIB files. `bea.asn1` and `mib.txt`. The `bea.asn1` file is used by the SNMP network management framework to set up that portion of its local MIB (on the management station) required to manage Tuxedo applications. The `mib.txt` file, which is created from the `bea.asn1` file, is used by the Tuxedo SNMP agent (`tux_snmpd`) at startup to set up its local SNMP MIB on the managed node.

Note: Only the `bea.asn1` file is of interest in the discussions that follow.

Using BEA SNMP Agent with a Management Framework

To use BEA SNMP Agent with your management framework, follow these steps:

1. Load the SNMP MIB for BEA Tuxedo into the management framework.

   The SNMP MIB defines the data types and access permissions for the various managed objects that can be accessed through BEA SNMP Agent. It also defines the event

notifications that can be generated by BEA SNMP Agent. The MIB thus provides the management framework with information it requires to manage BEA Tuxedo resources.

By default, the SNMP MIB file, `bea.asn1`, is installed in the *`tux_prod_dir`*`/udataobj/snmp/etc` directory. The MIB file must be imported into the management database of your management framework. Some management frameworks refer to this process as loading a MIB. For a list of management frameworks tested with BEA SNMP Agent, refer to the *BEA Tuxedo Release Notes*.

2. Decide what kind of information you need to meet your system management goals.

   For example, are there particular attributes of the resources you are managing that you want to monitor? Do you want to be notified when certain BEA Tuxedo system events occur?

3. Configure the management framework response to incoming BEA Tuxedo system events.

   For details, see "Integrating BEA Tuxedo Event Notifications" on page 4-2.

4. Configure polling or data collection rules on the manager for performance and fault management.

   Periodic collection of values of pertinent objects is valuable for analysis of trends. This analysis is valuable for capacity planning and load-balancing. You can also use polling to generate alarms, which is useful for fault management.

5. Optional: Define the BEA SNMP Agent Integrator polling rules.

   If you are using the BEA SNMP agent as a SMUX subagent with the BEA SNMP Agent Integrator, you might want to off-load some threshold checking to the BEA SNMP Agent Integrator. The BEA SNMP Agent Integrator generates enterprise-specific traps when the user-defined threshold is crossed. Off-loading checking of selective thresholds to the BEA SNMP Agent Integrator reduces the network bandwidth consumed by the management framework's polling activities.

6. Set up and start the agents.

   The procedure for setting up and starting the BEA SNMP agent is described in "Setting Up BEA SNMP Agent on a Managed Node" on page 3-1.

Integrating BEA Tuxedo Event Notifications

To integrate the BEA Tuxedo system event traps with your management framework, follow these steps:

1. Make sure the Tuxedo EventBroker server (`TMSYSEVT`) is running for the domain being managed. For more information, see "Setting Up BEA SNMP Agent on a Managed Node" on page 3-1.

   BEA SNMP Agent will not receive event notifications unless the EventBroker server (`TMSYSEVT`) is running. For information on the Tuxedo EventBroker server, see reference page `TMSYSEVT(5)` in *BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes Reference*.

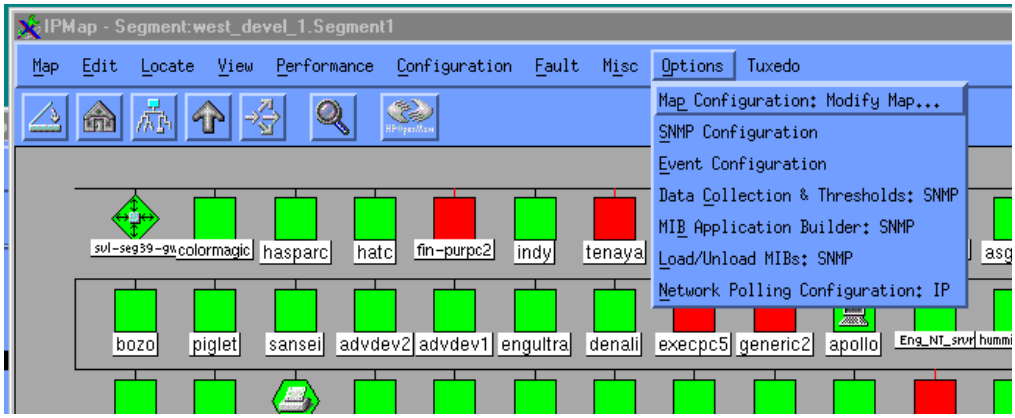
2. Modify the `TRAP_HOST` entry in the BEA SNMP Agent `beamgr.conf` configuration file to specify the location of the management machine that is to be the destination for traps generated by the agent. For more information, see "Setting Up BEA SNMP Agent on a Managed Node" on page 3-1.

3. If you have not already done so, load the BEA SNMP Agent `bea.asn1` MIB file into your management framework.

   For example, on HP OpenView Network Node Manager, see the following display and do the following:

   a. Select Options–>Load/Unload MIBs: SNMP.

Figure 4-1 Selecting Load/Unload MIBs in HP OpenView



   b. Select Load.

   c. Specify the path to the BEA SNMP Agent configuration file (`bea.asn1`). By default, this file is installed in:

tux_prod_dir\udataobj\snmp\etc\bea.asn1 (Windows)
tux_prod_dir/udataobj/snmp/etc/bea.asn1 (UNIX)

d. Select OK.

4. Configure the management framework to take the appropriate action in response to incoming Tuxedo SNMP traps.

   You might want to change the way in which Tuxedo SNMP traps are displayed on your management console, or the actions that the management framework takes in response to specified events. For example, you might choose to ignore some routine informational notifications. For example, to view the event configuration on HP OpenView, do the following:

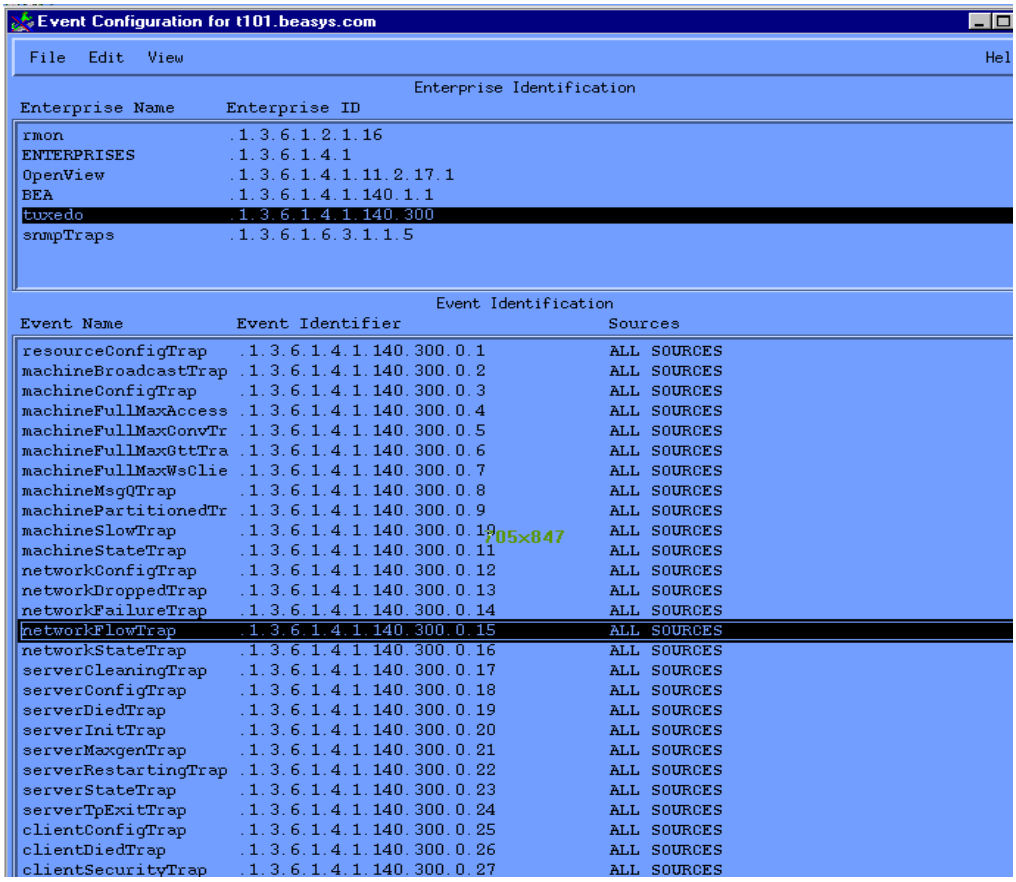
   a. Select Options→Event Configuration.

   b. Select the enterprise `tuxedo`.

   c. Select an event type, such as `networkFlowTrap` in HP Open View.

   d. Customize the management framework response to incoming events of that type. Select Edit→Modify Event in OpenView, which invokes the Event Configuration window shown in the following display.

Figure 4-2 HP OpenView Event Configuration Window



You can modify the event configuration to ignore an event, or generate a pop-up notification or run a program or script when the event is received. You might also want to create a separate category for Tuxedo events, as shown in the preceding display.

Retrieving or Modifying Object Values When Managing Multiple Domains

Monitoring of multiple BEA Tuxedo domains is done by running a separate Tuxedo SNMP agent for each domain being monitored. These agents must be run as SMUX subagents under the BEA SNMP Agent Integrator.

When more than one Tuxedo SNMP agent is running as a SMUX subagent on a node, then SNMP manager Set or Get requests to an agent must be addressed using a community of the form:

community@logical_agent_name

For example:

public@payrollagent

In this example payrollagent is a logical agent name that identifies the agent to which the request is to be forwarded by the BEA SNMP Agent Integrator.

Integrating Events Generated by BEA SNMP Agent Integrator Polling

The BEA SNMP Agent Integrator can be used to poll BEA Tuxedo objects, or other managed resources. To integrate the BEA SNMP Agent Integrator threshold-checking activity with the management framework, perform the following steps:

1. Set up the BEA SNMP Agent Integrator polling rules.

   A polling rule is defined by a RULE_ACTION entry in the BEA SNMP Agent beamgr.conf configuration file.

2. Configure the management framework to recognize the events generated by the BEA SNMP Agent Integrator.

   For example, in HP OpenView, you can add a new event type by doing the following:

   a. Select Options Event Configuration.

   b. Select beaSystemDescr.

   c. Select Edit Add Event.

      In the window that is invoked you would use the following as the event number:

      .1.3.6.1.4.1.140.1.1.0.*specific_trap_number*

3. Configure the management framework to respond appropriately to incoming BEA SNMP Agent Integrator events.

For more information, see "Using the BEA SNMP Agent Integrator for Polling" on page 7-1.

CHAPTER 5

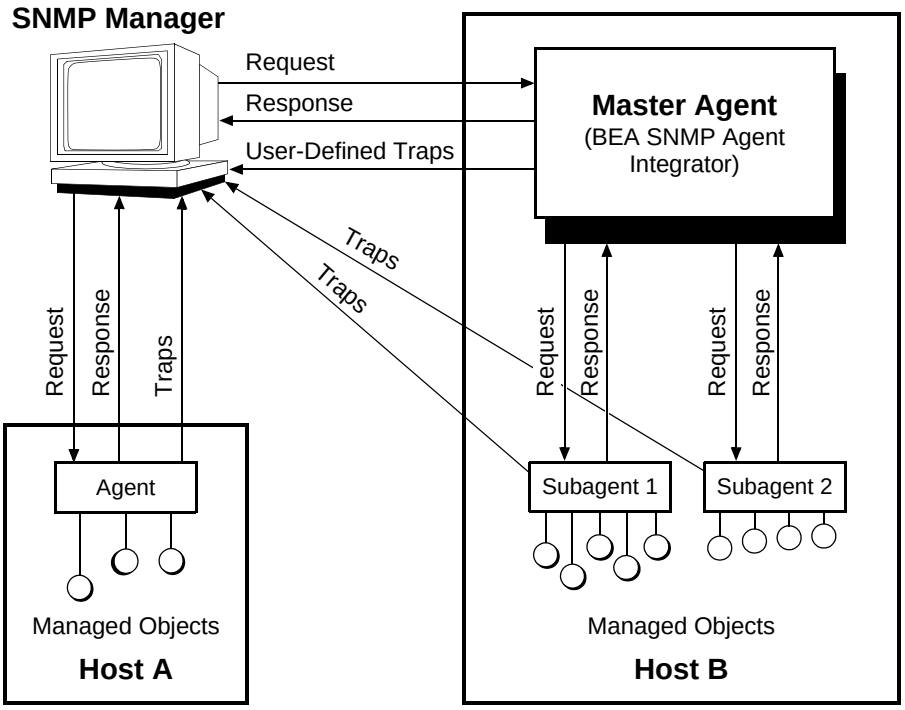
Setting Up the BEA SNMP Agent Integrator

The following sections provide information about the BEA SNMP Agent Integrator and describe the procedure to install the BEA SNMP Agent Integrator:

About the BEA SNMP Agent Integrator

The BEA SNMP Agent Integrator enables multiple agents and subagents from any vendor to cooperate in managing diverse hardware and software components on a single host. It makes possible the extended SNMP manager/agent model shown in the following figure.

Figure 5-1 SNMP Manager/Agent Model



The BEA SNMP Agent Integrator enables you to:

- Run multiple peer SNMP agents on a single managed node.

  All communication between the agents and the SNMP manager is handled through the BEA SNMP Agent Integrator master agent.

- Off-load polling from the management station to reduce network traffic and reduce load on the network manager.

  User-defined rules enable the BEA SNMP Agent Integrator to check for the occurrence of significant system events and send alarms or execute programs when the events are detected. Communication over the network occurs only when an event is detected or when polling activity from the manager is started or stopped.

- Manage system resources using multiple SNMP multiplexing (SMUX) subagents.

The BEA SNMP Agent Integrator uses the SMUX protocol defined in RFC 1227 to respond to the SMUX subagents and fans out requests from an SNMP-compliant system or network management station to the appropriate subagent.

- Manage system resources using any other master agent/subagent protocol, such as Distributed Program Interface (DPI), where the master agent responds to SNMP management requests.

  The DPI master agent uses SNMP to respond to requests from network managers. The DPI master agent can, therefore, be configured to communicate through the BEA SNMP Agent Integrator in the same manner as other peer SNMP agents.

