PEER NETWORKS

PATROL® SNMP Toolkit

OptiMate Encapsulator Guide

Printing 1.8

October 24, 1996

BMC® SOFTWARE

This Page Left Blank Intentionally

Chapter 1 — Introduction

PATROL® SNMP Toolkit's *OptiMate*™ Encapsulator™ extends the toolkit's Object Development Environment by providing a way to encapsulate proprietary SNMP agents. *OptiMate* provides an integrated environment for hosts that also run an off-the-shelf or custom monolithic, non-extensible, or incompatible agent or proxy. This Guide introduces and explains *OptiMate* and leads you through its installation, configuration, operation, trouble-shooting and customization.

OptiMate is distributed as part of the PATROL SNMP Toolkit package, and is installed with it. The distribution includes a `Makefile`, source code and a sample configuration file for *OptiMate*.

This Introduction provides an overview of the PATROL SNMP Toolkit management architecture and how *OptiMate* relates; it explains how *OptiMate* may be used and how the package components are distributed; it summarizes related products and documents, and refers to related reference material; it outlines the rest of this document, lists typographical conventions this Guide uses, and tells how to contact PEER Networks division Technical Support.

1.1 PATROL SNMP Toolkit Management Architecture Related to *OptiMate*

The *OptiMaster* master agent uses SNMP to communicate with SNMP Network Management Stations (NMS) on behalf of your applications and devices (see Figure 1). This is accomplished with the *OptiGen* and *OptiPro* tools, enhancing applications and devices with *sub-agents*. *OptiMate* is a specialized sub-agent application. NMS SNMP requests are forwarded by *OptiMaster* to the appropriate sub-agent, whether it's one developed with PATROL SNMP Toolkit tools, or it's *OptiMate* handling incompatible agents. *OptiMate* uses the same *OptiPro* management API and run-time libraries that you use when you develop sub-agents for your own applications and devices.

OptiMate's application is an SNMP pass-through service for incompatible agents on the same host as *OptiMaster*. When *OptiMate* receives SNMP requests forwarded by the Master Agent, it re-formulates them as SNMP requests to the appropriate "encapsulated" incompatible agent. The reverse operation is used for incompatible agents' SNMP responses and traps.

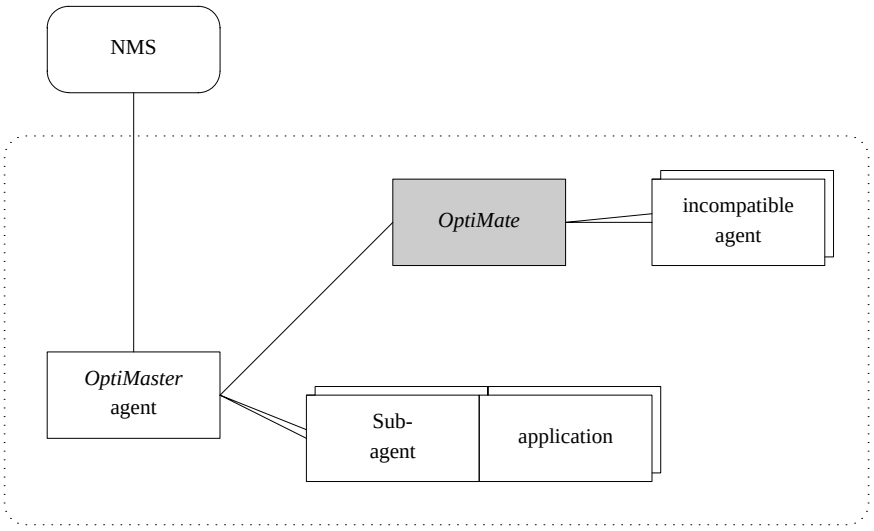

Figure 1. PATROL SNMP Toolkit Management Architecture

1.2 Using *OptiMate* Encapsulator

OptiMate is installed on hosts that contain *OptiMaster* and one or more incompatible agents. The NMS sends all SNMP requests for all managed variables on that host to *OptiMaster*, which forwards the relevant requests to *OptiMate*.

1.2.1 How *OptiMate* Works

Most Network Management Stations view managed objects on a host through a single SNMP agent, as shown previously in Figure 1. The NMS will usually direct its SNMP requests to an agent listening at port 161. Since only a single SNMP agent can be listening at port 161, this limits the NMS to managing only the variables accessible to the one agent listening at that port. You will be configuring *OptiMaster* to listen at port 161, so that *all* queries directed to that host will go to *OptiMaster*. This will include variables from your own MIBs (managed by your sub-agents), and variables managed by the incompatible agents on the same hosts. The incompatible agent(s) are invoked listening at alternate port(s), not at 161. Then *OptiMate* is configured to pass on the appropriate SNMP requests from *OptiMaster* to the encapsulated agents at their respective ports. *OptiMate* works by hiding the incompatible agents, so they are visible to the NMS only through *OptiMaster*.

