



- [Table of Contents](#)
- [Index](#)
- [Reviews](#)
- [Examples](#)
- [Reader Reviews](#)
- [Errata](#)

Essential SNMP

By Douglas Mauro, Kevin Schmidt

Publisher : O'Reilly
Pub Date : July 2001
ISBN : 0-596-00020-0
Pages : 291

This practical guide for network and system administrators introduces SNMP along with the technical background to use it effectively. But the main focus is on practical network administration: how to configure SNMP agents and network management stations, how to use SNMP to retrieve and modify variables on network devices, how to configure management software to react to traps sent by managed devices. Covers all SNMP versions through SNMPv3.

Copyright © 2001 O'Reilly & Associates, Inc. All rights reserved.

Printed in the United States of America.

Published by O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol, CA 95472.

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly & Associates, Inc. Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly & Associates, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps. The association between the image of red deer and the topic of SNMP is a trademark of O'Reilly & Associates, Inc.

While every precaution has been taken in the preparation of this book, the publisher assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

Preface

The Simple Network Management Protocol (SNMP) is an Internet-standard protocol for managing devices on IP networks. Many kinds of devices support SNMP, including routers, switches, servers, workstations, printers, modem racks, and uninterruptible power supplies (UPSs). The ways you can use SNMP range from the mundane to the exotic: it's fairly simple to use SNMP to monitor the health of your routers, servers, and other pieces of network hardware, but you can also use it to control your network devices and even send pages or take other automatic action if problems arise. The information you can monitor ranges from relatively simple and standardized items, like the amount of traffic flowing into or out of an interface, to more esoteric hardware- and vendor-specific items, like the air temperature inside a router.

Given that there are already a number of books about SNMP in print, why write another one? Although there are many books on SNMP, there's a lack of books aimed at the practicing network or system administrator. Many books cover how to implement SNMP or discuss the protocol at a fairly abstract level, but none really answers the network administrator's most basic questions: How can I best put SNMP to work on my network? How can I make managing my network easier?

We provide a brief overview of the SNMP protocol in [Chapter 2](#) then spend a few chapters discussing issues such as hardware requirements and the sorts of tools that are available for use with SNMP. However, the bulk of this book is devoted to discussing, with real examples, how to use SNMP for system and network administration tasks.

Most newcomers to SNMP ask some or all of the following questions:

- What exactly is SNMP?
- How can I, as a system or network administrator, benefit from SNMP?
- What is a MIB?
- What is an OID?
- What is a community string?
- What is a trap?

- I've heard that SNMP is insecure. Is this true?
- Do any of my devices support SNMP? If so, how can I tell if they are configured properly?
- How do I go about gathering SNMP information from a device?
- I have a limited budget for purchasing network-management software. What sort of free/open source software is available?
- Is there an SNMP Perl module that I can use to write cool scripts?

This book answers all these questions and more. Our goal is to demystify SNMP and make it more accessible to a wider range of users.

Audience for This Book

This book is intended for system and network administrators who could benefit from using SNMP to manage their equipment but who have little or no experience with SNMP or SNMP applications. In our experience almost any network, no matter how small, can benefit from using SNMP. If you're a Perl programmer, this book will give you some ideas about how to write scripts that use SNMP to help manage your network. If you're not a Perl user you can use many of the other tools we present, ranging from Net-SNMP (an open source collection of command-line tools) to Hewlett Packard's OpenView (a high-end, high-priced network-management platform).

Organization

[Chapter 1](#) provides a nontechnical overview of network management with SNMP. We introduce the different versions of SNMP as well as the concepts of managers and agents.

[Chapter 2](#) discusses the technical details of SNMP. We look at the Structure of Management Information (SMI) and the Management Information Base (MIB) and discuss how SNMP actually works; i.e., how management information is sent and received over the network.

[Chapter 3](#) helps you to think about strategies for deploying SNMP.

[Chapter 4](#) discusses what it means when a vendor says that its equipment is "SNMP-compatible."

[Chapter 5](#) introduces some of the available network-management software. We discuss the pros and cons of each package and provide pointers to vendors' web sites. We include both commercial and open source packages in the discussion.

[Chapter 6](#) provides a basic understanding of what to expect when installing NMS software by looking at two NMS packages, HP's OpenView and Castle Rock's SNMPc.

[Chapter 7](#) describes how to configure the Windows SNMP agent and several SNMP agents for Unix, including the Net-SNMP agent. To round the chapter out, we discuss how to configure the embedded agents on two network devices: the Cisco SNMP agent and the APC Symetra SNMP agent.

[Chapter 8](#) shows how you can use command-line tools and Perl to gather (poll) SNMP information and change (set) the state of a managed device.

[Chapter 9](#) discusses how to configure OpenView and SNMPc to gather SNMP information via polling. This chapter also discusses RMON configuration on a Cisco router.

[Chapter 10](#) examines how to send and receive traps using command-line tools, Perl, OpenView, and other management applications.

[Chapter 11](#) shows how several popular SNMP agents can be extended. Extensible agents provide end users with a means to extend the operation of an agent without having access to the agent's source code.