- Coordinate communication between an SNMP manager and multiple SNMP agents on multiple network nodes.

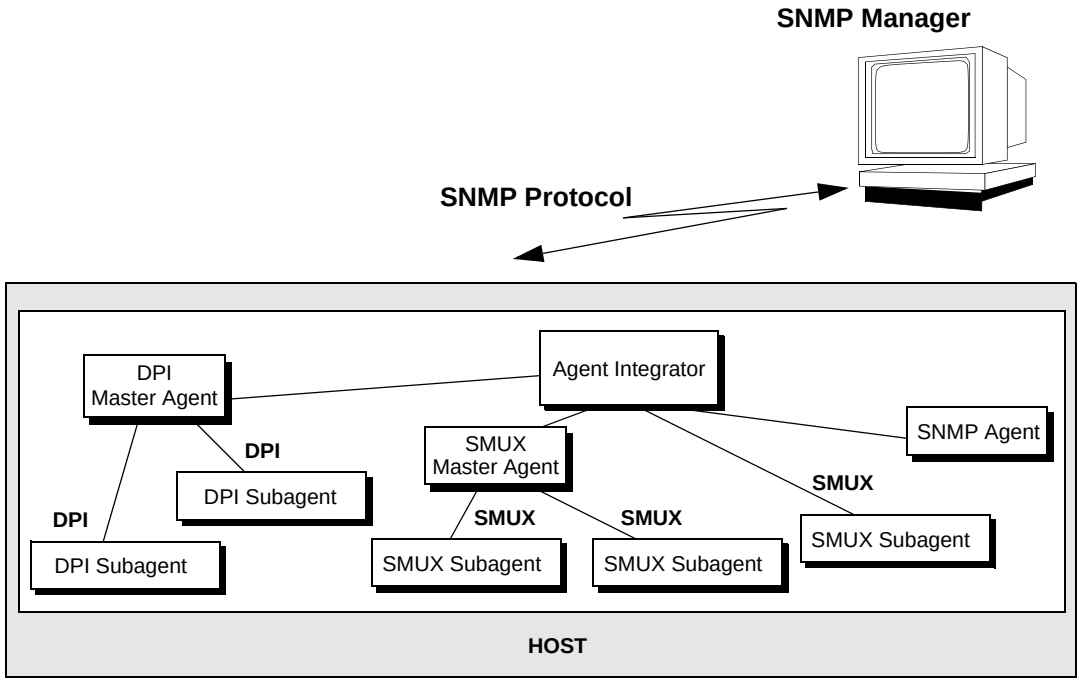
  This feature is particularly useful for off-loading polling to the BEA SNMP Agent Integrator to manage a distributed system whose components are spread over a number of computers. To the BEA SNMP Agent Integrator, the managed resources appear as if they are on a single computer. For more information, see "Using the BEA SNMP Agent Integrator for Polling" on page 7-1.

- Listen on UDP port 161 and handle all communications with the SNMP manager.

  SNMP requests received by the BEA SNMP Agent Integrator are fanned out to the appropriate peer SNMP agent or SMUX subagent, and the responses from the agents or subagents are passed on to the SNMP manager by the BEA SNMP Agent Integrator.

The following figure summarizes how the BEA SNMP Agent Integrator master agent can control an assortment of master agents, SNMP agents, and SMUX subagents.

Figure 5-2 BEA SNMP Agent Integrator Master/Subagent Architecture



The various agents and subagents all appear to the SNMP manager as a single SNMP agent. The BEA SNMP Agent Integrator acts as a proxy for the SNMP manager.

The BEA SNMP Agent Integrator can run on the same node with any master agent/subagent architecture as long as the master agent uses SNMP to respond to management requests.

Configuring the BEA SNMP Agent Integrator

The BEA SNMP Agent Integrator uses the following environment variables:

`BEA_SMUX_PASSWD`
: Indicates the password that a SMUX subagent must use when re-establishing communication with the BEA SNMP Agent Integrator.

BEA_PEER_MAX_TRIES
: Indicates the number of times the BEA SNMP Agent Integrator retries sending an SNMP request to peer SNMP agents when no response is received within the established timeout interval.

BEA_PEER_MAX_WAIT
: Modifies the default time interval that the BEA SNMP Agent Integrator waits for a reply to a request sent to a SMUX subagent or an SNMP peer agent. This value can also be set by adding a BEA_PEER_MAX_WAIT entry to the BEA SNMP Agent configuration file.

  If this environment variable is not set, and there is no BEA_PEER_MAX_WAIT entry in the configuration file, the default is three seconds. For peer SNMP agents, the default timeout value can be overridden for individual SNMP agents using the timeout parameter in NON_SMUX_PEER entries in the BEA SNMP Agent beamgr.conf configuration file.

  (Note that "peer SNMP agent" and "non-SMUX peer agent" are identical terms. A peer SNMP agent, or a non-SMUX peer agent, is an SNMP agent running under the BEA SNMP Agent Integrator.)

BEA_SM_BEAMGR_CONF
: Specifies the absolute path to the BEA SNMP Agent beamgr.conf configuration file, ending with the filename beamgr.conf.

Once you have set up BEA SNMP Agent according to the instructions in "Setting Up BEA SNMP Agent on a Managed Node" on page 3-1, perform the following step to set up and use the BEA SNMP Agent Integrators:

1. Indicate the managed objects (if any) that are available from peer SNMP agents.

   The peer SNMP agents can be on the same managed node (IP address) as the BEA SNMP Agent Integrator, or they can be on remote nodes. Access to the objects managed by the peer SNMP agents is defined through NON_SMUX_PEER entries in the beamgr.conf configuration file. Each entry defines or moves a branch of the OID tree that is accessible via that agent. This task is described in "Using Multiple SNMP Agents" on page 6-1.

2. Indicate the managed objects that are available (if any) through Distributed Program Interface (DPI) master agents.

   Because a DPI master agent speaks SNMP, it appears to the BEA SNMP Agent Integrator as just another peer SNMP agent. Setting up access to DPI subagents is thus done the same way as setting up access to peer SNMP agents, as described in Step 1.

3. Modify the management scope of SMUX subagents, if desired.

   You can modify a SMUX subagent's management scope—for example, to avoid conflicts with other agents—by specifying OID_CLASS entries in the beamgr.conf configuration

file. By default, a SMUX subagent automatically indicates the section of the OID tree for which it is responsible when it registers with the BEA SNMP Agent Integrator master agent. The syntax for `OID_CLASS` entries is defined in "Configuration Files" on page 9-1.

4. Define local polling rules and the actions to be taken by the BEA SNMP Agent Integrator if user-defined thresholds are crossed.

   This step is necessary only if you want to use the BEA SNMP Agent Integrator to off-load polling from the management station. Polling rules are defined through `RULE_ACTION` entries in the `beamgr.conf` configuration file. Polling is automatically active when the BEA SNMP Agent Integrator starts. BEA SNMP Agent Integrator local polling can be de-activated or re-activated from a management station using SNMP Set commands. BEA SNMP Agent Integrator polling rules, and how to start and stop polling, are described in "Using the BEA SNMP Agent Integrator for Polling" on page 7-1.

5. Configure your SNMP management framework for the BEA SNMP Agent Integrator. For more information, see "Integrating BEA SNMP Agent with a Management Framework" on page 4-1.

   Configure the management system for BEA SNMP Agent Integrator traps. Some configuration is required on your SNMP-compliant management framework to make use of SNMP trap notifications that are generated by the BEA SNMP Agent Integrator.

   The exact set of steps you need to perform vary depending upon which management system you are using. Typically, some configuration or mapping is required to get the management system to perform a desired action (such as turning an icon red) when a trap is received. Consult your management system documentation for specific instructions.

6. Modify other entries in the configuration file (if desired)

   You may want to modify the following fields in the BEA SNMP Agent `beamgr.conf` configuration file:

   - `SYS_DESCR`
   - `SYS_CONTACT`
   - `SYS_LOCATION`
   - `SYS_SERVICES`

These entries are supported by the BEA SNMP Agent Integrator in the MIB-II `snmp` group.

If you are using a BEA SNMP agent as a SMUX subagent to manage BEA Tuxedo applications, configure the BEA SNMP Agent Integrator timeout to at least 30 seconds. To do this, add a `BEA_PEER_MAX_WAIT` entry to the BEA SNMP Agent `beamgr.conf` configuration file as follows:

```
BEA_PEER_MAX_WAIT 30
```

Another way you can set the timeout value is to set the environment variable `BEA_PEER_MAX_WAIT` to 30. For C shell on UNIX systems, for example, use this command:

```
prompt> setenv BEA_PEER_MAX_WAIT 30
```

Starting the BEA SNMP Agent Integrator and Subagents on a Windows System

To start the BEA SNMP Agent Integrator and SMUX subagents on a Windows system, follow these steps:

1. Start the BEA SNMP Agent Integrator and SMUX subagents from the Services window.

   On the Windows taskbar, choose Start->Settings->Control Panel->Services (or Start->Programs->Administrative Tools->Services on a Windows 2003 system) to display the Services window.

2. Locate each of the installed services. The BEA SNMP Agent Integrator is installed as a Windows Service named `snmp_integrator`. It should be started before the SMUX subagents.

   A BEA SNMP agent (`tux_snmpd`) is installed as a Windows service named `tux81_snmpd` or some other logical agent name if additional BEA SNMP agents were installed.

3. Click Start. There may be a short delay as each service is initiated.

4. Start any other SMUX subagents.

Note: For any `tux_snmpd` process started as a non-SMUX peer agent (`-s` option specified at startup), you must start that process *before* starting the BEA SNMP Agent Integrator. The starting order for BEA SNMP Agent is as follows: start all non-SMUX peer agents, then the BEA SNMP Agent Integrator, and then all SMUX subagents.

Starting the BEA SNMP Agent Integrator and Subagents on a UNIX System

To start the BEA SNMP Agent Integrator and SMUX subagents on a UNIX system, log in as `root` and start the following programs in the specified order:

- `snmp_integrator`
- `tux_snmpd`

Note: For any `tux_snmpd` process started as a non-SMUX peer agent (`-s` option specified at startup), you must start that process *before* starting the BEA SNMP Agent Integrator. The starting order for BEA SNMP Agent is as follows: start all non-SMUX peer agents, then the BEA SNMP Agent Integrator, and then all SMUX subagents.

Stopping the BEA SNMP Agent Integrator and Subagents on a Windows System

To stop the BEA SNMP Agent Integrator or one or more subagents on a Windows system, follow these steps:

1. On the Windows taskbar, choose Start->Settings->Control Panel->Services (or Start->Programs->Administrative Tools->Services on a Windows 2003 system) to display the Services window.
2. In the list of Services, locate and select the installed service and click Stop to stop it.

Stopping the BEA SNMP Agent Integrator and Subagents on a UNIX System

To stop the BEA SNMP Agent Integrator or one or more subagents on a UNIX system, issue the following command:

prompt> `stop_agent` *`logical_agent_name`* `| all [`*`logical_agent_name`*`]`

For all SNMP agents other than `tux_snmpd`, the *`logical_agent_name`* is always the name of the executable. If you specify `all`, all SNMP agents are stopped.

CHAPTER 6

Using Multiple SNMP Agents

The following sections describe how to use the BEA SNMP Agent Integrator component with other SNMP agents:

- Configuring the BEA SNMP Agent Integrator for Use with Multiple SNMP Agents
- Integrator Access to Managed Objects
- SNMP Agents on Multiple Nodes

Configuring the BEA SNMP Agent Integrator for Use with Multiple SNMP Agents

A BEA SNMP agent started as an SNMP agent—started with the `-s` option—and running under the BEA SNMP Agent Integrator is known as a *non-SMUX peer agent*. A non-SMUX peer agent must be started before starting the BEA SNMP Agent Integrator.

Each non-SMUX peer agent running on the managed node must have one or more `NON_SMUX_PEER` entries in the BEA SNMP Agent `beamgr.conf` configuration file. The syntax of `NON_SMUX_PEER` entries is described in "Configuration Files" on page 9-1.

Note: "Non-SMUX peer agent" and "peer SNMP agent" are identical terms. A non-SMUX peer agent, or a peer SNMP agent, is an SNMP agent running under the BEA SNMP Agent Integrator.

Each `NON_SMUX_PEER` entry lists the port that the BEA SNMP Agent Integrator is to use in communicating with the non-SMUX peer agent. When the agent is started, it must be configured

to listen on the port assigned to it in the NON_SMUX_PEER entry (or entries) in the beamgr.conf file.

Integrator Access to Managed Objects

Each NON_SMUX_PEER entry for a non-SMUX peer agent lists a branch of the OID tree for which the agent is responsible. If agent A is listed as responsible for a certain branch of the OID tree, then management requests for objects within that branch will be passed on to agent A by the BEA SNMP Agent Integrator.

Example

For example:

```
NON_SMUX_PEER 2001 snmp .1.3.6.1.2.1.1,ro
NON_SMUX_PEER 2002 squid .1.3.6.1.4.1.141 .1.3.6.1.4.1.145
NON_SMUX_PEER 161 * .1.3.6.1.4.1.140 .1.3.6.1.4.1.145
```

The first entry tells the BEA SNMP Agent Integrator to look for an SNMP agent at port 2001. All the requests from the BEA SNMP Agent Integrator to this SNMP agent will use snmp as the community. The agent supports the subtree .1.3.6.1.2.1.1, and is available for read-only commands.

The second entry tells the BEA SNMP Agent Integrator to look for an SNMP agent at UDP port 2002. All the requests from the BEA SNMP Agent Integrator to this SNMP agent will use squid as the community. The agent supports the subtrees .1.3.6.1.4.1.141 and .1.3.6.1.4.1.145. Because no access option is specified, both subtrees default to read-write.

The third entry lists an agent at UDP port 161. The asterisk means that the BEA SNMP Agent Integrator uses the community string supplied by the management station. The agent supports two subtrees: .1.3.6.1.4.1.140 and .1.3.6.1.4.1.145. The subtree entries list no access information, so access defaults to read-write.

Note: The SMUX protocol provides that SMUX subagents automatically register the sections of the OID tree that they support with the master agent. Hence, it is not necessary to add specific configuration file entries to specify which sections of the OID tree are accessible from the SMUX subagents. However, this default behavior can be overridden using a configuration file OID_CLASS entry. For more information about the OID_CLASS entry, see "Configuration Files" on page 9-1.