You configure *OptiMate* to recognize which incompatible agents are managing which sub-trees (a sub-tree is the set of MIB variables subordinate to a node in the naming tree), through which SNMP ports, and which traps they're sending. Then when an incompatible agent sends a trap, *OptiMate* is listening for the trap and forwards it up to *OptiMaster* or discards it, as configured. When *OptiMaster* receives an NMS SNMP request about an encapsulated agent's managed sub-tree, it passes it on to *OptiMate* which then forwards it to the encapsulated agent, at that agent's listening port. *OptiMate* emulates an NMS — passing NMS requests through to the encapsulated agent as SNMP requests from an NMS, and receiving the agent traps as an NMS. *OptiMate* supports any number of encapsulated agents. (It need not be on the host running *OptiMaster* and the encapsulated agents.)

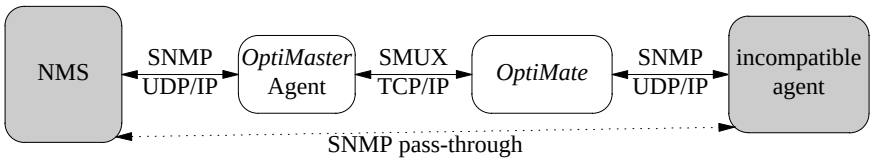

Figure 2. How *OptiMate* Works

Follow the *OptiMate* installation, configuration and start-up instructions later in this Guide to get your management configuration working smoothly. Using *OptiMate* does not require any change to *OptiMaster* that also supports your sub-agents.

1.2.2 How to Use *OptiMate*

OptiMate is configured to filter SNMP requests before forwarding them to the incompatible agent. In the reverse direction, *OptiMate* is configured to either forward up all traps sent by the incompatible agent, or filter some of them out. How *OptiMate* is configured determines the NMS view of what's manageable on the network, as explained in the examples below.

If *OptiMate* is configured to forward requests for all of an agent's sub-trees, and to forward all traps emitted by an agent, the NMS has access to all variables managed by that agent on that host. This would enable full access to both a MIB-II agent and a hardware device agent on the same host, for example.

While *OptiMaster* is supporting *OptiMate* as configured above, *OptiMaster* also supports sub-agents, including those developed with PATROL SNMP Toolkit tools.

You might choose to configure *OptiMate* to filter out some of an agent's traps to reduce the number of traps received by an NMS.

OptiMate could also be configured to filter out some of the sub-trees managed by an incompatible agent, hiding those variables from the NMS. The same variables could be managed by a sub-agent developed with the PATROL SNMP Toolkit Object Development Environment tools *OptiGen* and *OptiPro*.

1.3 How You Distribute the Package Components

OptiMate is built on your development system using code on this distribution and libraries and files from the Object Development Environment. It is then configured and started up on the target host. For more detail, see the chapter on installation and start-up.

1.4 Related Products and Documentation

OptiMate, a specialized PATROL SNMP Toolkit sub-agent, requires an SNMP master agent that uses SMUX, like the *OptiMaster* master agent. *OptiMate* is built with run-time libraries and include files that are components of the Object Development Environment distributed with *OptiMaster*. For more information on using and installing the Agent and Development Environment, see the PATROL SNMP Toolkit *Programmer's Guide*, Document 20009. The *SNMP Porting Guide*, Document 20010, covers *OptiMaster* Master Agent configuration, tracing, and porting the run-time libraries to other operating systems. *Release Notes* are Document 20038. If you need additional help, contact PEER Networks division Technical Support.

1.5 Reference Material

Documents defining SNMP, SNMP's management information structure, SMUX and related subjects are available as Internet Requests for Comments (RFCs). See the *Programmer's Guide* for a list of relevant documents and how to get them.

1.6 How this Guide is Organized

Chapter 2 explains how to install *OptiMate*.

Chapter 3 covers how to configure, invoke and operate *OptiMate*.

Chapter 4 discusses some operational issues.

Chapter 5 covers customization.

1.7 Typographical Conventions

The following typographical conventions are used throughout this document.

- When a term is first defined, it is shown in *italics*.
- When actual "C" code is given, in fragments, programs or synopses, it is shown in `Courier` font.
- In text, names of functions (or variables used in referenced code) are shown in `Courier` font, as in `mgmt_new_instance()`.
- Throughout the document, **bold** is used to call attention to additions, extensions, or differences.

1.8 PEER Networks division Technical Support

Technical Support is available from our California offices on business days from 9:00a.m. to 6:00p.m., Pacific time.

Telephone:	+1 408 556-0720
FAX:	+1 408 556-0735
Electronic mail:	support@peer.com

Chapter 2 — Installing *OptiMate*

OptiMate™ is installed along with *OptiMaster*, *OptiGen*, and *OptiPro*. If the PATROL® SNMP Toolkit Object Development Environment has not yet been installed, follow the *General Porting Guide* to install and configure it.

2.1 Installation

The `Makefile` in `/usr/peer/subagts/encaps/src/wrk` builds *OptiMate*. It is automatically invoked when the PATROL SNMP Toolkit package is installed. This does not require root or super-user permission. The following information may be relevant for re-building *OptiMate* without re-building the remainder of the toolkit.