[Chapter 12](#) is geared toward Perl-savvy system administrators. We provide Perl scripts that demonstrate how to perform some common system-administration tasks with SNMP.

[Chapter 13](#) introduces one of the most widely used open source SNMP applications, the Multi Router Traffic Grapher (MRTG). MRTG provides network administrators with web-based usage graphs of router interfaces and can be configured to graph many other kinds of data.

[Appendix A](#) discusses how to use OpenView to graph input and output octets.

[Appendix B](#) discusses how to graph external data with Network Node Manager (NNM), add menu items to NNM, configure user profiles, and use NNM as a centralized communication interface.

[Appendix C](#) summarizes the usage of the Net-SNMP command-line tools.

[Appendix D](#) provides an authoritative list of the various RFC numbers that pertain to SNMP.

[Appendix E](#) is a good summary of the SNMP Perl module used throughout the book.

[Appendix F](#) provides a brief introduction to SNMPv3. Two configuration examples are provided: configuring SNMPv3 on a Cisco router and configuring SNMPv3 for Net-SNMP.

Example Programs

All the example programs in this book are available at <http://www.oreilly.com/catalog/esnmp/>.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic

Used for commands, object IDs, URLs, filenames, and directory names. It is also used for emphasis and for the first use of technical terms.

Constant width

Used for examples, object definitions, literal values, and datatypes. It is also used to show source code, the contents of files, and the output of commands.

Constant width bold

Used in interactive examples to show commands or text that would be typed literally by the user. It is also used to emphasize when something, usually in source code or file-contents examples, has been added to or changed from a previous example.

Constant width italic

Used for replaceable parameter names in command syntax.



Indicates a tip, suggestion, or general note.



Indicates a warning or caution.

Comments and Questions

Please address comments and questions concerning this book to the publisher:

O'Reilly & Associates, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (in the United States or Canada)
(707) 829-0515 (international/local)
(707) 829-0104 (fax)

There is a web page for this book, which lists errata, the text of several helpful technical papers, and any additional information. You can access this page at:

<http://www.oreilly.com/catalog/esnmp/>

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about books, conferences, software, Resource Centers, and the O'Reilly Network, see the O'Reilly web site at:

<http://www.oreilly.com>

Acknowledgments

It would be an understatement to say that this book was a long time in the making. It would never have been published without the patience and support of Michael Loukides. Thanks Mike! We would also like to thank the individuals who provided us with valuable technical review feedback and general help and guidance: Mike DeGraw-Bertsch at O'Reilly & Associates; Donald Cooley at Global Crossing; Jacob Kirsch at Sun Microsystems, Inc.; Bobby Krupczak, Ph.D., at Concord Communications; John Reinhardt at Road Runner; Patrick Bailey and Rob Sweet at Netrail; and Jürgen Schönwälder at the Technical University of Braunschweig. Rob Romano, O'Reilly & Associates graphic artist, deserves a thank you for making the figures throughout the book look great. Finally, thanks to Jim Sumser, who took the project over in its final stages, and to Rachel Wheeler, the production editor, for putting this book together.

Douglas

For years I worked as a system and network administrator and often faced the question, "How are things running?" This is what led me to SNMP and eventually the idea for this book. Of course I would like to thank Kevin for his hard work and dedication. Special thanks go to the two special girls in my life: my wife, Amy, and our daughter, Kari, for putting up with my long absences while I was writing in the computer room. Thanks also go to my family and friends, who provided support and encouragement.

Kevin

While at MindSpring Enterprises (now Earthlink) I was fortunate enough to work for Allen Thomas, who gave me the freedom to explore my technical interests, including SNMP. I would like to thank Bobby Krupczak for providing me with valuable feedback on the SystemEDGE agent. Thanks also to my colleagues Patrick Bailey and Rob Sweet at Netrail, who provided some general Perl code feedback. I'm very fortunate to have worked with Douglas on this book; thanks for allowing me to help out. My parents deserve a thank you for buying me my first computer all those years ago. And finally, I would like to thank Callie, my

significant other, for allowing me to use our nights and weekends to work on this book.

Chapter 1. What Is SNMP?

In today's complex network of routers, switches, and servers, it can seem like a daunting task to manage all the devices on your network and make sure they're not only up and running but performing optimally. This is where the *Simple Network Management Protocol* (SNMP) can help. SNMP was introduced in 1988 to meet the growing need for a standard for managing Internet Protocol (IP) devices. SNMP provides its users with a "simple" set of operations that allows these devices to be managed remotely.

This book is aimed toward system administrators who would like to begin using SNMP to manage their servers or routers, but who lack the knowledge or understanding to do so. We try to give you a basic understanding of what SNMP is and how it works; beyond that, we show you how to put SNMP into practice, using a number of widely available tools. Above all, we want this to be a practical book -- a book that helps you keep track of what your network is doing.

1.1 Network Management and Monitoring

The core of SNMP is a simple set of operations (and the information these operations gather) that gives administrators the ability to change the state of some SNMP-based device. For example, you can use SNMP to shut down an interface on your router or check the speed at which your Ethernet interface is operating. SNMP can even monitor the temperature on your switch and warn you when it is too high.