Assigning Priority for Conflicting Agents

If two agents, A and B, conflict in the portions of the OID tree for which they are responsible, then the NON_SMUX_PEER entries that define these responsibilities must assign a distinct priority to the two agents. The BEA SNMP Agent Integrator thus refers requests for objects in the overlapping area to the agent with the lowest priority number. (The lower the priority number, the higher the priority.) For example:

```
NON_SMUX_PEER 2008 * .1.3.6.1.2.1.4,rw,8
NON_SMUX_PEER 2009 * .1.3.6.1.2.1.4,ro,5
```

In this example, the agents on UDP port 2008 and UDP port 2009 both support the MIB-II ip group. Thus, both agents support the ipAddrTable (object .1.3.6.1.2.1.4.20). Because the agent at port 2009 has a higher priority (5 is a higher priority than 8), the BEA SNMP Agent Integrator calls it for management requests for the ipAddrTable. Notice that this entry specifies read-only access. The other entry specifies read-write access, but because it has a lower priority, it is completely ignored for ipAddrTable requests.

The syntax of the NON_SMUX_PEER entry in the configuration file is described in detail, with examples, in "Configuration Files" on page 9-1.

SNMP Agents on Multiple Nodes

The BEA SNMP Agent Integrator can be used as a proxy agent for the management station, conducting polling of SNMP agents on a number of machines, and sending an enterprise-specific trap when a user-defined condition is encountered. This is useful when resources of a single distributed system are spread over a number of machines.

From the perspective of the BEA SNMP Agent Integrator, the resources managed by these multiple agents are viewed as though they were on a single machine. The polling capability of BEA SNMP Agent Integrator is described in "Using the BEA SNMP Agent Integrator for Polling" on page 7-1. Polling of agents on multiple machines assumes that NON_SMUX_PEER entries have been defined for these SNMP agents in the BEA SNMP Agent beamgr.conf configuration file. The following is an example where the BEA SNMP Agent Integrator is used as a proxy agent for communication with SNMP agents in a single subnet:

```
NON_SMUX_PEER 206.189.39.86.161 seahorse .1.3.6.1.2.1.4,rw,8
NON_SMUX_PEER 206.189.39.204.161 * \
.1.3.6.1.2.1.4.20,ro,5 .1.3.6.1.2.1.2
```

In this example, the SNMP agent on machine 206.189.39.86 communicates with the BEA SNMP Agent Integrator on port 161, uses a community string of seahorse, and is responsible

for the MIB-II `ip` group. The SNMP agent on machine `206.189.39.204` also communicates with the BEA SNMP Agent Integrator on port 161, using the community string passed from the SNMP manager.

The machine at IP address `206.189.39.204` is responsible for the SNMP interfaces group (`.1.3.6.1.2.1.2`) as well as the `ipAddrTable` (`.1.3.6.1.2.1.4.20`). The machine at IP address `206.189.39.86` is responsible for the MIB-II `ip` group that includes the `ipAddrTable`. Even though these agents are on physically distinct network nodes, there is still a conflict in the responsibilities of these two agents, as far as the integrator is concerned, because the integrator views the resources managed by the agents specified in its configuration file as though they were all on a single machine. Thus, the BEA SNMP Agent Integrator only consults the machine at `206.189.39.204` for management requests for the `ipAddrTable` because the priority number for this entry is lower than the priority number for the machine at `206.189.39.86`. However, all other `ip` group requests and requests for the interfaces group are sent to `206.189.39.86`.

This feature of the BEA SNMP Agent Integrator is particularly useful when different functions of a distributed system are located on different machines, each with its own SNMP agent. The limitation to non-overlapping OID tree branches should not be a significant problem when different managed resources are located on the distinct nodes. If the same type of managed resource is located on multiple machines, then multiple BEA SNMP Agent Integrators can be used to manage those resources.

CHAPTER 7

Using the BEA SNMP Agent Integrator for Polling

The following sections describe how to use the BEA SNMP Agent Integrator as a proxy for the management station to poll locally on the managed node:

Overview of Polling

Polling is the activity of checking the value of an attribute of a managed resource at a specific interval. To track faults in critical system components or applications, management systems use polling to determine whether attributes of the managed resource have crossed a significant threshold. However, direct polling by a management station becomes increasingly inefficient as the number of components being polled increases—the load on network bandwidth increases as does the load on the management station itself.

The BEA SNMP Agent Integrator can be configured to act as a proxy for the manager and do the polling locally on the managed node. The load on the management station is reduced and less network bandwidth is consumed by off-loading polling to distributed integrator agents.

You can specify the threshold to check, and polling can be activated during BEA SNMP Agent Integrator startup or by an SNMP Set request from the management station. The BEA SNMP Agent Integrator sends an enterprise-specific SNMP trap when the threshold is crossed. To indicate the cause of the alarm, you can configure a specific-trap type number to be sent in the trap generated when a given threshold is crossed. Communication between the manager and BEA SNMP Agent Integrator occurs only when the manager sends a Set request to de-activate (or re-activate) the polling, or when the BEA SNMP Agent Integrator sends an SNMP trap if it detects the specified event in the managed resource. The BEA SNMP Agent Integrator can also be configured to execute a script or program when a threshold is crossed.

Procedure for Setting Up Local Polling

The steps in using the BEA SNMP Agent Integrator for local polling can be summarized as follows:

1. Decide which resources you want to monitor.

   The attributes of the resource that you want to monitor must be defined as MIB objects. These MIB objects must be supported by an agent or subagent that has been installed on the managed node.

2. Make the managed resource accessible to the BEA SNMP Agent Integrator.

   The BEA SNMP Agent Integrator must know how to access the managed object. This means the object identifier for that object must lie within branches of the OID tree that are known to the BEA SNMP Agent Integrator.

   If the managed object you want to monitor is supported by a SMUX subagent that has been installed on the managed node, the subagent automatically registers its section of the OID tree with the BEA SNMP Agent Integrator when the subagent is started. This can be modified using `OID_CLASS` entries in the BEA SNMP Agent configuration file, as described in "Configuration Files" on page 9-1.

   For peer SNMP agents (or DPI or SMUX master agents), you must define the segments of the OID tree supported by those agents in `NON_SMUX_PEER` entries in the BEA SNMP Agent configuration file. This is described in "Configuring the BEA SNMP Agent Integrator for Use with Multiple SNMP Agents" on page 6-1.

   The BEA SNMP Agent Integrator directly supports the following MIB groups: MIB-II `system` and `snmp` groups, the SMUX MIB, and the BEA SNMP Agent `beaintAgtTable` in the SNMP agent MIB.

3. Define polling instructions for the BEA SNMP Agent Integrator

This task consists of two main subtasks:

- Define the desired threshold.
- Specify the action to take if the threshold is crossed.

Each polling instruction for the SNMP Integrator is called a *rule*. Rules are defined under the `RULE_ACTION` entry in the BEA SNMP Agent `beamgr.conf` configuration file. You can use your favorite text editor to modify this file. (For information on how to create rules, see "Creating New Polling Rules" on page 7-15.) Rules are explained in "BEA SNMP Agent Integrator Rules" on page 7-3, while "Configuration Files" on page 9-1 provides the complete syntax for the `RULE_ACTION` entries.

4. Configure your SNMP management system for BEA SNMP Agent Integrator traps

   When a polling threshold is crossed, the BEA SNMP Agent Integrator sends an enterprise-specific SNMP trap notification to the destinations specified by the `TRAP_HOST` entries in the BEA SNMP Agent configuration file. Some configuration is required on your SNMP-compliant management system to make use of the traps that are thus generated. The exact set of steps you need to perform varies, depending upon which management system you are using. Typically, some configuration or mapping is required to get the management system to perform a desired action when a trap is received. For example, you might want the management system to turn an icon red when a trap is received. Consult your management system documentation for specific instructions.

5. Start the BEA SNMP Agent Integrator polling.

   The BEA SNMP Agent Integrator begins executing all valid polling rules when it is started. For more information, see "Starting BEA SNMP Agent Integrator Polling Activity" on page 7-14.

6. De-activate or re-activate BEA SNMP Agent Integrator polling, when desired.

   Polling rules are available as MIB objects; thus you can de-activate or re-activate polling from the management station by means of an SNMP Set request. This topic is described in "Starting BEA SNMP Agent Integrator Polling Activity" on page 7-14.

BEA SNMP Agent Integrator Rules

A BEA SNMP Agent Integrator rule is a polling instruction for the SNMP BEA SNMP Agent Integrator, and it consists of the following parts:

- A unique name for the rule (can be a maximum of eight characters long)

- A *condition* (or threshold) for which the integrator is to check. (This is described further in "Conditions" on page 7-4.)
- An *action* to take if the specified threshold is crossed. (This is described in "States and Transitions" on page 7-11.)
- A *polling frequency* (specified in seconds), that is, the time delay between each access of the specified object value

Conditions

When the BEA SNMP Agent Integrator polls, it checks to determine if a specified condition holds. A *condition* is defined as a relationship between an object (specified by its object identifier) and a value.

Relations for Defining Conditions

The condition obtains (the threshold is crossed) if and only if the specified relation holds between the object and the value. For example, the relation "greater than" defines the following condition: "disk capacity in use greater than 90 percent."

In this case, the condition holds (evaluates to true) if the object (percentage of disk capacity in use) has a value that is greater than 90. (In this example, the condition is described in English, not the actual code used to define BEA SNMP Agent Integrator polling rules.)

Any of the relations shown in the following table can be used to define conditions.

Symbol	Meaning
==	Is identical to
!=	Is not identical to
<	Is less than (for numeric values is a substring of (for strings)
>	Is greater than (for numeric values) contains (for strings)
<=	Is less than or equal to
>=	Is greater than or equal to

Polling with a SMUX Subagent

For example, suppose that you want the BEA SNMP Agent Integrator to check if any server group is inactive. The state of the server group is represented by the `tuxTgroupState` object in the `beaSysPerf` group. The BEA SNMP Agent component uses SNMP multiplexing (SMUX) protocol to talk to the BEA SNMP Agent Integrator, and to supply the value of the object to the BEA SNMP Agent Integrator on the same machine.object in the `beaSysPerf` group. This subagent uses SMUX protocol to talk to the BEA SNMP Agent Integrator.

Examples

You can use the following condition to define a polling rule for the BEA SNMP Agent Integrator:

```
(VAL(.1.3.6.1.4.1.140.300.4.1.1.4.*) !=1)
```

The expression `VAL()` is used to obtain the value of the `tuxTgroupState` object. The specified condition is obtained if any server group is not active (`!=1`). In this example, the initial dot indicates that this is an absolute OID; that is, the path to the `tuxTgroupState` object is `.1.3.6.1.4.1.140.300.4.1.1.4.` The asterisk (`*`) wildcard for the instance index indicates that the condition is satisfied if any `tuxTgroupState` is not equal to 1. Use 0 as the instance index for scalar objects. For more information, see "Instance Indexes" on page 7-10.

The following example is of a BEA SNMP Agent Integrator rule that uses the condition previously specified:

```
RULE_ACTION grpState 60 if (VAL(140.300.4.1.1.4.*) !=1)
   {TRAPID_ERR=300}
```

In this example, `grpState` is the name of the rule. The BEA SNMP Agent Integrator checks the server group state every 1 minute (60). If any value of `tuxTgroupState` is not equal to 1, `TRAPID_ERR=300` instructs the BEA SNMP Agent Integrator to generate an enterprise-specific trap with a specific-trap type number of 300.

Note: (The MIB objects whose values the BEA SNMP Agent Integrator can obtain depend on the MIB objects supported by the agents or subagents that the BEA SNMP Agent Integrator is managing. In the previous example, the BEA SNMP Agent Integrator can poll for the `tuxTgroupState` object value only if the `tux_snmpd` subagent is running on the managed node. The MIB objects that the BEA SNMP Agent Integrator can access through a peer SNMP agent depend on the `NON_SMUX_PEER` entries in the BEA SNMP Agent configuration file, as explained in "Using Multiple SNMP Agents" on page 6-1.

Polling with SNMP Peer Agents

The BEA SNMP Agent Integrator can also obtain MIB object values from SNMP peer (non-SMUX peer) agents on either the same machine or other machines in the network. For example, suppose that a peer SNMP agent supports the MIB-II `interfaces` group. If so, you might want the BEA SNMP Agent Integrator to check if a physical interface is not operational. This feature of the interface is represented by the `ifOperStatus` object in the `ifTable` in the MIB-II `interfaces` group. In this case, you want to know whether the value of `ifOperStatus` is not equal to 1. (An interface is operational if its `ifOperStatus` value is 1.) If you want to check the `ifOperStatus` value for the first interface on the machine, you could use the following condition:

```
(VAL(.1.3.6.1.2.1.2.2.1.8.1) != 1)
```

This condition holds if, and only if, the first interface in the `ifTable` is operational. The last numeral, `1`, specifies the instance index—the first interface entry in the table.

If the condition is satisfied, you want the BEA SNMP Agent Integrator to take some action. For example, if the `ifOperStatus` value for an interface is not 1 (that is, the interface is not up), you might want the BEA SNMP Agent Integrator to notify the management station. To do this, you can specify that the BEA SNMP Agent Integrator send an enterprise-specific SNMP trap to the management station with a special specific-trap value that identifies the cause of the trap to you (the system administrator).

Instead of requesting this notification if a specific interface (such as the first one in the `ifTable`) is down, you might want to be notified if *any* of the interfaces is down.

Here is an example of a rule entry that would do this:

```
RULE_ACTION checkIf 120
   if (VAL(.1.3.6.1.2.1.2.2.1.8.*) != 1) {TRAPID_ERR=300}
```

In this example, `checkIf` is the name given to this particular rule. The value `120` indicates that the BEA SNMP Agent Integrator should check the interface every two minutes. By using the asterisk wildcard for the instance index, the condition is satisfied if any interface in the `ifTable` has an `ifOperStatus` not equal to 1; that is, all instances are checked. If the value of the OID is not equal to 1 (the interface is not up) for any instance, an enterprise-specific trap is sent with a specific trap ID of 300.

Note: This rule only causes a trap to be generated when the BEA SNMP Agent Integrator first detects that an interface is down. If the interface continues to be down, it does not generate additional traps.