`Makefile` requires the following from the PATROL distribution directory:

- header files from `include/ame`, `include/ode` and `common/ode/src/wrk`
- compilation script `devtools/peercc`
- run-time libraries `devtools/PEERmgmt.o`
- MIB compiler `devtools/gdmoc`

To re-build the package, run the `Makefile` as follows:

```
% cd /usr/peer/subagts/encaps/src/wrk
% make
```

Several steps of the `make` will not normally execute because their output files are also provided on the distribution. (The MIB Compiler in `devtools/gdmoc` creates `proxy.h`; `yacc` creates `ytab.c` and `ytab.h`; `lex` creates `lexyy.c`) The complete `Makefile` is provided in case you want to customize *OptiMate*, the *OptiMate* MIB and/or *OptiMate* configuration file grammar, tokens or processing. Refer to Chapter 5 for more information. Please contact PEER Networks division Technical Support for customization assistance.

2.2 Compatibility with Master Agents

The *OptiMate* executable, built with the specified *OptiPro* run-time libraries and *OptiGen* MIB compiler is compatible with PATROL SNMP Toolkit and **OPTIMA** *OptiMaster* master agents built from Release 1.6 and later distributions. For details, see the *Release Notes*, Document 20038.

Chapter 3 — *OptiMate* Start-up and Operation

OptiMate™ is invoked with a number of command line parameters. These control *OptiMate*'s communications with the master agent, *OptiMaster*, and *OptiMate*'s encapsulated agents, and give the name of *OptiMate*'s configuration file.

OptiMate's configuration file identifies the incompatible agent(s) to encapsulate, what ports they're listening at, which of their sub-trees to manage, and which traps to forward.

3.1 Invoking *OptiMate*

The *OptiMate* executable is built as `Program.o` under the directory `subagts/encaps/src/wrk`. There are two ways to invoke it:

- invoke *OptiMate* manually as follows:

```
% cd /usr/peer/subagts/encaps/src/wrk
% ./Program.o [parmlist] &
```

- or create a script that includes the following lines and invoke this script from `/etc/rc.local`:

```
cd /usr/peer/subagts/encaps/src/wrk
./Program.o [parmlist] &
```

The parameters are explained in the following section.

OptiMate needs to run with super-user or root permission only if it listens to privileged ports, e.g., port 161 or port 162.

OptiMaster must be started before *OptiMate*. Start encapsulated agent(s) before or after *OptiMate*; but all must be executing for the agents' managed objects to be visible to the NMS.

3.2 *OptiMate* Command Line Parameters

The are six command line parameters. Figure 3 is a guide to the parameters described below.

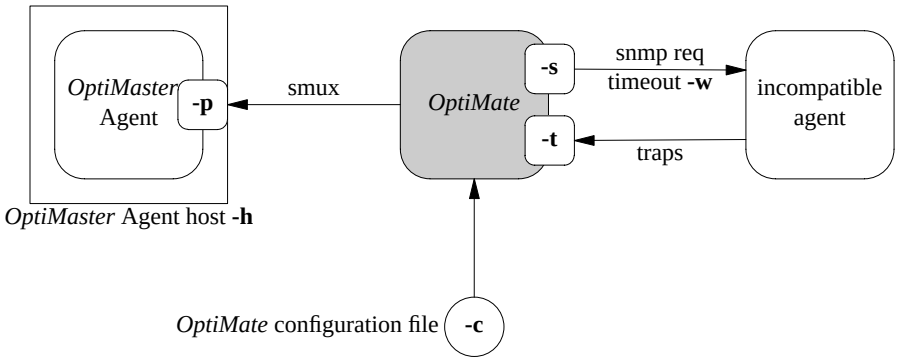

Figure 3. ***OptiMate*** command line parameters

-t port — *OptiMate*'s incoming trap port, i.e., where *OptiMate* listens to receive SNMP traps sent by encapsulated agents.
Example: `-t 1162`
Default: no traps forwarded

-s port — *OptiMate*'s sending SNMP request port, i.e., the port *from* which *OptiMate* sends SNMP requests to encapsulated agents, and to which the encapsulated agents send their SNMP responses. This is *not* the port at which the encapsulated agent is *listening*, specified in the configuration file.
Example: `-s 3361`
Default: assigned by the host

-h host — The *OptiMaster* host address, i.e., the host where the master agent resides, normally the same host as *OptiMate*'s host. This is *not* the host of the *encapsulated* agent(s), specified in the configuration file.
Example: `-h dorothy`
Example: `-h 192.146.153.3`
Default: `localhost`

-p port — *OptiMaster*'s listening SMUX port, i.e., the port to which *OptiMate* sends its SMUX traffic.
Example: `-p 199`
Default: `/etc/services` SMUX port, if provided, else `SMUX_LISTEN_PORT` from `include/ame/portnums.h`

-w wait — timeout, in seconds, for the SNMP response from the encapsulated agent.
Example: `-w 3`
Default: 2 seconds

-c file — *OptiMate*'s configuration file name.
Example: `-c myconfig`
Default: `CONFIG`

The configuration file is described in the following section. Be sure to review Chapter 4 — Operational Issues as you plan your port assignments.