SNMP usually is associated with managing routers, but it's important to understand that it can be used to manage many types of devices. While SNMP's predecessor, the *Simple Gateway Management Protocol* (SGMP), was developed to manage Internet routers, SNMP can be used to manage Unix systems, Windows systems, printers, modem racks, power supplies, and more. Any device running software that allows the retrieval of SNMP information can be managed. This includes not only physical devices but also software, such as web servers and databases.

Another aspect of network management is *network* monitoring; that is, monitoring an entire network as opposed to individual routers, hosts, and other devices. *Remote Network Monitoring* (RMON) was developed to help us understand how the network itself is functioning, as well as how individual devices on the network are affecting the network as a whole. It can be used to monitor not only LAN traffic, but WAN interfaces as well. We

discuss RMON in more detail later in this chapter and in [Chapter 2](#).

Before going any further, let's look at a before-and-after scenario that shows how SNMP can make a difference in an organization.

1.1.1 Before and After SNMP

Let's say that you have a network of 100 machines running various operating systems. Several machines are file servers, a few others are print servers, another is running software that verifies credit card transactions (presumably from a web-based ordering system), and the rest are personal workstations. In addition, there are various switches and routers that help keep the actual network going. A T1 circuit connects the company to the global Internet, and there is a private connection to the credit card verification system.

What happens when one of the file servers crashes? If it happens in the middle of the workweek, it is likely that the people using it will notice and the appropriate administrator will be called to fix it. But what if it happens after everyone has gone home, including the administrators, or over the weekend?

What if the private connection to the credit card verification system goes down at 10 p.m. on Friday and isn't restored until Monday morning? If the problem was faulty hardware and could have been fixed by swapping out a card or replacing a router, thousands of dollars in web site sales could have been lost for no reason. Likewise, if the T1 circuit to the Internet goes down, it could adversely affect the amount of sales generated by individuals accessing your web site and placing orders.

These are obviously serious problems -- problems that can conceivably affect the survival of your business. This is where SNMP comes in. Instead of waiting for someone to notice that something is wrong and locate the person responsible for fixing the problem (which may not happen until Monday morning, if the problem occurs over the weekend), SNMP allows you to monitor your network constantly, even when you're not there. For example, it will notice if the number of bad packets coming through one of your router's interfaces is gradually increasing, suggesting that the router is about to fail. You can arrange to be notified automatically when failure seems imminent, so you can fix the router before it actually breaks. You can also arrange to be notified if the credit card processor appears to get hung -- you may even be able to fix it from home. And if nothing goes wrong, you can return to the office on Monday morning knowing there won't be any surprises.

There might not be quite as much glory in fixing problems before they occur, but you and your management will rest more

easily. We can't tell you how to translate that into a higher salary -- sometimes it's better to be the guy who rushes in and fixes things in the middle of a crisis, rather than the guy who makes sure the crisis never occurs. But SNMP does enable you to keep logs that prove your network is running reliably and show when you took action to avert an impending crisis.

1.1.2 Human Considerations

Implementing a network-management system can mean adding more staff to handle the increased load of maintaining and operating such an environment. At the same time, adding this type of monitoring should, in most cases, reduce the workload of your system-administration staff. You will need:

- Staff to maintain the management station. This includes ensuring the management station is configured to properly handle events from SNMP-capable devices.
- Staff to maintain the SNMP-capable devices. This includes making sure that workstations and servers can communicate with the management station.
- Staff to watch and fix the network. This group is usually called a *Network Operations Center* (NOC) and is staffed 24/7. An alternative to 24/7 staffing is to implement rotating pager duty, where one person is on call at all times, but not necessarily present in the office. Pager duty works only in smaller networked environments, in which a network outage can wait for someone to drive into the office and fix the problem.

There is no way to predetermine how many staff members you will need to maintain a management system. The size of the staff will vary depending on the size and complexity of the network you're managing. Some of the larger Internet backbone providers have 70 or more people in their NOCs, while others have only one.

1.2 RFCs and SNMP Versions

The *Internet Engineering Task Force* (IETF) is responsible for defining the standard protocols that govern Internet traffic, including SNMP. The IETF publishes *Requests for Comments* (RFCs), which are specifications for many protocols that exist in the IP realm. Documents enter the standards track first as proposed standards, then move to draft status. When a final draft is eventually approved, the RFC is given standard status -- although there are fewer completely approved standards than you might think. Two other standards-track designations, historical and experimental, define (respectively) a document that has been replaced by a newer RFC and a document that is not yet ready to become a standard. The following list includes all the current SNMP versions and the IETF status of each (see [Appendix D](#) for a full list of the SNMP RFCs):