Use of Logical Operators in Conditions

Conditions are of two types, simple and complex. A *simple* condition consists of a relation between a managed object and a value. All of the examples in the previous sections have been simple conditions.

You can use the logical operators AND, OR, and NOT to define *complex* conditions. For example, if A and B are two simple conditions, you can specify a complex condition where *both* A and B occur.

The symbols in the following table can be used to define complex conditions.

Symbol	Meaning
`!(condition_A)`	Logical negation. The threshold is crossed if and only if *condition_A* does not hold.
`(condition_A || condition_B)`	Logical disjunction. The threshold is crossed if and only if either *condition_A* or *condition_B* obtain.
`(condition_A && condition_B)`	Logical conjunction. The threshold is crossed if and only if both *condition_A* and *condition_B* obtain.

Scenario for Using a Complex Condition

For example, you might not want the BEA SNMP Agent Integrator to send an alarm when `ifOperStatus` is not up for an interface if you have taken that interface down for repair. In that case, you could define a rule that asks the BEA SNMP Agent Integrator to determine if two conditions hold: `ifOperStatus` is not up AND `ifAdminStatus` is up. That is, you want to be notified if the interface *should* be up but is not.

Note: The MIB objects whose values the BEA SNMP Agent Integrator can obtain depend on the MIB objects supported by the agents or subagents that the BEA SNMP Agent Integrator is managing.

Sample Code for this Scenario

To do this, you might modify your `checkIf` rule as follows:

```
RULE_ACTION checkIf 60
   if ((VAL(.1.3.6.1.2.1.2.2.1.8.*) != 1) &&
```

```
(VAL(.1.3.6.1.2.1.2.2.1.7.*) == 1))
   {TRAPID_ERR=301}
```

How this Rule Works

In this example, the BEA SNMP Agent Integrator checks the interfaces every minute (`60`) and generates an enterprise-specific trap, with a specific trap value of 301, if any of the interfaces is not up (`ifOperStatus` not equal to 1) but has an `ifAdminStatus` value of up (that is, the interface should be up but it is not).

Note: This rule causes this trap to be generated only when the condition first evaluates to true. As long as the interface continues in the same state, a new trap is not generated.

Data Types for Defining Conditions

The syntax for a simple condition is as follows:

```
(VAL(oid) relation value)
```

where

`relation`

Is one of the relations described in the table titled "Relations for Defining Conditions" on page 7-4.

`oid`

Is specified in one of the formats described in "Specifying Object Identifiers in Conditions" on page 7-9.

`value`

Can be one of the following data types:

- integer
- string
- IP address (in the form *number1.number2.number3.number4*)
- Object identifier, surrounded by single quotes (‘). The OID should be specified exactly as returned by the agent managing that object.

Specifying Object Identifiers in Conditions

In defining polling conditions, the object identifier (OID) must be specified numerically, not using textual symbols (other than MIB II [`mib-2`] or `enterprises` as indicated in the following list). One of the following formats can be used to specify the object identifier:

- An absolute object identifier, that is, the full path to the object is specified from the root of the OID tree. An initial dot is used to indicate that the path starts at root, (for example, `.1.3.6.1.2.1.1.1.0`). Note that the trailing zero in this example is the instance index.
- A relative OID under the MIB-II branch can be specified in the form:

  `mib-2.number.number ...`

  When the reserved word `mib-2` appears as the leading sub-oid, `.1.3.6.1.2.1` is assumed to be prefixed to the rest of OID. For example:

  `mib-2.1.1.0`

  represents the absolute OID:

  `.1.3.6.1.2.1.1.1.0`
- A relative OID under the enterprises branch can be specified in the form:

  `enterprises.number.number ...`

  When the reserved word `enterprises` appears as the leading sub-oid, `.1.3.6.1.4.1` is assumed to be prefixed to the rest of OID. For example:

  `enterprises.140.1.0`

  represents the absolute OID:

  `.1.3.6.1.4.1.140.1.0`
- A relative OID under the enterprises branch can also be specified in purely numeric form:

  `number.number.number ... ,`

  If there is no leading "." and the OID starts with a number, `.1.3.6.1.4.1` is assumed to be prefixed to the rest of OID. For example:

  `140.1.1.0`

  represents the absolute OID:

  `.1.3.6.1.4.1.140.1.1.0`

Instance Indexes

Columnar objects are used to represent a column of a tabular MIB group. Columnar objects, accordingly, can have multiple instances. To specify an instance, the index is appended to the rest of the OID. If the index is a single attribute, the last number in an OID is used to specify the particular instance. If the more than one attribute is required to uniquely identify an instance, an instance number for each attribute is appended to the OID, separated by a dot, in the order specified by the INDEX definition in the `ASN.1` file.

For example, suppose that you want to check for the condition in which the state of a particular server is anything but active. To uniquely specify a server instance, you require both the group number and the server ID. The INDEX entry for `tuxTsrvrTbl` in the `ASN.1` file specifies the following as an INDEX to particular instances.

```
INDEX (tuxTsrvrGrpNo,tuxTsrvrId)
```

The relative OID for `tuxTsrverState` is the following:

```
140.300.20.1.1.5
```

Thus, to specify the particular server instance for group 55 and server ID 3, you use the following OID:

```
140.300.20.1.1.5.55.3
```

Note that the order of the two attribute instances added to the `tuxTsrvrState` OID is indicated by the INDEX definition above: `tuxTsrvrGrpNo` followed by `tuxTsrvrId`.

You can thus define the condition that you want to check as follows:

```
VAL(140.300.20.1.1.5.55.3) != 1
```

This condition evaluates to true whenever this particular server instance is not active.

You can use a specific number to specify a particular instance or the asterisk wildcard to specify all instances. Use zero as the instance index in the case of *scalar* objects (objects that can have only one instance). Use the asterisk wildcard only to represent all instances of a columnar object. For example:

```
.1.3.6.1.4.1.140.1.1.0
```

specifies the single instance of a scalar object while:

```
.1.3.6.1.4.1.140.2.22.1.2.*
```

specifies all of the instances of a columnar object. When a wildcard is used to define a condition, the condition is satisfied if *any* instance satisfies the condition.

Notes:

- When you specify multiple OIDs in a complex rule, the OIDs should either all use wildcards or none should. That is, you should not combine an OID that specifies a particular instance with an OID that uses a wildcard in the same rule.
- When you use multiple OIDs with wildcards in a single rule, all the OIDs should specify objects only within the same table.
- In complex conditions, when you use asterisk (*) as the index for the OIDs, the condition is checked between columns of the same row, for all rows available in that table.
- When you use a wildcard to define a condition, the condition is satisfied if any instance satisfies the condition. For example, `VAL(.1.3.6.1.2.1.2.1.2.2.1.8.*) == 1` is satisfied if `1` is the value of any instance. Once the condition is satisfied, the rule is in the ERR state. The rule remains in the ERR state as long as any instance satisfies the condition. The rule transitions to the OK state only when no instance satisfies the rule.

States and Transitions

Associated with each active polling rule is a *state*. There are two possible states for an active rule:

- OK—A rule is in the OK state when the specified condition does not hold (threshold is not crossed).
- ERR—A rule is in the ERR state when the specified condition does hold (threshold is crossed).

Transitions determine when an action is to be taken in response to a poll. BEA SNMP Agent Integrator polling rules execute an action (such as generating a trap notification) only when a polling rule undergoes a transition from OK to ERR or from ERR to OK.

When the BEA SNMP Agent Integrator begins executing a polling rule, the rule is initially in the OK state. As long as the threshold is not crossed, the rule remains in the OK state. If the threshold is crossed, the rule undergoes a transition from the OK state to the ERR state. As long as the condition continues to evaluate as true, the rule remains in the ERR state. If the condition subsequently evaluates to false, the rule then transitions back to the OK state. Thus, there are two types of transition:

- A transition from the OK state to the ERR state
- A transition from the ERR state to the OK state

Note: When conditions are defined for all instances of a columnar object using a wildcard (“*”), the rule transitions from the OK state to the ERR state if the column in any row of the table evaluates to true for the defined condition. The rule transitions to OK if the condition evaluates to false for the column in all rows of the table.

Actions

An BEA SNMP Agent Integrator rule can specify two types of action to be taken if the polling rule undergoes a transition:

- An enterprise-specific trap with a user-specified specific trap number.
- A specified program or script (or batch file).

Both types of action can be specified in the same rule.

Note: The BEA SNMP Agent Integrator carries out an action *only* when a transition occurs. Continued polling does not result in duplicate actions as long as the rule remains in the same state. This restriction prevents duplicate traps from being generated in response to detection of a single event.

Four keywords are used to define actions:

`TRAPID_ERR = specific-trap-number`
: Indicates that a trap should be sent if the state of the rule transitions from OK to ERR.

`TRAPID_OK = specific-trap-number`
: Indicates that a trap should be sent if the state of the rule transitions from ERR to OK.

`COMMAND_ERR = “command”`
: The program specified by *`command`* is executed if the state of the rule transitions from OK to ERR.

`COMMAND_OK = “command”`
: The program specified by *`command`* is executed if the state of the rule transitions from ERR to OK.

Note: The string specifying the command to be executed must be in quotes. For example: `COMMAND_ERR = “usr/mybin/test.ksh”`.

If you do not specify the absolute path to the executable or script, the path should be specified in the BEA SNMP Agent Integrator’s environment settings.

The statements specifying actions must be placed within curly braces. When multiple commands are specified in a rule, the commands must be separated by spaces and `command` must be enclosed in quotes.

A string containing the name of the rule and the direction of the state transition (OK to ERR or ERR to OK) is passed as an argument to the script or program called by the COMMAND_ERR or COMMAND_OK actions.

Trap Information

The following information is passed in the enterprise-specific traps generated by BEA SNMP Agent Integrator polling rules:

- User-defined specific trap number
- Rule name and state change

  A TRAPID_ERR action passes a string in the variable bindings of the trap that takes the following form:

  ```
  Rule id rule-name has triggered from OK to ERR state
  ```

  A TRAPID_OK action passes a string in the variable bindings of the trap that takes the following form:

  ```
  Rule id rule-name has triggered from ERR to OK state
  ```

- Enterprise ID

  When a threshold is crossed, the BEA SNMP Agent Integrator generates an SNMP trap packet (PDU) that has the following enterprise OID in its enterprise field:

  ```
  .1.3.6.1.4.1.140.1.1
  ```

  Note: BEA SNMP Agent Integrator polling alarms have a different enterprise identifier in the enterprise field of the trap than the BEA SNMP Agent traps that forward Tuxedo system events. BEA SNMP Agent Integrator polling alarms use bea as an enterprise identifier, whereas the BEA Tuxedo SNMP Agent system traps use tuxedo as the enterprise identifier.

Examples

In the following example, the BEA SNMP Agent Integrator polls every ten minutes (600) to determine if disk capacity in use is greater than 90 percent. If any file system has more than 90 percent capacity in use, an enterprise-specific trap with number 102 is generated. If subsequently all the file systems have less than or equal to 90 percent of capacity in use, an enterprise-specific trap with trap number 202 is generated.

```
RULE_ACTION diskchk 600
   if (VAL(140.2.22.1.5.*) > 90) {TRAPID_ERR = 102 TRAPID_OK = 202}
```

In the next example, a Tuxedo application is checked to determine if the transaction `triptime` exceeds 36 msec. If the threshold is crossed, an enterprise-specific trap is generated and a user script, `logtime`, is invoked to log the time of the event. If the `triptime` is subsequently less than 36 msec after having crossed that threshold on the previous poll, an enterprise-specific trap with a number of 302 is generated.

```
RULE_ACTION triptime 20
   if (VAL(140.150.1.3.*) > 36)
      {TRAPID_ERR = 301 TRAPID_OK = 302
         COMMAND_ERR = “/usr/sbin/logtime”}
```

Note: The object identifier in this example is not defined in the BEA MIB, which is an example of an object that might be defined in a user-supplied custom MIB.

In the next example, the BEA SNMP Agent Integrator polls every five seconds to check whether the number of requests completed by the Tuxedo server `Server1` is greater than six. If it is, an enterprise-specific trap is generated with a specific trap number of 210 and the command `C:/etc/srv_reqs.cmd` is executed.

```
RULE_ACTION Server1 5
   if ((VAL(140.300.20.2.1.12.*) > 6))
      { TRAPID_ERR=210  COMMAND_ERR="C:/etc/srv_reqs.cmd" }
```

In the next example, the BEA SNMP Agent Integrator is checking a particular server instance in any state other than active. The server that is being checked is uniquely identified by its group number and server ID: group number 55 and server ID 3.

```
RULE_ACTION srvrUp 60 if (VAL(140.300.20.1.1.5.55.3) != 1
   {TRAPID_ERR = 306 TRAPID_OK = 307}
```

Whenever the server satisfies the condition, the rule transitions to the ERR state and generates an enterprise-specific trap with the specific trap number of 306. Whenever the server becomes active again, it transitions back to the OK state and issues a trap with the specific trap number of 307.

Starting BEA SNMP Agent Integrator Polling Activity

Polling rules are defined as `RULE_ACTION` entries in the BEA SNMP Agent `beamgr.conf` configuration file. The default location of this file is `C:\etc` on Windows systems and `/etc` on UNIX systems. Individual rules are MIB objects, stored as an entry (row) in the `beaIntAgtTable`.

The status of each rule entry determines whether the BEA SNMP Agent Integrator executes that rule (that is, whether the BEA SNMP Agent Integrator actively checks the condition specified in

the rule). The status of each rule entry is stored in the `beaIntAgtStatus` object. Polling is active for a rule if the status of that rule is valid (integer value of 1). Polling is inactive for a rule if its status has been set to inactive (integer value of 3). The specific rule can be Set from a management station (such as OpenView or SunNet Manager) by using the unique name of the rule as the key field used to specify the entry instance (row).

Note: The BEA SNMP Agent Integrator must be running in order to successfully Set objects in the `beaIntAgtTable`.

The BEA SNMP Agent Integrator begins to execute all polling rules defined in `RULE_ACTION` entries in the BEA SNMP Agent `beamgr.conf` configuration file when it first starts up. The status of each rule object in the `beaIntAgtTable` is valid at startup.