3.3 *OptiMate* Configuration File

The *OptiMate* configuration file specifies which agents are to be encapsulated, and what to make visible to the NMS. The name of the file is specified in the `-c` parameter when *OptiMate* is invoked. If no `-c` parameter is specified, the name "`CONFIG`" is used. Review Chapter 4 — Operational Issues, as you plan your encapsulation strategy.

To cause Encapsulator to reread an (altered) configuration file, send the Encapsulator process or task a `HUP` (hangup) signal, where available, with the `kill -HUP <process_id>` command. On Windows platforms use the `Reload Configuration` menu item in the Encapsulator window.

The *OptiMate* configuration file consists of one or more `AGENT` entries, each terminated with a semi-colon. The `AGENT` entry must include a `SUBTREES` entry and may include a `FORWARD TRAPS` entry. The overall configuration file grammar is:

```
[
AGENT [ON HOST <ip>] [AT PORT <port>] [WITH COMMUNITY <community>]
SUBTREES <treelist>
[FORWARD <traplist> TRAPS]
;
]+
```

`AGENT` entries establish which incompatible agents are encapsulated. Any agent that is to be encapsulated must have an `AGENT` entry. Each `AGENT` entry defines the host, i.e., processor, supporting that encapsulated agent; at which port the encapsulated agent is listening for incoming SNMP requests; and which community string *OptiMate* should use on SNMP requests sent to that agent. The order of `AGENT` entries is not important except for agents that are managing the same sub-tree(s). See the following sub-section on this point.

The `AGENT HOST` entry provides either the encapsulated agent's hostname or its IP address. If the HOST entry is omitted, *OptiMate*'s own host is the default. Note this is different from the host parameter `-h` on the command line, which specifies the host of *OptiMate*'s master agent, the *OptiMaster* Agent.

The `AGENT PORT` entry states the encapsulated agent's SNMP listening port. This must state the port the encapsulated agent is configured to listen at, so that *OptiMate* can correctly forward NMS SNMP requests. If the PORT entry is omitted, the SNMP port in in `include/ame/portnums.h` is used. Note this port is different from the SNMP port parameter `-s` on the command line, which specifies the port *OptiMate* uses to issue its SNMP requests, the port to which the encapsulated agent *returns* its SNMP responses.

OptiMate uses the `AGENT COMMUNITY` entry to code a community string in all SNMP request messages it sends to the encapsulated agent. (The `COMMUNITY` entries in the *OptiMaster* Agent configuration file may need to be adjusted, since authorization is first performed by the *OptiMaster* Agent.) If the community string is incorrectly stated, the encapsulated agent will ignore some or all of the incoming requests (possibly issuing an authentication trap). The community string may be given as a quoted or unquoted string or in hexadecimal format. The `COMMUNITY` entry may be omitted entirely, in which case "`public`" is used.

The detailed grammar of the items in the AGENT entry is:

```
AGENT [ON HOST <ip>] [AT PORT <port>] [WITH COMMUNITY <community>]
ip        ::=  <hostname> | <ipaddr>
ipaddr          ::=  <d>.<d>.<d>.<d>
d         ::=  0 .. 255
port      ::=  1 .. 65535
community ::=  <string> | <hexstring>
hexstring ::=  0x<hexdigits>
```

The AGENT SUBTREES entry establishes which of the sub-trees managed by the encapsulated agent should be made visible to, i.e., manageable by, the NMS. For an encapsulated agent's sub-trees to be visible to the NMS, the SUBTREES list must include those mounted sub-trees' object ids. *OptiMate* will register each listed sub-tree with the master agent, making it available for NMS-issued GETs and SETs. In order to have traps forwarded from an encapsulated agent, include both the SUBTREES entry for that agent and one FORWARD TRAPS entry for any agent on that agent's host. Encapsulator forwards traps for every agent on a single host using the first FORWARD TRAPS entry for any agent on that host. Any subsequent FORWARD TRAPS entry for any agent on that host is ignored. The SUBTREES treelist consists of one or more object ids, separated by commas. The grammar for SUBTREES is:

```
SUBTREES <treelist>
treelist       ::=  <oid> | <oid> , <treelist>
oid            ::=  <decimal> | <decimal>.<oid>
decimal        ::=  0 .. 4294967295
```