- *SNMP Version 1 (SNMPv1)* is the current standard version of the SNMP protocol. It's defined in RFC 1157 and is a full IETF standard. SNMPv1's security is based on *communities*, which are nothing more than passwords: plain-text strings that allow any SNMP-based application that knows the strings to gain access to a device's management information. There are typically three communities in SNMPv1: read-only, read-write, and trap.
- *SNMP Version 2 (SNMPv2)* is often referred to as community string-based SNMPv2. This version of SNMP is technically called SNMPv2c, but we will refer to it throughout this book simply as SNMPv2. It's defined in RFC 1905, RFC 1906, and RFC 1907, and is an experimental IETF. Even though it's experimental, some vendors have started supporting it in practice.
- *SNMP Version 3 (SNMPv3)* will be the next version of the protocol to reach full IETF status. It's currently a proposed standard, defined in RFC 1905, RFC 1906, RFC 1907, RFC 2571, RFC 2572, RFC 2573, RFC 2574, and RFC 2575. It adds support for strong authentication and private communication between managed entities. [Appendix F](#) provides an introduction to SNMPv3 and goes through the SNMPv3 agent configuration for Net-SNMP and Cisco. The information in this appendix provides any system or network administrator with the practical knowledge needed to begin using SNMPv3 as it gains acceptance in the network-management world.

The official site for RFCs is <http://www.ietf.org/rfc.html>. One of the biggest problems with RFCs, however, is finding the one you want. It is a little easier to navigate the RFC index at Ohio State University (<http://www.cis.ohio-state.edu/services/rfc/index.html>).

1.3 Managers and Agents

In the previous sections we've vaguely referred to SNMP-capable devices and network-management stations. Now it's time to describe what these two things really are. In the world of SNMP there are two kind of entities: *managers* and *agents*. A manager is a server running some kind of software system that can handle management tasks for a network. Managers are often referred to as *Network Management Stations (NMSs)*.^[1] An NMS is responsible for polling and receiving traps from agents in the network. A *poll*, in the context of network management, is the act of querying an agent (router, switch, Unix server, etc.) for some piece of information. This information can later be used to determine if some sort of catastrophic event has occurred. A *trap* is a way for the agent to tell the NMS that something has happened. Traps are sent asynchronously, not in response to queries from the NMS. The NMS is further responsible for performing an action^[2] based upon the information it receives from the agent. For example, when your T1 circuit to the Internet goes down, your router can send a

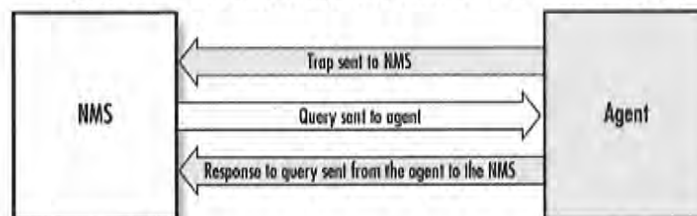
trap to your NMS. In turn, the NMS can take some action, perhaps paging you to let you know that something has happened.

^[1] See [Chapter 5](#) for a pro-and-con discussion of some popular NMS applications.

^[2] Note that the NMS is preconfigured to perform this action.

The second entity, the *agent*, is a piece of software that runs on the network devices you are managing. It can be a separate program (a daemon, in Unix language), or it can be incorporated into the operating system (for example, Cisco's IOS on a router, or the low-level operating system that controls a UPS). Today, most IP devices come with some kind of SNMP agent built in. The fact that vendors are willing to implement agents in many of their products makes the system administrator's or network manager's job easier. The agent provides management information to the NMS by keeping track of various operational aspects of the device. For example, the agent on a router is able to keep track of the state of each of its interfaces: which ones are up, which ones are down, etc. The NMS can query the status of each interface on a router, and take appropriate action if any of them are down. When the agent notices that something bad has happened, it can send a trap to the NMS. This trap originates from the agent and is sent to the NMS, where it is handled appropriately. Some devices will send a corresponding "all clear" trap when there is a transition from a bad state to a good state. This can be useful in determining when a problem situation has been resolved. [Figure 1-1](#) shows the relationship between the NMS and an agent.

Figure 1-1. Relationship between an NMS and an agent



It's important to keep in mind that polls and traps can happen at the same time. There are no restrictions on when the NMS can query the agent or when the agent can send a trap.

1.4 The Structure of Management Information and MIBS

The *Structure of Management Information* (SMI) provides a way to define managed objects and their behavior. An agent has in its possession a list of the objects that it tracks. One such object is the operational status of a router interface (for example, up, down, or testing). This list collectively defines

the information the NMS can use to determine the overall health of the device on which the agent resides.

The *Management Information Base* (MIB) can be thought of as a database of managed objects that the agent tracks. Any sort of status or statistical information that can be accessed by the NMS is defined in a MIB. The SMI provides a way to define managed objects, while the MIB is the definition (using the SMI syntax) of the objects themselves. Like a dictionary, which shows how to spell a word and then gives its meaning or definition, a MIB defines a textual name for a managed object and explains its meaning. [Chapter 2](#) goes into more technical detail about MIBs and the SMI.

An agent may implement many MIBs, but all agents implement a particular MIB called MIB-II ^[3] (RFC 1213). This standard defines variables for things such as interface statistics (interface speeds, MTU, octets^[4] sent, octets received, etc.) as well as various other things pertaining to the system itself (system location, system contact, etc.). The main goal of MIB-II is to provide general TCP/IP management information. It doesn't cover every possible item a vendor may want to manage within its particular device.