Creating New Polling Rules

You can add rules to the configuration file in two ways:

- Use a text editor, such as `vi`, to add a `RULE_ACTION` entry to file, making sure that you conform to the syntax of the rule, as described in "Configuration Files" on page 9-1. However, if the BEA SNMP Agent Integrator is already running, the new rule does not take effect until you execute the following command:

  ```
  reinit_agent snmp_integrator
  ```

  Executing this command causes the BEA SNMP Agent Integrator to re-read its configuration file.

- Since individual rules are MIB objects, stored as an entry (row) in the `beaIntAgtTable`, you can use an SNMP manager (or the `snmptest` utility) to create a new entry (row) in the `beaIntMgtTable`. (The SNMP manager must have the ability to issue SNMP Set requests that contain multiple objects in a single Set request.) To create the new row, issue a Set request after you specify an index value that does not already exist in the table. This Set request causes a new `RULE_ACTION` entry to be created in the configuration file.

Deleting or Modifying Polling Rules

BEA SNMP Agent Integrator polling rules can be modified in the same two ways they can be created:

- Use a text editor to delete (or comment out) or modify a `RULE_ACTION` entry in the `beamgr.conf` file. This change does not take effect unless you issue the following command to force the BEA SNMP Agent Integrator to re-read its configuration file:

  ```
  reinit_agent snmp_integrator
  ```

- Use SNMP Set commands to delete or modify rules.

Stopping BEA SNMP Agent Integrator Polling Activity

Polling can be de-activated in one of two ways:

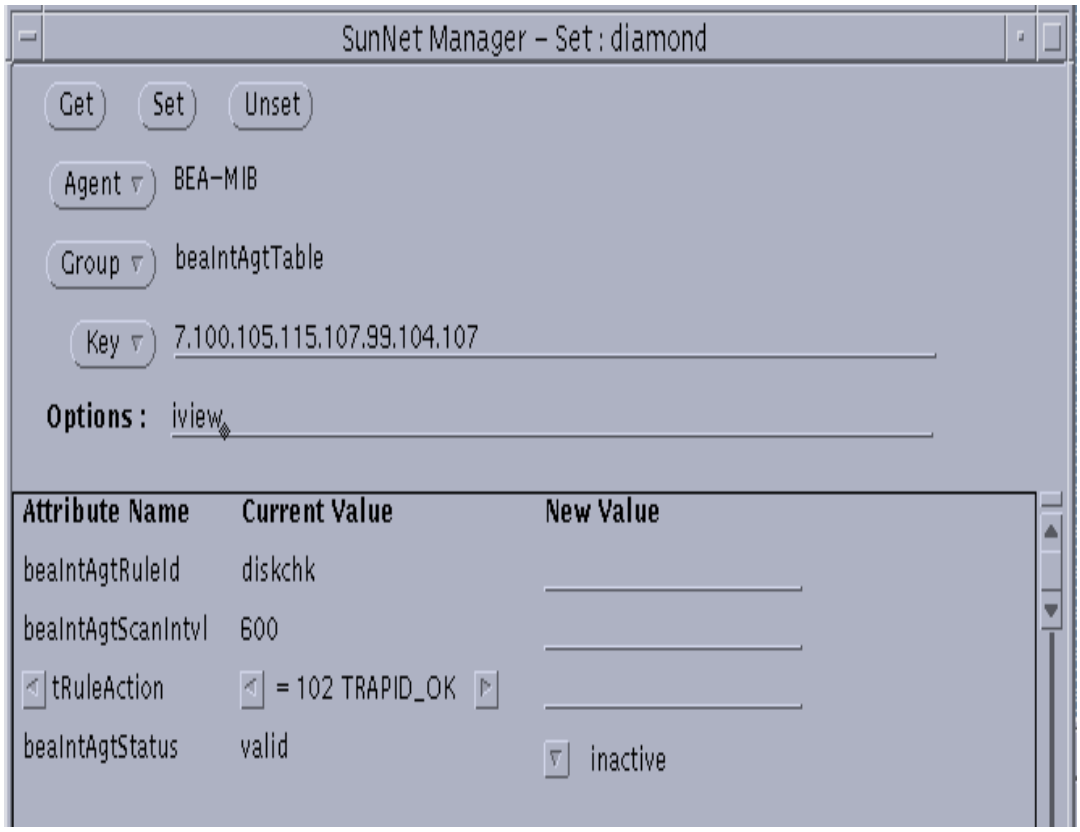
- Remove the `RULE_ACTION` entry from the configuration file.

  You can turn off a polling rule by commenting out or deleting that `RULE_ACTION` entry in the BEA SNMP Agent `beamgr.conf` configuration file. However, for this to take effect, you need to execute `reinit_agent snmp_integrator`. This execution, in turn, causes the BEA SNMP Agent Integrator to re-read its configuration file.

- Use `snmptest` or an SNMP-compliant manager to Set the value of the rule status to inactive.

  Polling for that rule can be de-activated from the management station (or by using the `snmptest` utility packaged with the BEA SNMP Agent Integrator) by setting the value of that object to inactive (an integer value of 3). Setting the value to 2 (invalid) causes the `RULE_ACTION` entry to be deleted from the configuration file. The following figure shows the rule `diskchk`, discussed earlier, being set to `inactive`. Note that the read/write community string (in this example, `iview`) is required for set permission.

Figure 7-1 Setting a Polling Rule to Inactive



Restarting BEA SNMP Agent Integrator Polling Activity

When a polling rule has been de-activated using a Set request from a management station, the rule can be re-activated using a Set request to set the value of the corresponding `beaIntAgtStatus` object to valid (integer value of 1).

CHAPTER 8

BEA SNMP Agent Integrator Commands

The following sections explain the commands and utilities used for the BEA SNMP Agent Integrator:

Commands

reinit_agent

Syntax

```
reinit_agent all | logical_agent_name [logical_agent_name]
```

Description

Causes the specified agents to re-read their configuration file. This utility must be run with root permissions. Using the `all` argument causes all SNMP agents to re-initialize. For all SNMP agents other than `tux_snmpd`, *`logical_agent_name`* is the name of the executable.

For example, the command:

```
reinit_agent snmp_integrator
```

causes the BEA SNMP Agent Integrator to re-read its configuration file.

snmp_integrator

Syntax

```
snmp_integrator [-d] [-n] [-p port | -r smux_port]
   [-b ipaddr_list | hostname_list ]
```

Arguments

`-d`
Causes the program to display a message for each SNMP/SMUX packet sent or received.

`-n`
The program is not run as a daemon (UNIX only).

`-p port`
Specifies the UDP port on which the BEA SNMP Agent Integrator listens for SNMP requests (default: 161/udp).

`-r smux_port`
Specifies the TCP port used to communicate with SMUX subagents (default: 199/tcp).

`-b ipaddr_list | hostname_list`
If the machine where the BEA SNMP Agent Integrator is running has multiple IP addresses, by default the BEA SNMP Agent Integrator listens on all IP addresses. The `-b` option can be used to specify a subset of IP addresses to monitor for incoming SNMP requests.

`ipaddr_list`

Can consist of a single IP address or a blank-separated list of IP addresses.

`hostname_list`

Can consist of one host name or a blank-separated list of host names.

For example, if the machine on which the BEA SNMP Agent Integrator is running has the following IP addresses:

```
130.86.34.3
130.86.33.13
130.86.23.1
```

you can configure the BEA SNMP Agent Integrator to only service requests addressed to `130.86.23.1` by starting it with the following command:

```
snmp_integrator -b 130.86.23.1
```

Description

The `snmp_integrator` file is the SNMP BEA SNMP Agent Integrator executable. It allows multiple SNMP agents and SMUX subagents from any vendor to coexist on the same node and to appear as a single SNMP agent to any SNMP manager.

The BEA SNMP Agent Integrator can simultaneously support any number of:

- SNMP agents
- SMUX subagents
- Master agents such as other SMUX or DPI master agents or any other proprietary master agents. (The master agent must respond to requests from a manager using SNMP.)

Also, the BEA SNMP Agent Integrator can coexist on the standard SNMP port (161/udp) with any other SNMP agent. It directly supports the SMUX MIB (RFC 1227) in addition to the MIB-II `system(1)` and `snmp(3)` groups.

When the program is running as an SNMP agent, it generates a coldStart trap to the host specified by the `TRAP_HOST` entry in the `beamgr.conf` file at startup. If there is no `TRAP_HOST` entry, the trap is sent to UDP port 162 on the host where the utility is running, with a community defined as `public`.

Read-write and read-only communities supported by the BEA SNMP Agent Integrator can be specified in the `beamgr_snmpd.conf` file. By default, read-only community is `public` and read-write community is `iview`.

Using the `beamgr_snmpd.conf` file, you can configure the BEA SNMP Agent Integrator to expect a password from SMUX subagents that register with it.

On Windows Systems

Messages displayed with the `-d` argument are sent to the Event Log.

The `-n` argument has no effect.

On UNIX Systems

The `-d` argument is usually used for debugging purposes when the program is executed on the command line. Messages displayed are sent to the standard output of the program. If the program is started by `init`(1M), the destination of these messages is determined by the UNIX platform and version. These messages are most frequently sent to the console.

The `-n` argument is usually used when the program is started by `init`(1M) with the respawn option.

stop_agent

Syntax

```
stop_agent logical_agent_name | all [logical_agent_name]
```

Arguments

`all`
Stops all SNMP agents.

`logical_agent_name`
For all SNMP agents other than `tux_snmpd`, the logical agent name is always the name of the executable.

show_agent

Syntax

```
show_agent all | logical_agent_name [logical_agent_name]
```

Description

Lists the names and PIDs of all the running agents and requested agents.

BEA SNMP Agent Utilities

The BEA SNMP Agent software provides the following utilities to help you install and test an agent or subagent:

`instsrv`	Installs an agent as a Windows service.
`snmpget`	Reports information about scalar managed objects.
`snmpgetnext`	Returns the next consecutive managed object in a MIB.
`snmptest`	Selectively performs Get, GetNext, and Set operations on any MIB object.
`snmptrap`	Sends an SNMP trap message to a host.
`snmptrapd`	Receives and logs SNMP trap messages sent on a local machine to the snmp-trap port.
`snmpwalk`	Traverses the OID tree using the SNMP GetNext request to query managed objects.

instsrv

Purpose

Used to install any module built as a Windows service under a specified name. For example, use it to reinstall the BEA SNMP Agent Integrator in a multi-version environment after uninstalling one of the versions. Applies to Windows systems only (not UNIX systems).

Synopsis

```
instsrv service_name [executable_file | remove]
```

Enter an *executable_file* to create a service. Enter `remove` to remove a service.

For example:

```
instrsrv snmp_integrator c:\tux81\bin\snmp_integrator.exe
```

Arguments

service_name
The name of the service.

executable_file
The complete path to the executable file.

snmpget

Purpose

Reports information about scalar managed objects.

Synopsis

```
snmpget [-d] [-p port] host community variable_name
   [variable_name ...]
```

Arguments

`-d`
Causes the program to display a message for each packet.

`-p` *port*
Specifies the UDP port used to communicate with the SNMP agent (default: 161).

host
The internet address or host name of the node executing the SNMP agent to be queried.

community
The community name for the transaction.

variable_name

At least one unique object identifier (OID).

Description

The `snmpget` utility uses SNMP Get requests to retrieve information about managed objects. You can enter one or more object identifiers as arguments on the command line. These names can be absolute, starting from the root of the tree, or relative to `.iso.org.dod.internet`.

Environment Variables

`BEA_SM_SNMP_MIBFILE`

Must be set to specify the path to `mib.txt`, which provides an ASCII text description of the contents of your private MIB.

Examples

The following command sends a query to the SNMP agent running on the host named `topaz`, using `public` as the community for authorization. The agent retrieves the value of the managed object `beaSysHasDisk` in the BEA private MIB. Note that in this example, a relative OID (`private.enterprises.bea.beaSystem`) is specified. `.iso.org.dod.internet.` is prepended to generate an absolute path.

```
snmpget topaz public private.enterprises.bea.beaSystem
   .beaSysHasDisk.0
```

This command returns the following information about the object:

```
Name: private.enterprises.bea.beaSystem.beaSysHasDisk.0
INTEGER: yes(2)
```

The following command sends a query to the SNMP agent running on the host named `ruby`, using `public` as the community for authorization. The agent retrieves the value of the managed objects `sysDescr` and `sysUptime` in the MIB.

```
snmpget ruby public mgmt.mib.system.sysDescr.0
   mgmt.mib.system.sysUpTime.0
```

This command returns the following information:

```
Name: mgmt.mib.system.sysDescr.0
OCTET STRING- (ascii): Kinetics FastPath2
```

```
Name: mgmt.mib.system.sysUpTime.0
Timeticks: (2270351) 6:18:23
```

snmpgetnext

Purpose

Returns the next entry in a table or the next consecutive managed object in a MIB.

Synopsis

```
snmpgetnext [-d] [-p port] host community variable_name
   [variable_name ...]
```

Arguments

`-d`

Causes the program to display a message for each packet.

`-p` *`port`*

Specifies the UDP port used to communicate with the SNMP agent (default: 161).

`host`

The internet address or host name of the node executing the SNMP agent to be queried.

`community`

The community name of the transaction.

`variable_name`

At least one unique object identifier (OID).

Description

You can enter one or more object identifiers as arguments on the command line. These names can be absolute, starting from the root of the tree, or relative to `.iso.org.dod.internet.`

Environment Variables

`BEA_SM_SNMP_MIBFILE`

Must be set to specify the path to `mib.txt`, which provides an ASCII text description of your private MIB.

Examples

This example contacts the host named `blueberry` using the community name `public` and retrieves the value of the instance immediately following `mgmt.mib.interfaces.ifTable.ifEntry.ifOutOctets.0` from the MIB:

```
snmpgetnext blueberry public mgmt.mib.interfaces.ifTable.ifEntry
   .ifOutOctets.0
```

Note that the instance index `.0` must be appended to the end of the OID to refer to the value of the object.

The output of the previous command might look like this:

```
Name: mgmt.mib.interfaces.ifTable.ifEntry.ifOutOctets.1
COUNTER: 85655250
```

You could then enter a command that retrieves information about the next variable:

```
snmpgetnext blueberry public mgmt.mib.interfaces.ifTable.ifEntry
   .ifOutOctets.1
```

snmptest

Purpose

Selectively performs Get, GetNext, and Set operations on any MIB object.