The first FORWARD TRAPS entry for all agents on a host defines which traps sent by encapsulated agents on a host will be forwarded by *OptiMate* to Encapsulator's Master Agent. Since any other FORWARD TRAPS entries for agents on that same host are ignored, include only one such entry for all encapsulated agents on that host. If no FORWARD TRAPS entry is provided for a host, Encapsulator forwards none of the traps emitted by any encapsulated agent on that host. The entry may take one of four constructions. If *all* traps emitted by any agent on that host should be forwarded, the entry should consist of "FORWARD ALL TRAPS". If *none* of the host's agents' traps are to be forwarded, the entry should consist of "FORWARD NO TRAPS". If *only certain types* of traps are to be forwarded, the entry should consist of "FORWARD ONLY <kindlist> TRAPS", where "kindlist" is the list of the kinds of traps to be forwarded. If *all traps except certain types* of traps are to be forwarded, the entry should consist of "FORWARD ALL EXCEPT <kindlist> TRAPS", where "kindlist" is the list of the kinds of traps *OptiMate* will omit when forwarding traps it receives. (The MANAGER entry in the Master Agent's configuration file controls the ultimate trap destination.)

The grammar for FORWARD TRAPS is:

```
[FORWARD <traplist> TRAPS]
traplist       ::=  ALL | NO | <onlylist> | <exceptlist>
onlylist       ::=  ONLY <kindlist>
exceptlist     ::=  ALL EXCEPT <kindlist>
kindlist       ::=  <kind> | <kind> , <kindlist>
kind           ::=  <enterprise> | <enterprise> <generic>
enterprise     ::=  ANY | ENTERPRISE <oid>
generic        ::=  coldStart | warmStart | linkDown | linkUp |
                    authenticationFailure | egpNeighborLoss | <specific>
specific       ::=  enterpriseSpecific (<s>)
s              ::=  <d> | <range>
d              ::=  0..255
range          ::=  <d> .. <d>
```

The sample file subagts/encaps/src/wrk/CONFIG illustrates how to set up a configuration that encapsulates a MIB-II agent on the same host as *OptiMate* and the *OptiMaster* Master Agent. The master agent will manage the SNMP group. The sample,

extracted below, will almost certainly *not* work in your environment unless you change it! If you omit the `-c` command line parameter, the `CONFIG` file is used.

```
#    Sample OptiMate Config File for MIB-II agent
#
#    It will be responsible for
#    the first 8 groups of MIB-II.
#
#    The agent is listening at port 1161.
#    It will accept SNMP requests with "public"
#    community string.
#    It is on the same host as OptiMate.
#    OptiMate is to forward all of its traps.
#
AGENT AT PORT 1161 WITH COMMUNITY public
SUBTREES  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
FORWARD ALL TRAPS;
```

The two configuration file `FORWARD TRAPS` entries below are independent examples of trap filtering, illustrating how the grammar may be used.

```
1.   FORWARD ALL EXCEPT
     ENTERPRISE 1.3.6.1.2,
     ENTERPRISE 1.3.6.1.7 coldStart,
     ENTERPRISE 1.2.3.4.5.6.7.8 enterpriseSpecific 1..100
     TRAPS

2.   FORWARD ONLY
     ANY coldStart,
     ANY linkUp,
     ENTERPRISE 2.3.4.5.6.7.8.3 enterpriseSpecific 777,
     ANY linkDOWN TRAPS
```

3.4 Managing Multiply-Mounted Sub-Trees

If a sub-tree is registered more than once, two SMUX mechanisms control which registration takes precedence, priority numbers assigned at registration time, and the status of each registration. How these impact multiply-mounted subtrees is explained below.

When a sub-agent, including *OptiMate*, mounts, or registers, a sub-tree, that sub-tree registration gets an SMUX priority. The later the registration, the lower the priority number, and the higher the priority. If more than one sub-agent registers the same sub-tree, the most recent registration gets a lower priority number. The lower the number, the higher the priority. The priorities may be viewed through variables in the SMUX MIB, under `1.3.6.1.4.1.4.4.` See RFC 1227 for more information. You will see an SMUX row for the *OptiMate* sub-agent, and one for each encapsulated agent.

Each sub-tree registration has a status ("valid" or "invalid") associated with it. If there are multiple registrations of the same sub-tree, management of the sub-tree will go to the sub-agent whose registration has the highest priority (the lowest priority number) of those with valid status.

3.4.1 Sub-Trees Mounted by More than One Encapsulated Agent

As *OptiMate* reads the `AGENT` entries, it registers the sub-trees listed in the `SUBTREE` entry. A sub-tree listed in a later `AGENT` entry will be registered with a higher priority than one listed earlier. If the same sub-tree is mounted by more than one encapsulated agent, the last `AGENT` entry's sub-tree is the one visible to the NMS. You can control the priority by changing the order of the `AGENT` entries. The order in which the agents are invoked, or whether they are executing, is irrelevant.

3.4.2 Sub-Trees Mounted by both the *OptiMaster* Agent and an Encapsulated Agent

Often, an encapsulated agent manages MIB-II, including the system group and the SNMP group. The *OptiMaster* Agent's internal sub-agent also manages those two groups. Since the *OptiMaster* Agent is invoked before *OptiMate*, the *OptiMaster* Agent's groups will be registered before the encapsulated agent's groups. Therefore, the encapsulated agent's registered system and SNMP sub-trees will get higher priority, thus making them visible to the NMS rather than the sub-trees registered by the *OptiMaster* Agent. Then, should *OptiMate* be terminated, its sub-tree registrations are deleted, thus leaving the *OptiMaster* Agent's registrations with the highest active priority. The *OptiMaster* Agent will thus take over management of the sub-tree(s), until *OptiMate* is restarted and re-registers its configured group(s).