^[3] MIB-I is the original version of this MIB, but it is no longer referred to since MIB-II enhances it.

^[4] An octet is an 8-bit quantity, which is the fundamental unit of transfer in TCP/IP networks.

What other kinds of information might be useful to collect? First, there are many draft and proposed standards developed to help manage things such as frame relay, ATM, FDDI, and services (mail, DNS, etc.). A sampling of these MIBs and their RFC numbers includes:

- ATM MIB (RFC 2515)
- Frame Relay DTE Interface Type MIB (RFC 2115)
- BGP Version 4 MIB (RFC 1657)
- RDBMS MIB (RFC 1697)
- RADIUS Authentication Server MIB (RFC 2619)
- Mail Monitoring MIB (RFC 2249)
- DNS Server MIB (RFC 1611)

But that's far from the entire story, which is why vendors, and individuals, are allowed to define MIB variables for their own use.^[5] For example, consider a vendor that is bringing a new router to market. The agent built into the router will respond to NMS requests (or send traps to the NMS) for the variables defined by the MIB-II standard; it probably also implements

MIBs for the interface types it provides (e.g., RFC 2515 for ATM and RFC 2115 for Frame Relay). In addition, the router may have some significant new features that are worth monitoring but are not covered by any standard MIB. So, the vendor defines its own MIB (sometimes referred to as a proprietary MIB) that implements managed objects for the status and statistical information of their new router.

^[5] This topic is discussed further in the next chapter.



Simply loading a new MIB into your NMS does not necessarily allow you to retrieve the data/values/objects, etc. defined within that MIB. You need to load only those MIBs supported by the agents from which you're requesting queries (e.g., *snmpget*, *snmpwalk*). Feel free to load additional MIBs for future device support, but don't panic when your device doesn't answer (and possibly returns errors for) these unsupported MIBs.

1.5 Host Management

Managing host resources (disk space, memory usage, etc.) is an important part of network management. The distinction between traditional system administration and network management has been disappearing over the last decade, and is now all but gone. As Sun Microsystems puts it, "The network is the computer." If your web server or mail server is down, it doesn't matter whether your routers are running correctly -- you're still going to get calls. The Host Resources MIB (RFC 2790) defines a set of objects to help manage critical aspects of Unix and Windows systems.^[6]

^[6] Any operating system running an SNMP agent can implement Host Resources; it's not confined to agents running on Unix and Windows systems.

Some of the objects supported by the Host Resources MIB include disk capacity, number of system users, number of running processes, and software currently installed. In today's e-commerce world, more and more people are relying on service-oriented web sites. Making sure your backend servers are functioning properly is as important as monitoring your routers and other communications devices.

Unfortunately, some agent implementations for these platforms do not implement this MIB, since it's not required.

1.6 A Brief Introduction to Remote Monitoring (RMON)

Remote Monitoring Version 1 (RMONv1, or RMON) is defined in RFC 2819; an enhanced version of the standard, called RMON Version 2 (RMONv2), is defined in RFC 2021. RMONv1 provides the NMS with packet-level statistics about an entire LAN or WAN. RMONv2 builds on RMONv1 by providing network- and application-level statistics. These statistics can be gathered in several ways. One way is to place an RMON probe on every network segment you want to monitor. Some Cisco routers have limited RMON capabilities built in, so you can use their functionality to perform minor RMON duties. Likewise, some 3Com switches implement the full RMON specification and can be used as full-blown RMON probes.

The RMON MIB was designed to allow an actual RMON probe to run in an offline mode that allows the probe to gather statistics about the network it's watching without requiring an NMS to query it constantly. At some later time, the NMS can query the probe for the statistics it has been gathering. Another feature that most probes implement is the ability to set thresholds for various error conditions and, when a threshold is crossed, alert the NMS with an SNMP trap. You can find a little more technical detail about RMON in the next chapter.

1.7 Getting More Information

Getting a handle on SNMP may seem like a daunting task. The RFCs provide the official definition of the protocol, but they were written for software developers, not network administrators, so it can be difficult to extract the information you need from them. Fortunately, many online resources are available. The most notable web site is the Network Management Server at the University at Buffalo (<http://netman.cit.buffalo.edu>). It contains useful links to other sites that provide similar information, as well as a network-management product list (<http://netman.cit.buffalo.edu/Products.html>) that includes both software and hardware vendors; it even has product reviews. This site is a great starting point in the search for network-management information and can be an extremely useful tool for determining what kinds of hardware and software are currently out there. Two more great web sites are the SimpleWeb (<http://www.snmp.cs.utwente.nl>) and SNMP Link (<http://www.SNMPLink.org>). The Simple Times, an online publication devoted to SNMP and network management, is also useful. You can find the current edition, and all the previous ones, at <http://www.simple-times.org>.

Another great resource is Usenet news. The newsgroup most people frequent is comp.dcom.net-management. Another good newsgroup is comp.protocols.snmp. Groups such as these promote a community of information sharing, allowing seasoned

professionals to interact with individuals who are not as knowledgeable about SNMP or network management.