Synopsis

`snmptest [-d] [-p` *`port`*`]` *`host community`*

Arguments:

`-d`

Causes the program to display a message for each packet.

`-p` *`port`*

Specifies the UDP port used to communicate with the SNMP agent (default: 161).

`host`

The internet address or host name of the node executing the SNMP agent to be queried.

`community`

The community name of the transaction.

Description

When this program is executed, it prompts you to enter an OID. The `snmptest` utility returns information about request and reply packets as well as the name and type of the object.

By default, the program sends a Get request packet. This can be changed by entering a value from the following table at the prompt.

Command	Request Type
`$G`	Get
`$N`	GetNext
`$S`	Set

If you choose the Set request mode, the program prompts you for a variable type from the following list.

Variable Type	Description
a	IP address.
d	Octet string as decimal bytes separated by white space (that is, 105 118 105 101 119)
i	Integer
n	Null value
o	Object identifier
s	Octet string in ASCII (that is, bea)
t	Time ticks
x	Octet string as hexadecimal bytes separated by white space (for example, 69 76 69 65 77)

After you specify the request type, the program prompts you to enter a value of the type you just specified. At this prompt, enter the integer (in decimal) or enter a string and press Enter. To send the request packet, press Enter again at the next prompt.

To quit the program, enter:

```
$Q
```

Environment Variables

`BEA_SM_SNMP_MIBFILE`

Must be set to specify the path to `mib.txt`, which provides an ASCII text description of your private MIB.

Examples

Start the program by entering the command:

```
snmptest topaz public
```

The program responds with:

```
Please enter the variable name:
```

Enter a variable name and press Enter:

```
private.enterprises.bea.beaEm.beaEmMonitorTimer.0
```

The program requests another variable name:

```
Please enter the variable name:
```

You can either enter another variable name, or press Enter to see the result. When you press Enter, the program displays the result of the test:

```
Received GET RESPONSE from 192.84.232.47
requestid 0x775efba0 errstat 0x0 errindex 0x0
Name: private.enterprises.bea.beaEm.beaEmMonitorTimer.0
INTEGER: 5000
```

After displaying the result, you can enter another variable name, or `$Q` to quit the program.

```
Please enter the variable name: $Q
```

If you enter `$Q`, a quit message appears:

```
Quitting, Good-bye
```

snmptrap

Purpose

Sends an SNMP trap message to a host.

Synopsis

```
snmptrap [-a agent_addr] [-d] [-p port] host community
   trap_type specific_trap variable_binding_value
```

Arguments

`-a` *`agent_addr`*

Specifies an originating address, if it is different from that of the host, where `snmptrap` is executed. This argument enables you to send a trap on behalf of another host.

`-d`

Causes the program to display a message for each packet.

`-p` *`port`*

Specifies the UDP port to which the SNMP trap should be sent on the target host (default: 162).

`host`

The Internet address or name of the host to which the SNMP trap is to be sent.

`community`

The community name of the transaction.

trap_type
An integer that specifies the generic type (in the range 0 to 6) of the trap to be sent.

specific_trap
An integer that identifies the enterprise-specific trap that occurs when *trap_type* is set to generic trap type 6.

variable_binding_value
Information to be transported within the trap packet. The program uses this as the value in the variable binding list when it sends the trap.

Description
The following table defines the valid (generic) trap types.

Name of Trap Type	Generic Trap Number	Description
coldStart	0	The sending agent is re-initializing itself, typically due to a reboot.
warmStart	1	The sending agent is re-initializing itself, typically due to a normal restart.
linkDown	2	One of the communication links on the agent node has failed. The first element in the variable bindings contains the name and value of the ifIndex instance for the downed interface.
linkUp	3	One of the communication links on the agent node has come up. The first element in the variable bindings contains the name and value of the ifIndex instance for the affected interface.
authenticationFailure	4	The agent is reporting it has received a request with an invalid community specification or a community with insufficient permissions to complete the request.
egpNeighborLoss	5	The agent is reporting that the peer relationship between an External Gateway Protocol (EGP) neighbor and an EGP peer no longer exists.
enterpriseSpecific	6	The sending agent is reporting that an enterprise-specific event has occurred. The value of the specific-trap field indicates the nature of the event.

The trap generated by this tool has a fixed variable-binding list that contains only one object-value pair. The object is:

```
.iso.org.dod.internet.private.enterprises.bea.beaSystem.
   beaTrapDescr.0
```

The value of this object can be specified in the `variable-binding-value` argument.

The enterprise field, which is part of the SNMP trap PDU header, is always:

```
.1.3.6.1.4.1.140.1.1
```

which is equivalent to:

```
.iso.org.dod.internet.private.enterprises.bea.beaSystem.
   sysDescr
```

Examples

The following command sends a `coldStart` trap to the host named `topaz`, using `public` as the community for authorization. Note that a value for the `specific-trap` argument must be present, even though it is ignored when the value of the `trap-type` argument is not 6 (`enterpriseSpecific`).

```
snmptrap topaz public 0 1 “host xyz is booting”
```

snmptrapd

Purpose

Receives and logs SNMP trap messages sent to the snmp-trap port.

Synopsis

`snmptrapd [-d] [ -l` *`port`* `] [-p]`

Arguments

`-d`

Causes the program to display a debug message for each packet.

`-l` *`port`*

Specifies the UDP port to use when listening for incoming trap packets (default: 162).

`-p`

Causes the program to print trap information output to the standard output.

Environment Variables

`BEA_SM_SNMP_MIBFILE`

Must be used to specify the path to `mib.txt`, which provides an ASCII text description of your private MIB.

Description

This utility receives SNMP traps sent on the UDP port specified by the `-l` argument. If no port is specified, it uses port number 162. This utility must be able to open the snmp-trap port, which usually requires `root` permissions.

On UNIX platforms, if the `-p` argument is not specified, `snmptrapd` uses the UNIX `syslog` utility to log messages with a status of WARNING. If the LOG_LOCAL0 facility is available, it is used instead of `syslog` or `snmptrapd`.

On Windows systems, if the `-p` argument is not specified, the Event Log is used to log WARNING messages.

Examples

This command collects the incoming SNMP trap sent by another host, and displays it to standard output:

```
snmptrapd -p
```

When the host receives the trap, it displays the following information:

```
192.84.232.47: Cold Start Trap (0) Uptime: 0:00:00
Name: private.enterprises.bea. beaSystem.beaTrapDescr.0
OCTET STRING- (ascii): host xyz is booting
```

snmpwalk

Purpose

Traverses the OID tree using the SNMP GetNext request to query managed objects.

Synopsis

`snmpwalk [-d] [-p` *`port`*`] host community [`*`variable_name`* `...]`

Arguments

`-d`

Causes the program to display a message for each packet.

`-p` *`port`*

Specifies the UDP port used to communicate with the SNMP agent (default: 161).

`host`

The host name or an Internet address, in “dot-dot” notation (that is, separated with periods), where the SNMP request is to be sent.

`community`

The community name to use in the SNMP request.

variable_name

The unique object identifier, expressed symbolically, decimally, or as a combination of both. If you do not specify a variable name, `snmpwalk` searches the entire MIB.

Description

This utility traverses the OID tree from the object specified on the command line. You can enter one or more object identifiers as arguments on the command line. These names can be absolute, starting from the root of the tree, or relative to `.iso.org.dod.internet`. If no objects are specified, `snmpwalk` searches the entire MIB tree supported by the SNMP agent.

Environment Variables

`BEA_SM_SNMP_MIBFILE`

Must be used to specify the path to `mib.txt`, which provides an ASCII text description of your private MIB objects.

Diagnostics

If the tree search causes the program to search beyond the end of the MIB, this message appears:

```
End of MIB
```

Examples

This is an example of an `snmpwalk` command:

```
snmpwalk blueberry public private.enterprises.bea.beaSystem
```

This is some of the output generated from the command:

```
Name: private.enterprises.bea.beaSystem.beaSysSysname.0
   OCTET STRING- (ascii): SunOS

Name: private.enterprises.bea.beaSystem.beaSysNodename.0
   OCTET STRING- (ascii): blueberry
```

SNMP Request Format

BEA SNMP Agent utilities use SNMP requests to query SNMP agents for information about managed objects. Refer to RFC 1157 (SNMP) for more information about the format of SNMP requests. For information about locating RFCs on the Internet, see "SNMP Information" on page A-1.

MIB Variable Definition Files

When a MIB variable is used with a BEA SNMP Agent utility, the utility attempts to convert the variable to a numeric OID by searching first in a file named `mib.txt` in the current directory, then in a file specified in the environment variable `BEA_SM_SNMP_MIBFILE`, and finally in the *`tux_prod_dir`*`\udataobj\snmp\etc\mib.txt` file on a Windows system, or *`tux_prod_dir`*`/udataobj/snmp/etc/mib.txt` file on a UNIX system. These files should use ASN.1 notation and use the OBJECT TYPE macro defined in RFC 1155 (Structure of Management Information).

The `mib.txt` file describes the RFC 1213 (MIB-II) and the BEA private MIB objects.

CHAPTER 9

Configuration Files

The following sections describe the configuration files accessed by the BEA SNMP Agent Integrator and the BEA SNMP agents:

- BEA SNMP Agent Configuration File: beamgr.conf
- BEA SNMP Agent Passwords File: beamgr_snmpd.conf

BEA SNMP Agent Configuration File: beamgr.conf

At startup, the BEA SNMP Agent Integrator (`snmp_integrator`) and the BEA SNMP agent (`tux_snmpd`) read the absolute pathname specified by the `BEA_SM_BEAMGR_CONF` environment variable to locate the `beamgr.conf` configuration file on the host system. The BEA SNMP Agent Integrator and agents read certain configurations in the `beamgr.conf` file.

Note: The `BEA_SM_BEAMGR_CONF` setting ends with filename `beamgr.conf`.

Default Location

If the `BEA_SM_BEAMGR_CONF` environment variable is not set, the BEA SNMP Agent Integrator and agents look for `beamgr.conf` at the following default location:

- For Windows systems: `C:\etc\beamgr.conf`
- For UNIX systems: `/etc/beamgr.conf`

If the BEA SNMP Agent Integrator and agents cannot find the `beamgr.conf` file on the host system, they will print an error message and *not* start. The `beamgr.conf` file is a *mandatory* configuration file.

Description

The `beamgr.conf` configuration file contains information used by the BEA SNMP Agent Integrator and the BEA SNMP agents. A configuration entry in the file consist of two or more blank or tab-separated fields:

`KEYWORD parameters`

If an entry is too long, you can use the backslash (\) character as a continuation character at the end of the line. There should be a newline character immediately following the \ character.

The following keywords have corresponding MIB objects:

- `TMEVENT_FILTER`
- `SYS_INSTALL`
- `SYS_CONTACT`
- `SYS_NAME`
- `SYS_LOCATION`
- `SYS_SERVICES`
- `RULE_ACTION`

Keywords Used by All BEA SNMP Agent Components

`TRAP_HOST`

Host name, port number, and community name necessary to send SNMP traps.

`host_name trap_port trap_community`

`host_hame`

The name of the target destination machine for the trap notification. The default is the local host.

You can have multiple `TRAP_HOST` entries in the configuration file if you need to send traps to multiple destinations.

The `TRAP_HOST` entry is used by the BEA SNMP Agent Integrator and the SNMP agents to determine trap destinations when they generate SNMP trap notifications.

`SNMP_ENABLE_AUTH_TRAP`

Contains a value that indicates whether the SNMP agent is permitted to generate authentication-failure traps. If the value is 1, the SNMP agent generates authentication-failure traps when an invalid request (according to the community profile) is received. If the value is not 1, no authentication-failure trap is generated.

`OID_CLASS`

The OID is a unique number assigned to each object in the MIB as an object identifier. OIDs fall into specific categories or classes. When the SNMP agent accesses a specific object, it traverses the OID tree in the MIB file to find the object. An OID identifies an object by specifying a unique path to the object from the root of the OID tree.

Keywords Used by the BEA SNMP Agent Integrator

`INTEGRATOR_TIMEOUT`

Sets the BEA SNMP Agent Integrator timeout in waiting for responses to requests. The default timeout is 30 seconds.You can set the BEA SNMP Agent Integrator timeout by adding an `INTEGRATOR_TIMEOUT` entry as follows:

```
BEA_PEER MAX_WAIT 30
```

`INTEGRATOR_MAX_TIMEOUTS`

Sets the maximum number of times the BEA SNMP Agent Integrator permits requests to an SNMP peer or SMUX subagent to time out before disregarding any further requests for that agent or subagent. The default is 3.

`NON_SMUX_PEER`

Specifies the OIDs supported by an SNMP agent when it is running as a peer of the BEA SNMP Agent Integrator. For more information, see “NON_SMUX_PEER Entry” on page 9-6.

`RULE_ACTION`

Specifies an BEA SNMP Agent Integrator polling rule. This allows threshold-checking to be offloaded from the network management system to the distributed integrator agents on managed nodes. For more information, see “RULE_ACTION Entry” on page 9-11.

Keywords Used by the BEA SNMP Agent

`TMAGENT`

Defines the BEA Tuxedo domain that an agent is to monitor. There must be one `TMAGENT` entry for each Tuxedo SNMP agent on a managed node. The format of the entry is as follows:

`TMAGENT` *`logical_agent_name tuxdir tuxconfig_path`*

Monitoring of multiple domains is performed by running a separate Tuxedo agent for each domain being monitored. These agents must be run as SMUX subagents under the BEA SNMP Agent Integrator.

When there are multiple Tuxedo SNMP agents running as SMUX subagents on a managed node, the logical agent name must be used to modify the community in Set or Get requests. The community must be of the following form:

`community@logical_agent_name`

For example:

`public@payrollagent`

The community does not need to be qualified with the logical agent name when there is only one Tuxedo SNMP agent running on the managed node.

`TMEVENT_FILTER`

Defines a subset of BEA Tuxedo event notifications that are to be forwarded as SNMP trap notifications. By default, if no filter is provided, BEA SNMP Agent forwards all Tuxedo events as SNMP traps. The format of the entry is:

`TMEVENT_FILTER` *`filter_id logical_agent_name tux_evt_expr tux_evt_filter status`*

Note that the strings used for each of the parameters must not have blanks or white space.