If it is desirable instead, to have the *OptiMaster* Agent manage one or both of these MIB-II groups while *OptiMate* is active, omit those groups from the *OptiMate* configuration file.

3.4.3 Sub-Trees Mounted by an Encapsulated Agent and a Sub-Agent

If both an encapsulated agent and a custom sub-agent mount, or register the same sub-tree, the order in which *OptiMate* and the sub-agent are invoked will determine the registration order and therefore the management priority, and ultimately which manages that sub-tree. If both registrations have status "valid" and the managing sub-agent (which might be *OptiMate*) is terminated, that respective registration is removed, and the remaining registration will now have the highest priority, switching management to the remaining sub-agent until the previously managing sub-agent is restarted.

3.5 The *OptiMate* MIB

OptiMate is managed through its own enterprise MIB, defined in concise MIB format in the file `proxy.smi` in `subagts/encaps/src/wrk`. This enterprise MIB group has the `1.3.6.1.4.1.442.1.2` object identifier. The first three variables in the MIB are the SNMP timeout, the trap listening port and the SNMP sending port values (0 if defaulted) provided on the command line in the `-w, -t` and `-s` parameters:

```
timeout
trapPort
snmpPort
```

The next variable, `badTrapSources`, refers to traps that *OptiMate* received from agents on hosts not supporting any encapsulated agents, due to inconsistent port definitions. Traps that were received from encapsulated agents but that were badly formed are counted in `badTrapSyntaxes`:

```
badTrapSources
badTrapSyntaxes
```

Following these is the alienTable describing the encapsulated agents, or “aliens”. The columns are shown below. The `alienEntryIndex` is a number assigned to the agents in the order of the `AGENT` entries in the configuration file. (This index is distinct from the index in the SMUX table.) `alienEntryIp`, `alienEntryPort` and `alienEntryCommunity` are taken from the configuration file `AGENT` entry. `alienEntryCommands` counts the number of SNMP requests that *OptiMate* forwarded to the agent according to the SUBTREES entry; `alienEntryResponses` counts the SNMP responses received. If the encapsulated agent is not responding because the community string is incorrect, `alienEntryResponses` will reflect this. `alienEntryTrapForwarding` is a code representing the type of `FORWARD TRAPS` entry provided in the configuration file — ALL traps (1) ; NO traps (2) ; ALL EXCEPT certain traps (3) ; ONLY certain traps (4) are being forwarded.

Traps received by Encapsulator are counted on a host basis, so the trap PDU counts shown for the first encapsulated agent on a host reflect the total for all agents on that host. (If port configuration is inconsistent, some of these traps may have been sent by unencapsulated agents.) `alienEntryDiscardedTraps` counts traps that were received from agents on that host but were filtered out according to the first `FORWARD TRAPS` entry for encapsulated agents on that host. `alienEntryForwardedTraps` counts those that were forwarded on to the Master Agent.

```
alienEntryIndex
alienEntryIp
alienEntryPort
alienEntryCommunity
alienEntryDiscardedTraps
alienEntryForwardedTraps
alienEntryCommands
alienEntryResponses
alienEntryTrapForwarding
```

The `subtreeTable` follows, representing the managed, or mounted, sub-trees as listed in the configuration file and the sequence number of the encapsulated agent that manages that sub-tree.

```
subtreeEntryOid
subtreeEntryAlien
```

Following `subtreeTable` in the MIB is the `trapTable`, giving the trap filters for each encapsulated agent, determined from the configuration file. The table is indexed by the sequential number of the encapsulated agent and an index for each trap kind listed in the `FORWARD TRAPS` entry for that agent. The `Min` and `Max` variables represent the lowest and highest values of the generic and enterprise-specific trap types that match their respective filters in the trap kind list. For example, if the `FORWARD TRAPS` entry is “`FORWARD ONLY ANY warmStart TRAPS`” then `trapEntryMinGeneric` and `trapEntryMaxGeneric` would be “1”; if it were “`FORWARD ONLY ANY linkDown TRAPS`” then `trapEntryMinGeneric` and `trapEntryMaxGeneric` would be “2”. For both entries, `trapEntryMinSpecific` would be “0” and `trapEntryMaxSpecific` would be “2147483647”, since no filtering on specific trap values was specified.

```
trapEntryAlienIndex
trapEntryIndex
trapEntryEnterprise
trapEntryMinGeneric
trapEntryMaxGeneric
trapEntryMinSpecific
trapEntryMaxSpecific
```

This Page Left Blank Intentionally

Chapter 4 — Operational Issues

As a summary of configuration and start-up, and to aid in trouble-shooting, this chapter reviews some configuration and start-up issues that may affect operation. (In trouble-shooting, first verify that both *OptiMate*™ and the encapsulated agents are executing on their intended hosts.) Figure 4 below reviews the command line parameters and the entries in the configuration file. Command line parameters are shown in standard font, while relevant configuration file entries are shown as `<entry>`.