If you would like to know if a particular vendor has SNMP-compatible equipment, the *Internet Assigned Numbers Authority* (IANA) has compiled a list of the proprietary MIB files various vendors supply. The list can be found at <ftp://ftp.iana.org/mib/>. There is also an SNMP FAQ, available in two parts at <http://www.faqs.org/faqs/snmp-faq/part1/> and <http://www.faqs.org/faqs/snmp-faq/part2/>.

Chapter 2. A Closer Look at SNMP

In this chapter, we start to look at SNMP in detail. By the time you finish this chapter, you should understand how SNMP sends and receives information, what exactly SNMP communities are, and how to read MIB files. We'll also look in more detail at the three MIBs that were introduced in [Chapter 1](#), namely MIB-II, Host Resources, and RMON.

2.1 SNMP and UDP

SNMP uses the *User Datagram Protocol* (UDP) as the transport protocol for passing data between managers and agents. UDP, defined in RFC 768, was chosen over the *Transmission Control Protocol* (TCP) because it is connectionless; that is, no end-to-end connection is made between the agent and the NMS when *datagrams* (packets) are sent back and forth. This aspect of UDP makes it unreliable, since there is no acknowledgment of lost datagrams at the protocol level. It's up to the SNMP application to determine if datagrams are lost and retransmit them if it so desires. This is typically accomplished with a simple timeout. The NMS sends a UDP request to an agent and waits for a response. The length of time the NMS waits depends on how it's configured. If the timeout is reached and the NMS has not heard back from the agent, it assumes the packet was lost and retransmits the request. The number of times the NMS retransmits packets is also configurable.

At least as far as regular information requests are concerned, the unreliable nature of UDP isn't a real problem. At worst, the management station issues a request and never receives a response. For traps, the situation is somewhat different. If an agent sends a trap and the trap never arrives, the NMS has no way of knowing that it was ever sent. The agent doesn't even know that it needs to resend the trap, because the NMS is not required to send a response back to the agent acknowledging receipt of the trap.

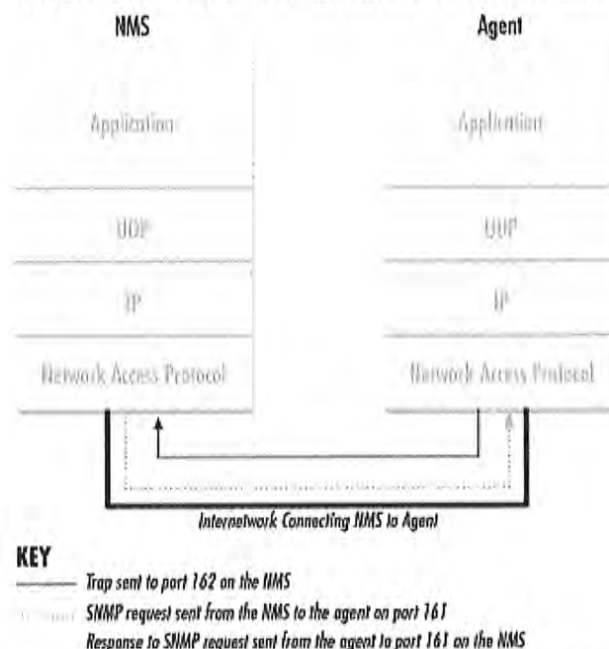
The upside to the unreliable nature of UDP is that it requires low overhead, so the impact on your network's performance is

reduced. SNMP has been implemented over TCP, but this is more for special-case situations in which someone is developing an agent for a proprietary piece of equipment. In a heavily congested and managed network, SNMP over TCP is a bad idea. It's also worth realizing that TCP isn't magic, and that SNMP is designed for working with networks that are in trouble -- if your network never failed, you wouldn't need to monitor it. When a network is failing, a protocol that tries to get the data through but gives up if it can't is almost certainly a better design choice than a protocol that will flood the network with retransmissions in its attempt to achieve reliability.

SNMP uses the UDP port 161 for sending and receiving requests, and port 162 for receiving traps from managed devices. Every device that implements SNMP must use these port numbers as the defaults, but some vendors allow you to change the default ports in the agent's configuration. If these defaults are changed, the NMS must be made aware of the changes so it can query the device on the correct ports.

[Figure 2-1](#) shows the TCP/IP protocol suite, which is the basis for all TCP/IP communication. Today, any device that wishes to communicate on the Internet (e.g., Windows NT systems, Unix servers, Cisco routers, etc.) must use this protocol suite. This model is often referred to as a protocol stack, since each layer uses the information from the layer directly below it and provides a service to the layer directly above it.

Figure 2-1. TCP/IP communication model and SNMP



When either an NMS or an agent wishes to perform an SNMP function (e.g., a request or trap), the following events occur in the protocol stack:

Application

First, the actual SNMP application (NMS or agent) decides what it's going to do. For example, it can send an SNMP request to an agent, send a response to an SNMP request (this would be sent from the agent), or send a trap to an NMS. The application layer provides services to an end user, such as an operator requesting status information for a port on an Ethernet switch.

UDP

The next layer, UDP, allows two hosts to communicate with one another. The UDP header contains, among other things, the destination port of the device to which it's sending the request or trap. The destination port will either be 161 (query) or 162 (trap).