`filter_id`

A string that must be unique among all `TMEVENT_FILTER` entries in the BEA SNMP Agent configuration file. The maximum length of the string is 16 characters.

`logical_agent_name`

Maps the event filter to a particular agent on that node. A logical agent name is a string up to 32 characters long. The logical agent name is the name given to the Windows service on a Windows system that starts the agent, or the name specified in the `-l` option when starting the agent from the command line on a UNIX system.

tux_evt_expr

A Tuxedo event name expression. This string can be a maximum of 255 characters long. For information about event name expressions, which are regular expressions, see reference page tpsubscribe(3c) in *BEA Tuxedo ATMI C Function Reference*.

This name must match a Tuxedo event name for that event to be forwarded as an SNMP trap. For a list of event names, see reference page EVENTS(5) in *BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes Reference*.

The default is all events. If NONE is used, no events are forwarded and the other parameters in the TMEVENT_FILTER entry can be omitted. An entry of NONE overrides all other event filter entries for the same logical agent name. For example:

. An entry of

```
\.Sys.*
```

matches all events.

An entry of

```
\.SysServer.*
```

matches all system events related to servers.

tux_evt_filter

An event filter expression. This string can be a maximum of 255 characters long. Each Tuxedo event is accompanied by an FML buffer that contains pertinent information about the event. The buffer is evaluated with respect to this filter if the filter is present. If the buffer content is evaluated to TRUE, the event is forwarded; otherwise the event is not forwarded. BEA SNMP Agent uses this filter as an argument for a call to tpsubscribe(). For more information, see reference page tpsubscribe(3c) in *BEA Tuxedo ATMI C Function Reference*.

status

Can be either active or inactive. If the status is active, the filter is being used; otherwise the filter is not being used.

There is a MIB table that corresponds to the TMEVENT_FILTER entries in the BEA SNMP Agent configuration file. These entries can be updated dynamically using an SNMP Set request. For more information, see the beaEvtFilterTable object in "Core MIB" in *BEA Tuxedo SNMP Agent MIB Reference*.

NON_SMUX_PEER Entry

To configure the BEA SNMP Agent Integrator to access MIB objects through a peer SNMP agent (for example, a non-SMUX master agent or a non-SMUX peer agent), add the `NON_SMUX_PEER` entry to the `beamgr.conf` file in the following format:

```
NON_SMUX_PEER port community|*
   OID_Node[,ro|rw][,priority][,timeout] ...
```

`NON_SMUX_PEER`

The keyword for the entry.

`port`

Specifies the UDP port number on which the SNMP agent is listening. This value may be specified in either of the following forms:

`ip_address.port`
`hostname.port`

when the SNMP agent is remote from the BEA SNMP Agent Integrator. If *`ip_address`* or *`hostname`* is not specified, the BEA SNMP Agent Integrator assumes that the peer SNMP agent is on the same managed node as the BEA SNMP Agent Integrator.

`community`

Specifies the community to be used by the BEA SNMP Agent Integrator when the SNMP agent is polled. The special value * is used to specify that the BEA SNMP Agent Integrator should use the community supplied by the SNMP manager.

`OID_Node`

Specifies the OID of the root of the MIB tree that is supported by this SNMP agent.

`ro|rw`

Specifies whether this OID tree is being exported as read-only or as read-write. The default is `rw`.

`priority`

A positive number that specifies the priority at which the OID tree is being exported. The lower the number, the higher the priority. If there are multiple agents/subagents supporting the same MIB tree, the subagent with the highest priority is consulted. Multiple SNMP agents and SMUX subagents can register the same subtree; however, they must do so at different priorities.

If an SNMP agent tries to register a subtree at a priority that is already taken, the BEA SNMP Agent Integrator repeatedly increments the integer value (lowering the priority) until an unused priority is found. A special priority `-1`, causes the selection of the highest available priority. When a request is made to register with priority `-1`, registration is made at the highest available number below 20. If the priority field is missing, the MIB tree is exported at a `-1` priority.

`timeout`

Specifies the time interval in seconds for which the BEA SNMP Agent Integrator waits for the replies from this SNMP agent for the particular MIB group. The default value is three seconds. This default can be changed by setting the `BEA_PEER_MAX_WAIT` environment variable to a different value.

The access (`ro` or `rw`), *`priority`*, and *`timeout`* fields are optional. However, you must specify access and *`priority`* if you need to specify `timeout`, and you must specify access if you need to specify *`priority`* for a MIB tree.

You can list multiple OID nodes.

Note that a subtree registration hides the registrations by other SNMP agents/subagents of objects within the subtree. So, if an agent A registers subtree `.1.3.6.1.4.1.140` and another agent/subagent, B, registers subtree `.1.3.6.1.4.1.140.1`, agent/subagent A is consulted for all the objects under the `.1.3.6.1.4.1.140.1` subtree.

Also, when the BEA SNMP Agent Integrator reads this entry, an SNMP agent should be running on the specified port. Otherwise, the BEA SNMP Agent Integrator disregards this entry. Also, if three consecutive requests to this SNMP agent time out, it is assumed that the SNMP agent specified by this entry is no longer alive and this entry is disregarded.

At any point, the command `reinit_agent snmp_integrator` can be invoked to force the BEA SNMP Agent Integrator to re-read its configuration file.

The BEA SNMP Agent Integrator disallows/disregards any attempt to register above, at, or below the SNMP (`mib2.snmp`) and SMUX subtrees of the MIB.

NON_SMUX_PEER Examples

The following examples are provided to illustrate the use of the `NON_SMUX_PEER` entry:

- Consider the following sample entries in the `beamgr.conf` file:

```
NON_SMUX_PEER 2001 snmp .1.3.6.1.2.1.1,ro
NON_SMUX_PEER 2002 squid .1.3.6.1.4.1.141 .1.3.6.1.4.1.145
NON_SMUX_PEER 2008 * .1.3.6.1.4.1,ro
NON_SMUX_PEER 2005 * .1.3.6.1.4.1.140 .1.3.6.1.4.1.145,rw
```

The first entry tells the BEA SNMP Agent Integrator to look for an SNMP agent at port 2001. All the requests from BEA SNMP Agent Integrator to this SNMP agent use `snmp` as the community. The agent supports the subtree `.1.3.6.1.2.1.1`, and is available for read-only commands.

The second entry tells the BEA SNMP Agent Integrator to look for an SNMP agent at port 2002. All the requests from BEA SNMP Agent Integrator to this SNMP agent use `squid` as the community. The agent supports the subtrees `.1.3.6.1.4.1.141` and `.1.3.6.1.4.1.145`. Since no access option is specified, both subtrees default to `rw`.

The third entry specifies a non-SMUX agent at port 2008 with a community of *. The * tells the BEA SNMP Agent Integrator to pass along the same community information it receives from the SNMP manager. For example, if the SNMP manager sends the community `nevus`, the BEA SNMP Agent Integrator sends `nevus` along to the subagent. (Of course, `nevus` must be a valid community for the BEA SNMP Agent Integrator in the first place.)

The fourth entry lists an agent at port 2005 with a community of *. The agent supports two subtrees: `.1.3.6.1.4.1.140` and `.1.3.6.1.4.1.145.` Since the first subtree lists no access information, access defaults to `rw`. The second subtree specifically lists `rw`. This means exactly the same thing; the `rw` could have been left off with no effect.

If the `rw` is redundant, then why have it at all? Because each OID node can have three arguments: access (`ro` or `rw`), `priority`, and `timeout`. If you specify any of these arguments, you must also specify all the arguments that come before it. For example, if you specify priority, you must also specify access. If you specify timeout, you must specify both access and priority.

- The following entry tells the BEA SNMP Agent Integrator to look for an agent at port 2008 with a community of *. The agent supports two subtrees. The first has an access of `rw`, a priority of `-1`, and a timeout of two seconds; the second has an access of `rw,` a priority of `-1`, and a timeout of ten seconds. Although `rw` and `-1` are the defaults for access and priority, these values must be stated explicitly in order to include a timeout value.

```
NON_SMUX_PEER 2008 * .1.3.6.1.2.1.1,rw,-1,2
   .1.3.6.1.2.1.2,rw,-1,10
```

  The timeout values are the maximum amount of time the BEA SNMP Agent Integrator waits for a response from the SNMP agent for a given object. In this case, two seconds for a response if the object falls under MIB tree `.1.3.6.1.2.1.1,` and ten seconds for a response if the object falls under the `.1.3.6.1.2.1.2` MIB group. The default value is three seconds. This default value can be changed by setting the environment variable `BEA_PEER_MAX_WAIT`.

- The priority value decides which agent is consulted by the BEA SNMP Agent Integrator in the event of a conflict. If more than one agent handles the same object, the one with the lowest number as a priority value is called. In the following entries, the agents on ports 2008 and 2009 both support object `.1.3.6.1.2.1.1.` The agent at port 2009 has a higher priority (5 is a higher priority than 8), so that is the one the BEA SNMP Agent Integrator calls. Notice that this entry specifies `rw` access. The other entry specifies `ro` access, but since it has a lower priority, it is completely ignored. As far as the BEA SNMP Agent Integrator is concerned, the only agent supporting `.1.3.6.1.2.1.1` is at port 2009.

  ```
  NON_SMUX_PEER 2008 * .1.3.6.1.2.1.1,ro,8
  NON_SMUX_PEER 2009 * .1.3.6.1.2.1.1,rw,5
  ```

  The default is `-1` if a priority is not specified. The BEA SNMP Agent Integrator reads through the file sequentially. When it comes to an object with a `-1` priority, it tries to assign it a priority of 20. If 20 has already been assigned to that MIB group (in another entry), it tries to assign 19. It keeps trying each successive lower number until it finds one that is not taken, or until it reaches 0, in which case an error message appears.

- In the following entries, the BEA SNMP Agent Integrator assigns a priority of 20 to the first entry and a priority of 19 to the second entry. Since 19 is a higher priority, the agent at port 2009 handles object `.1.3.6.1.2.1.1` and the first entry is ignored. As before, the access is `rw`, since the entry specifying `ro` access is ignored.

  ```
  NON_SMUX_PEER 2008 * .1.3.6.1.2.1.1,ro
  NON_SMUX_PEER 2009 * .1.3.6.1.2.1.1
  ```

- The MIB tree `.1.3.6.1.2.1.11` is a special case, called `mib2.snmp`. This MIB group is always handled by the BEA SNMP Agent Integrator itself, and should not be exported by any agent or subagent. Any registration at, above, or below this MIB tree is not permitted. For example, none of the following entries is permitted:

  ```
  NON_SMUX_PEER 2005 * .1.3.6.1.2.1
  NON_SMUX_PEER 2006 * .1.3.6.1.2.1.11
  NON_SMUX_PEER 2007 * .1.3.6.1.2.1.11.7
  ```

  The first includes `mib2.snmp`, the second specifies it exactly, and the third specifies a part of it.

  If an SNMP agent wants to support `mib2` (except for the `snmp` group, as it is not permitted), you need to enter each of the `mib2` subtrees explicitly:

  ```
  NON_SMUX_PEER 2002 * .1.3.6.1.2.1.1  .1.3.6.1.2.1.2
     .1.3.6.1.2.1.3  .1.3.6.1.2.1.4  .1.3.6.1.2.1.5
        .1.3.6.1.2.1.6  .1.3.6.1.2.1.7  .1.3.6.1.2.1.8
           .1.3.6.1.2.1.9  .1.3.6.1.2.1.10
  ```

Note: To continue an entry on another line, use a backslash. Make sure that there are no characters (other than the carriage return) immediately following the backslash.

- The BEA SNMP Agent Integrator supports row-level registration. Consider a table with five columns and two rows. Each item in the table is a separate object and can be identified by the column and row identifiers, in that order. The following entry:

  ```
  NON_SMUX_PEER 2000 * .1.3.6.1.4.1.140.100.5.1.2.1
  ```

  specifies row 1 of the second column of table object `.1.3.6.1.4.1.140.100.5.1`. Row 1 is not necessarily the first row. 1 is simply a unique identifier for the row.

  Notice that `.1.3.6.1.4.1.140.100.5.1.2` refers to the whole second column. To associate the two rows with different subagents, you need to specify each object in the row:

  ```
  NON_SMUX_PEER 2000 * .1.3.6.1.4.1.140.100.5.1.1.1
     .1.3.6.1.4.1.140.100.5.1.2.1  .1.3.6.1.4.1.140.100.5.1.3.1
        .1.3.6.1.4.1.140.100.5.1.4.1
           .1.3.6.1.4.1.140.100.5.1.5.1

  NON_SMUX_PEER 2002 * .1.3.6.1.4.1.140.100.5.1.1.2
     .1.3.6.1.4.1.140.100.5.1.2.2  .1.3.6.1.4.1.140.100.5.1.3.2
        .1.3.6.1.4.1.140.100.5.1.4.2
           .1.3.6.1.4.1.140.100.5.1.5.2
  ```

  The first entry lists the object in row 1 in each of the five columns. The second entry lists the object in row 2 in each of the five columns.

OID_CLASS Entry

This entry is used by the SMUX subagents. Normally, these SMUX subagents register all the MIB groups they know about with the SMUX master agent. But you can limit the MIB groups exported by the SMUX subagents. To do so, you need to add an `OID_CLASS` entry to `beamgr.conf` in the following format:

`OID_CLASS` *`agent_name OID_Node`*`[,ro|rw] [,`*`priority`*`] ..`

`OID_CLASS`

The keyword for the entry.

`agent_name`

Specifies the name of the SMUX subagent for which this entry is applicable.

`OID_Node`

Specifies the OID of the tree that is supported by this SMUX subagent.

`ro|rw`

Specifies whether this OID tree is being exported as read-only or as read-write. The default is rw.

`priority`

A positive number that specifies the sequence in which the OID tree is being exported. The lower the number, the higher the priority. If there are multiple subagents supporting the same MIB tree, the subagent with the highest priority is consulted. If the priority field is missing, the MIB tree is exported at the highest available priority. This entry is mainly used to limit the OID subtrees being exported by the BEA SNMP Agent Integrator SMUX subagents. If this entry is not present, the SMUX subagent exports all the MIB groups it knows about.

Multiple OID nodes can be listed.