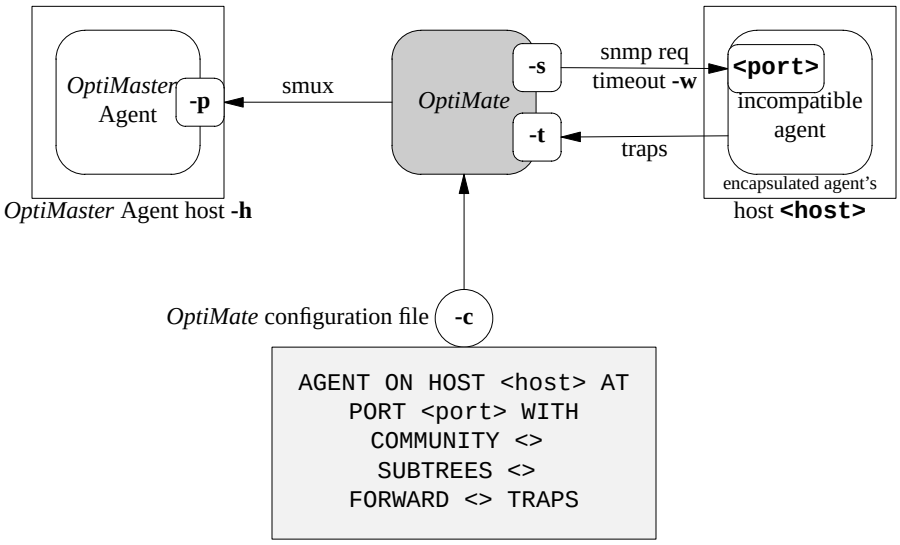

Figure. 4 *OptiMate* command line parameters and configuration file entries

4.1 Port Assignments

An encapsulated agent must be defined in an `AGENT` entry in the *OptiMate* configuration file. The port at which the executing encapsulated agent is listening for SNMP requests must be identical to the SNMP port specified for that agent in the AGENT entry in the configuration file. This is different from the SNMP `-s` port command-line parameter,

which is the port *OptiMate sends SNMP from*, and the port at which *OptiMate listens for returned SNMP responses* from encapsulated agents.

Omitting the agent SNMP port in the *OptiMate* configuration file will cause the listening port to be assigned from the `include/ame/portnums.h` file. This is the same file used the generate the *OptiPro* run-time library used by *OptiMaster* and other sub-agents use when port assignments are defaulted. Therefore it is dangerous to default SNMP port numbers in the *OptiMate* configuration file. Also, the default port is usually privileged, requiring super-user permission for *OptiMate*.

The *OptiMate* MIB variable `alienEntryPort` represents, for each encapsulated agent, the port *OptiMate* is sending its SNMP requests to, as specified in the `AGENT PORT` entry. The *OptiMate* MIB variables `trapPort` and `snmpPort` show the ports designated on *OptiMate*'s command line with `-t` and `-s`, respectively.

OptiMate ports, whether selected explicitly on the command line, or defaulted, need to be reviewed carefully for conflict with ports being used by other networking applications *on the same host*. These include, but are not limited to *OptiMaster*, other agents, and the NMS. Be aware that *OptiMaster* retrieves its port selections first from its configuration files, then from `/etc/services`, and finally from the *OptiPro* default configured when the run-time library was built, based on `portnums.h` in `include/ame`. Many NMS also use `/etc/services` to identify ports to use.

Also, be sure that the *OptiMate* command designates the same *OptiMaster* Agent SMUX listening port as your own sub-agents do. (Like *OptiMaster*, each sub-agent built with PATROL® SNMP Toolkit tools defaults to use first its respective host's `/etc/services` entry, and then the `portnums.h` entry).

If encapsulated agents are not participating as expected, look for port assignment errors or conflicts.

DO NOT ATTEMPT TO ENCAPSULATE THE *OptiMaster* AGENT!!

4.2 Host Assignments

The host on which the encapsulated agent is executing must match the host specified (or defaulted) in the `AGENT HOST` entry for that encapsulated agent, in the configuration file. The default host for an encapsulated agent is *OptiMate*'s host. The *OptiMate* MIB variable `alienEntryIp` represents the host for each respective encapsulated agent, as specified in the `AGENT HOST` entry. Note the host command-line parameter `-h` represents the host of the *OptiMaster* Agent that *OptiMate* uses, not the host of any encapsulated agent.

4.3 Community String Assignments

Each SNMP request includes a community string for authorization purposes. Each `AGENT COMMUNITY` entry states the string that *OptiMate* places in the requests sent to that agent. Agents ignore requests arriving with unauthorized strings (some circumstances will result in authentication traps being generated). The *OptiMate* MIB variable `alienEntryCommunity` represents the string assigned for an encapsulated agent, as specified in the `AGENT` entry for that agent. If `COMMUNITY` was not provided, "`public`" will be used.