IP

The IP layer tries to deliver the SNMP packet to its intended destination, as specified by its IP address.

Medium Access Control (MAC)

The final event that must occur for an SNMP packet to reach its destination is for it to be handed off to the physical network, where it can be routed to its final destination. The MAC layer is comprised of the actual hardware and device drivers that put your data onto a physical piece of wire, such as an Ethernet card. The MAC layer also is responsible for receiving packets from the physical network and sending them back up the protocol stack so they can be processed by the application layer (SNMP, in this case).

This interaction between SNMP applications and the network is not unlike that between two pen pals. Both have messages that need to be sent back and forth to one another. Let's say you decide to write your pen pal a letter asking if she would like to visit you over the summer. By deciding to send the invitation, you've acted as the SNMP application. Filling out the envelope with your pen pal's address is equivalent to the function of the UDP layer, which records the packet's destination port in the UDP header; in this case it's your pen pal's address. Placing a stamp on the envelope and putting it in the mailbox for the mailman to pick up is equivalent to the IP layer's function. The final act occurs when the mailman comes to your house and picks up the letter. From here the letter will be routed to its final destination, your pen pal's mailbox. The MAC layer of a computer network is equivalent to the mail trucks and airplanes that carry your letter on its way. When your pen pal receives the letter, she will go through the same process to send you a reply.

2.2 SNMP Communities

SNMPv1 and SNMPv2 use the notion of communities to establish trust between managers and agents. An agent is configured with

three community names: read-only, read-write, and trap. The community names are essentially passwords; there's no real difference between a community string and the password you use to access your account on the computer. The three community strings control different kinds of activities. As its name implies, the read-only community string lets you read data values, but doesn't let you modify the data. For example, it allows you to read the number of packets that have been transferred through the ports on your router, but doesn't let you reset the counters. The read-write community is allowed to read and modify data values; with the read-write community string, you can read the counters, reset their values, and even reset the interfaces or do other things that change the router's configuration. Finally, the trap community string allows you to receive traps (asynchronous notifications) from the agent.

Most vendors ship their equipment with default community strings, typically public for the read-only community and private for the read-write community. It's important to change these defaults before your device goes live on the network. (You may get tired of hearing this because we say it many times, but it's absolutely essential.) When setting up an SNMP agent, you will want to configure its trap destination, which is the address to which it will send any traps it generates. In addition, since SNMP community strings are sent in clear text, you can configure an agent to send an SNMP authentication-failure trap when someone attempts to query your device with an incorrect community string. Among other things, authentication-failure traps can be very useful in determining when an intruder might be trying to gain access to your network.

Because community strings are essentially passwords, you should use the same rules for selecting them as you use for Unix or NT user passwords: no dictionary words, spouse names, etc. An alphanumeric string with mixed upper- and lowercase letters is generally a good idea. As mentioned earlier, the problem with SNMP's authentication is that community strings are sent in plain text, which makes it easy for people to intercept them and use them against you. SNMPv3 addresses this by allowing, among other things, secure authentication and communication between SNMP devices.

There are ways to reduce your risk of attack. IP firewalls or filters minimize the chance that someone can harm any managed device on your network by attacking it through SNMP. You can configure your firewall to allow UDP traffic from only a list of known hosts. For example, you can allow UDP traffic on port 161 (SNMP requests) into your network only if it comes from one of your network-management stations. The same goes for traps; you can configure your router so it allows UDP traffic on port 162 to your NMS only if it originates from one of the hosts you

are monitoring. Firewalls aren't 100% effective, but simple precautions such as these do a lot to reduce your risk.



It is important to realize that if someone has read-write access to any of your SNMP devices, he can gain control of those devices by using SNMP (for example, he can set router interfaces, switch ports down, or even modify your routing tables). One way to protect your community strings is to use a *Virtual Private Network* (VPN) to make sure your network traffic is encrypted. Another way is to change your community strings often. Changing community strings isn't difficult for a small network, but for a network that spans city blocks or more and has dozens (or hundreds or thousands) of managed hosts, changing community strings can be a problem. An easy solution is to write a simple Perl script that uses SNMP to change the community strings on your devices.

2.3 The Structure of Management Information

So far, we have used the term "management information" to refer to the operational parameters of SNMP-capable devices. However, we've said very little about what management information actually contains or how it is represented. The first step toward understanding what kind of information a device can provide is to understand how this data itself is represented within the context of SNMP. The *Structure of Management Information Version 1* (SMIv1, RFC 1155) does exactly that: it defines precisely how managed objects^[1] are named and specifies their associated datatypes. The *Structure of Management Information Version 2* (SMIv2, RFC 2578) provides enhancements for SNMPv2. We'll start by discussing SMIv1 and will discuss SMIv2 in the next section.

^[1] For the remainder of this book "management information" will be referred to as "managed objects." Similarly, a single piece of management information (such as the operational status of a router interface) will be known as a "managed object."

The definition of managed objects can be broken down into three attributes:

Name

The name, or *object identifier* (OID), uniquely defines a managed object. Names commonly appear in two forms: numeric and "human readable." In either case, the names are long and inconvenient. In SNMP applications, a lot of work goes into helping you navigate through the namespace conveniently.