RULE_ACTION Entry

The BEA SNMP Agent Integrator can be configured to manage locally on the managed node and inform the SNMP manager selectively to reduce polling traffic generated by the SNMP manager. The user can define rules in terms of MIB objects available locally using a C-like “IF” syntax, and accordingly send SNMP traps or execute commands locally (or both). The MIB object can be the one supported by the BEA SNMP Agent Integrator itself or the one supported by one of its SNMP agents or SMUX subagents. For a discussion of `RULE_ACTION` entries, with examples, see “Using the BEA SNMP Agent Integrator for Polling” on page 7-1.

The configuration is done in the `beamgr.conf` file. Syntax of the entry looks like the following:

```
RULE_ACTION rule_name frequency_in_secs
   if (VAL(oid) rel_operator value) logical_op ( cond_2 ) ...
      { TRAPID_ERR=enterprise_specific_trapid
         TRAPID_OK=enterprise_specific_trapid
            COMMAND_ERR=executable COMMAND_OK=executable }
```

Note: The whole entry should appear on the same line, else the backslash (\) should be used as a continuation character. If \ is being used as a continuation character, a newline character should immediately follow it.

`RULE_ACTION`

A keyword to identify this entry.

rule_name

A unique identifier for each RULE_ACTION entry that is passed as a command-line argument to any commands specified as actions in the rule. This identifier can be a maximum of eight characters long.

frequency

The polling frequency (in seconds) at which the snmp_integrator should check the condition.

VAL

Each left-hand side of a condition should have this keyword, which should be followed by the object identifier (within parentheses) of the MIB object. Each rule can contain a maximum of ten VAL keywords.

oid

An object identifier (OID). The OID must be specified only in numeric form and can be in one of the following formats:

– An absolute OID, that is, the full path to the object is specified from the root of the OID tree. An initial dot is used to indicate that the path starts at root. For example: .1.3.6.1.2.1.1.1.0. Note that the trailing zero in this example is the instance index.

– A relative OID under the MIB II branch can be specified in the form

mib-2.*number.number ...*

When the reserved word mib-2 appears as the leading sub-oid, .1.3.6.1.2.1. is assumed to be prefixed to the rest of the OID. For example:

mib-2.1.1.0

represents the absolute OID:

.1.3.6.1.2.1.1.1.0

– A relative OID under the enterprises branch can be specified in the form:

enterprises.number.number ...

When the reserved word enterprises appears as the leading sub-oid, .1.3.6.1.4.1. is assumed to be prefixed to the rest of the OID. For example:

enterprises.140.1.0

represents the absolute OID:

.1.3.6.1.4.1.140.1.0

– A relative OID under the enterprises branch can also be specified in purely numeric form:

`number.number.number ... ,`

If there is no leading "." and the OID starts with a number, `.1.3.6.1.4.1.` is assumed to be prefixed to the rest of OID. For example:

`140.1.1.0`

represents the absolute OID:

`.1.3.6.1.4.1.140.1.1.0`

Columnar objects are used to represent a column of a tabular MIB group. Columnar objects, accordingly, can have multiple instances. The last number in an OID is used to specify the particular instance. A specific number can be used to specify a particular instance or the asterisk (*) wildcard can be used to specify all instances. Zero is used as the instance index in the case of *scalar* objects (objects that can have only one instance). The asterisk wildcard is only used to represent all instances of a columnar object. For example:

`.1.3.6.1.4.1.140.1.1.0`

specifies the single instance of a scalar object while

`1.3.6.1.4.1.140.2.22.1.2.*`

specifies all of the instances of a columnar object.

Note: When you specify multiple OIDs in a complex rule, you should not combine an OID that specifies a particular instance with an OID that uses a wildcard in the same rule. Also, when you use multiple OIDs with wildcards in a single rule, all the OIDs should specify objects only within the same table.

`rel_operator`

See the table titled "Relations for Defining Conditions" on page 7-4 for a list of valid relational operators.

`value`

The RHS in a condition can be one of the following: number, string, IP address: number1.number2.number3.number4, or OID (as previously explained).

If RHS is an OID it must be enclosed in single quotes. Also, the type of value on RHS should correspond to the type of VALUE of the OID in LHS of the condition.

logical_op

See the table in "Use of Logical Operators in Conditions" on page 7-7 for a list of valid logical operators.

Specifying Actions

You can take two actions whenever there is a transition in the state of a rule from true (ERR) to false (OK) and false (OK) to true (ERR)—namely, execute a command and/or generate an SNMP trap. When generic OIDs (those that use an asterisk to specify all instances of a columnar object) are used to define a rule, the rule state transitions from the OK state to the ERR state if the threshold evaluates to true for any row in the table; and the rule state transitions from ERR to OK if the threshold evaluates to false for all rows in the MIB table. Initially, the rule states of all rules are set to OK when the BEA SNMP Agent Integrator starts up (or is re-initialized using the `reinit_agent` command). Specify actions using the following keywords:

`TRAPID_ERR = number`

Indicates that an enterprise-specific SNMP trap with trapid of *number* should be generated whenever there is a transition from false (OK) to true (ERR).

`TRAPID_OK = number`

Indicates that an enterprise-specific SNMP trap with trapid of *number* should be generated whenever there is a transition from true (ERR) to false (OK).

`COMMAND_ERR = command`

Indicates that *command* should be executed whenever there is a transition from false (OK) to true (ERR).

`COMMAND_OK = command`

Indicates that *command* should be executed whenever there is a transition from true (ERR) to false (OK).

Setting Up RULE_ACTION for Multiple Domains

The configuration file can contain multiple domains defined by multiple `TMAGENT`s. The instructions for setting up `RULE_ACTION` are as follows:

1. The `TMAGENT` entry defines the BEA Tuxedo domain that an agent monitors. There must be one `TMAGENT` entry for a Tuxedo SNMP agent on a single managed node.

   `TMAGENT logical_agent_name TUXDIR TUXCONFIG`

2. The RULE_ACTION entry is used to inform the SNMP manager of selective information gathered by the BEA SNMP Agent Integrator.

   ```
   RULE_ACTION rule_name frequency_in_secs
   ```

3. The optional *rule_name* component, *logical _agent_name*, must be appended to *rule_name* if multiple Tuxedo SNMP agents are running on the same node and the rule uses any Tuxedo MIB objects.

   ```
   RULE_ACTION rule_name@logical_agent_name frequency_in_seconds
      if (...
   ```

   For example:

   ```
   RULE_ACTION rule1@tux_snmpd10...
   ```

BEA SNMP Agent Passwords File: beamgr_snmpd.conf

Unlike the beamgr.conf configuration file, which is mandatory, the beamgr_snmpd.conf passwords file is an *optional* configuration file. Even if the BEA SNMP Agent Integrator and agents cannot find the beamgr_snmpd.conf file on the host system, they will still run *as long as* they can find the beamgr.conf file.

Default Location

The BEA SNMP Agent Integrator and agents look for beamgr_snmpd.conf at the following default location:

- For Windows systems: C:\etc\beamgr_snmpd.conf
- For UNIX systems: /etc/beamgr_snmpd.conf

Description

The beamgr_snmpd.conf passwords file contains the SNMP community strings used as passwords in communication between agents and managers. On a UNIX system, the beamgr_snmpd.conf passwords file is installed with access privileges for root only. For password security, the read and write permissions for the beamgr_snmpd.conf file should be set to permit access only by root.

A configuration entry in the beamgr_snmpd.conf file consists of two or more blank or tab-separated fields:

```
KEYWORD parameters
```

Recognized values for *KEYWORD* are:

COMMUNITY_RW

The string following this keyword specifies the read-write community for the agent. If this keyword is not present in the configuration file, the SNMP agent uses `iview` as the read-write community. Entries with this keyword can be repeated more than one time to specify more than one read-write community.

COMMUNITY_RO

The string following this keyword specifies the read-only community for the agent. If this keyword is not present in the configuration file, the SNMP agent uses `public` as the read-only community. Entries with this keyword can be repeated more than one time to specify more than one read-only community.

SMUX_PASSWD

The string following this keyword specifies the SMUX password. Any SMUX subagent that needs to register with the BEA SNMP Agent Integrator must specify this password. If this keyword is not present, the BEA SNMP Agent Integrator does not authenticate connection requests from SMUX subagents.

A SMUX subagent obtains the password at startup by reading the value specified by the `BEA_SMUX_PASSWD` environment variable. If this variable is not set, the SMUX subagent does not specify a password when registering.

DISABLE_SET

The possible values for this keyword are `YES` or `NO`, with `NO` being the default. If set to `YES`, Set access for all SNMP agents is disabled.

If the BEA SNMP Agent Integrator and agents cannot find the `beamgr_snmpd.conf` file on the host system, they will use the default values for `COMMUNITY_RW`, `COMMUNITY_RO`, `SMUX_PASSWD`, and `DISABLE_SET`.

APPENDIX A

SNMP Information

The following sections describe frequently asked questions and concerns about the SNMP protocol and management information bases (MIBs):

Reference Books

The following books provide additional information about MIBs, agents, or the SNMP protocol:

- Comer, Douglas; *Internetworking with TCP/IP, Vol. 2*; Prentice-Hall, Englewood Cliffs, New Jersey, 1991
- Leinwand, Allan and Fang, Karen; *Network Management: A Practical Perspective*; Addison-Wesley, Reading, Massachusetts, 1993

- Rose, Marshall T.; *The Simple Book: An Introduction to Management of TCP/IP-based Internets*; Prentice-Hall, Englewood Cliffs, New Jersey, 1991
- Rose, Marshall T.; *The Open Book: A Practical Perspective on Open Systems Interconnection*; Prentice-Hall, Englewood Cliffs, New Jersey, 1989
- Miller, Mark; *Managing Internetworks with SNMP*, M & T Books
- Stallings, William; *SNMP, SNMPv2 and CMIP: The Practical Guide to Network Management Standards*, Addison-Wesley, Reading, Massachusetts, 1993

Obtaining MIBs

The vendor-specific MIBs and MIBs under development in the following list are available via anonymous FTP at the indicated IP address:

- venera.isi.edu [128.9.0.32] in directory mib (for vendor MIBs)
- nic.ddn.mil [192.67.67.20] in directory internet drafts
- nnsc.nsf.net [192.31.103.6] in directory internet drafts
- munnari.oz.au [128.250.1.21] in directory internet drafts (Pacific Rim)
- nic.nordu.net [192.36.148.17] in directory internet drafts (Europe)

Enterprise ID Assignment

To develop your own private MIB, you must obtain an enterprise ID assignment from the Internet Assigned Numbers Authority.

Contact	Joyce K. Reynolds
Mailing Address	Internet Assigned Numbers Authority USC/Information Sciences Institute 4676 Admiralty Way Marina del Rey, CA 90292-6695
Phone	+1.323.822.1511
e-mail	iana@isi.edu

Obtaining Requests for Comments

You can obtain Requests for Comments (RFCs) in either of the following ways:

- Download them from almost anywhere on the Internet
- Obtain them from SRI International

Mailing Address	SRI International, EJ291 DDN Network Information Center 333 Ravenswood Ave. Menlo Park CA 94025
Phone	+1.800.235.3155
e-mail	MAIL-SERVER@nisc.sri.com Leave the subject field blank. In the body, enter: SEND RFCnnnn.TXT-1
FTP	ftp://ftp.nisc.sri.com/rfc/rfcNNNN.txt

Obtaining Specifications

Refer to the following sources for SNMP specifications:

- IEEE and ISO/IEC[IEEE] Standards

Mailing Address	Service Center 445 Hoes Lane PO Box 1331 Piscataway NJ 08855-1331
Phone	+1.800.678.4333

- IEEE drafts

Mailing Address	IEEE Computer Society Documents c/o AlphaGraphics ATTN: P. Thrush 10215 N. 35th Ave., Suite A & B Phoenix AZ 85051

- ISO and ISO/IEC documents

Mailing Address	American National Standards Institute 1430 Broadway New York NY 10018 USA

- ITU-T (formerly CCITT) Blue Book documents (OSI NMF documents):

FTP	ftp://bruno.cs.colordo.edu ftp://gatekeeper.dec.com/pub/bruno.cs.colorado.edu/pub/standards ftp://ftp.uu.net/doc/standards
FTP	Europe: ftp://src.doc.ic.ac.uk/doc/ccitt-standards ftp://nic.ja.net/doc/ccitt-standards

Mailing Lists and News Groups

The following news and mail groups provide general information about SNMP. You can also contact BEA SNMP Agent Customer Support. (See the customer support card included in your BEA SNMP Agent product box).

SNMP in general
snmp-request@uu.psi.com

RMON MIB
rmonmib-request@lexcel.com

Security issues
snmp-sec-dev-request@tis.com

Device discovery
finder-request@emerald.acc.com

Standards and Drafts

The following standards and drafts are available for SNMP:

RFC Number	Description
052	IAB Recommendations
1089	SNMP over Ethernet
1109	Ad-hoc Review
1155	Structure of Management Information
1156	Management Information Base (MIB-I)
1157	SNMP
1161	SNMP over OSI
1187	Bulk table retrieval
1212	Concise MIB definitions
1213	Management Information Base (MIB-II)
1214	OSI MIB

RFC Number	Description
1215	Traps
1227	SNMP Multiplex (SMUX)
1228	SNMP-DPI
1229	Generic-interface MIB extensions
1230 IEEE 802.4	Token Bus MIB
1231 IEEE 802.5	Token Ring MIB
1239	Reassignment of MIBs
1243	AppleTalk MIB
1248	OSPF MIB
ISO 8824	ASN.1
ISO 8825	BER for ASN.1

Accessing Internet Drafts

Internet drafts are available by anonymous FTP. Log in with the username `anonymous` and password `guest`. After logging in, type `cd internet-drafts`.

Internet drafts directories are located at:

US East Coast	ftp://ds.intermic.net/internet-drafts
US West Coast	ftp://ftp.isi.edu/internet-drafts
Pacific Rim	ftp://munnari.oz.au/internet-drafts
Europe	ftp://nic.nordu.net/internet-drafts

Internet drafts are also available by mail. Send e-mail to mailserv@ds.intermic.net. In the body, type:

```
FILE/internet-drafts-ietf-rdbmsmib-mib-02.txt
```

For questions, please send e-mail to:

`Internet-Drafts@cnri.seston.va.us`