OptiMaster receives the SNMP requests. These are forwarded via *OptiMate* to the encapsulated agents. The original community string *OptiMaster* receives is not passed through. You may need to augment the current `COMMUNITY` entry in *OptiMaster*'s configuration file to accept community strings used for SNMP requests targeted at encapsulated agents.

4.4 Sub-Tree Accessibility

There is a SUBTREE entry for each AGENT entry in *OptiMate*'s configuration file, indicating which sub-trees of that agent should be registered with *OptiMaster*, and thus made visible to the NMS. The values are available in the *OptiMate* MIB table SubtreeEntry. If other encapsulated agents and/or custom sub-agents mount, or register, the same sub-tree, the most recent registration takes priority. See the previous chapter's section on this subject for more detail.

Incorrect port assignments (see the previous section) can make a sub-tree inaccessible. Of course, *OptiMate* and the managing agent must both be executing for encapsulation to work.

4.5 Trap Filtering

The port at which *OptiMate* is listening for traps (specified using -t on *OptiMate*'s command line and recorded in trapPort in *OptiMate*'s MIB) must match the port to which all encapsulated agents are sending traps. If the trap port is not specified on the command line, no traps will be received, therefore no traps will be forwarded. Also beware of port conflicts, as mentioned above.

If there is no FORWARD TRAPS entry specified in that agent's AGENT entry, or for any other encapsulated agent on that host, no traps will be forwarded. If the first FORWARD TRAPS entry for agents on that host is incorrect, the results will be incorrect.

If any agent on a host is encapsulated, traps will be accepted and filtered from every agent on that host, whether encapsulated or not. If none of the encapsulated agent entries for that host's agents has a FORWARD TRAPS entry, no trap received from any agent on that host will be forwarded, whether encapsulated or not.

The MANAGER entry in *OptiMaster*'s configuration file may need to be adjusted to complete the trap-forwarding process correctly.

The *OptiMate* MIB has several variables that represent the active configuration as well as the current trap counts, as mentioned in the previous chapter's section on the MIB:

badTrapSources	traps from agents on hosts with no encapsulated agents
badTrapSyntaxes	badly formed trap PDUs
alienEntryDiscardedTraps	all traps filtered out for that host (shown
alienEntryForwardedTraps	traps forwarded
alienEntryTrapForwarding	type of FORWARD TRAP list
TrapEntry table	columns indicate the filters in effect

4.6 Multiple Variable Bindings on a SET

A single SNMP SET request is permitted to contain multiple variable bindings. *OptiMaster* coordinates SMUX sub-agent requests and responses so that if one variable can't be set, none of them are set, even if multiple sub-agents are involved. However, since *SNMP protocol* cannot support atomicity across two or more agents, atomicity cannot be preserved across encapsulated agents. In other words, encapsulating SNMP agents cannot in itself add functionality not available under SNMP.

This Page Left Blank Intentionally

Chapter 5 — Customizing *OptiMate*

OptiMate™ is provided in source form so it may be customized if necessary. The configuration file structure can be altered easily since it is processed by a front-end to *OptiMate* that works off a defined grammar. Also, it might be desirable to change *OptiMate*'s MIB. If you do customize *OptiMate*, be sure to change *OptiMate*'s object id (in files `main.c` and `proxy.smi`) to use your own enterprise id rather than that of the PEER Networks division. Please contact PEER Networks division Technical Support for assistance.

5.1 Using `lex` and `yacc` to Define Configuration File Processing

OptiMate's parsing logic is not hard-coded, but resides in files generated using the `lex` and `yacc` tools. The grammar of the configuration file and/or its processing logic may be customized if you have access to `lex` and `yacc`. The `lex` input file `aliencfg.l` defines the tokens of the file, while the `yacc` input file `aliencfg.y` specifies the parsing rules. If you modify either of these files, rebuild *OptiMate* by rerunning `make`. The `lex` and `yacc` output files `lexyy.c`, `ytab.c` and `ytab.h` (see the next section) are automatically recreated and used to rebuild *OptiMate*.

5.2 Changing *OptiMate's MIB*

OptiMate's MIB in concise MIB template form is in the file `proxy.smi`, described in a previous section. If you should need to change the MIB definition, modify the object ids for the MIB and for *OptiMate* and rerun `make` for *OptiMate*. The `Makefile` will cause the MIB compiler in `devtools/gdmoc` to recreate `proxy.h` and rebuild the executable.

Index

Special —

A —

B —

C —

D —

E —

F —

G —

U —

V —

W —

Y —

Table of Contents

PEER Networks, a division of BMC Software, Inc.

Document No. 20013

PEER Networks, a division of BMC Software, Inc.
1190 Saratoga Avenue, Suite 130
San José, CA 95129-3433 USA
Telephone +1 408 556-0720