Type and syntax

A managed object's datatype is defined using a subset of *Abstract Syntax Notation One* (ASN.1). ASN.1 is a way of specifying how data is represented and transmitted between managers and agents, within the context of SNMP. The nice thing about ASN.1 is that the notation is machine-independent. This means that a PC running Windows NT can communicate with a Sun SPARC machine and not have to worry about things such as byte ordering.

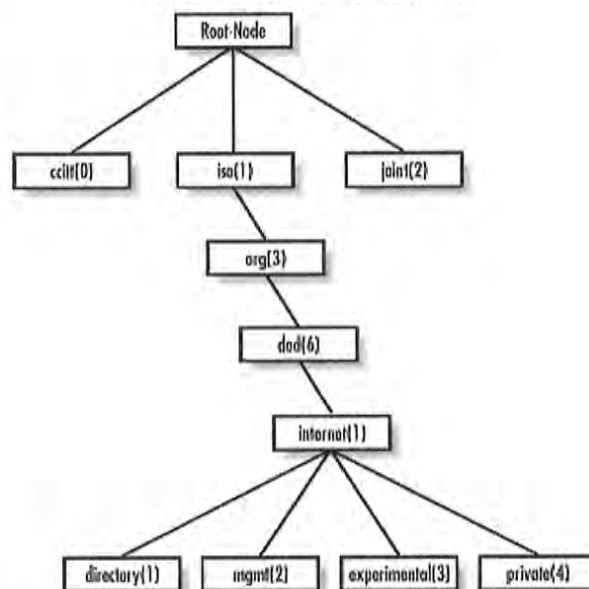
Encoding

A single instance of a managed object is encoded into a string of octets using the *Basic Encoding Rules* (BER). BER defines how the objects are encoded and decoded so they can be transmitted over a transport medium such as Ethernet.

2.3.1 Naming OIDs

Managed objects are organized into a tree-like hierarchy. This structure is the basis for SNMP's naming scheme. An object ID is made up of a series of integers based on the nodes in the tree, separated by dots (.). Although there's a human-readable form that's more friendly than a string of numbers, this form is nothing more than a series of names separated by dots, each of which represents a node of the tree. So you can use the numbers themselves, or you can use a sequence of names that represent the numbers. [Figure 2-2](#) shows the top few levels of this tree. (We have intentionally left out some branches of the tree that don't concern us here.)

Figure 2-2. SMI object tree



In the object tree, the node at the top of the tree is called the *root*, anything with children is called a *subtree*, and anything without children is called a *leaf node*. For example,

Figure 2-2's root, the starting point for the tree, is called "Root-Node." Its subtree is made up of ccitt(0), iso(1), and joint(2). In this illustration, iso(1) is the only node that contains a subtree; the other two nodes are both leaf nodes. ccitt(0) and joint(2) do not pertain to SNMP, so they will not be discussed in this book.^[2]

^[2] The ccitt subtree is administered by the International Telegraph and Telephone Consultative Committee (CCITT); the joint subtree is administered jointly by the International Organization for Standardization (ISO) and CCITT. As we said, neither branch has anything to do with SNMP.

For the remainder of this book we will focus on the iso(1).org(3).dod(6).internet(1) subtree,^[3] which is represented in OID form as 1.3.6.1 or as iso.org.dod.internet. Each managed object has a numerical OID and an associated textual name. The dotted-decimal notation is how a managed object is represented internally within an agent; the textual name, like an IP domain name, saves humans from having to remember long, tedious strings of integers.

^[3] Note that the term "branch" is sometimes used interchangeably with "subtree."

The directory branch currently is not used. The management branch, or mgmt, defines a standard set of Internet management objects. The experimental branch is reserved for testing and research purposes. Objects under the private branch are defined unilaterally, which means that individuals and organizations are responsible for defining the objects under this branch. Here is the definition of the internet subtree, as well as all four of its subtrees:

```
internet    OBJECT IDENTIFIER ::= { iso org(3) dod(6) 1 }
directory   OBJECT IDENTIFIER ::= { internet 1 }
mgmt        OBJECT IDENTIFIER ::= { internet 2 }
experimental OBJECT IDENTIFIER ::= { internet 3 }
private     OBJECT IDENTIFIER ::= { internet 4 }
```

The first line declares internet as the OID 1.3.6.1, which is defined as a subtree of iso.org.dod, or 1.3.6 (the ::= is a definition operator). The last four declarations are similar, but they define the other branches that belong to internet. For the directory branch, the notation { internet 1 } tells us that it is part of the internet subtree, and that its OID is 1.3.6.1.1. The OID for mgmt is 1.3.6.1.2, and so on.

There is currently one branch under the private subtree. It's used to give hardware and software vendors the ability to define their own private objects for any type of hardware or software they want managed by SNMP. Its SMI definition is:

```
enterprises OBJECT IDENTIFIER ::= { private 1 }
```

The Internet Assigned Numbers Authority (IANA) currently manages all the private enterprise number assignments for individuals, institutions, organizations, companies, etc.^[4] A list of all the current private enterprise numbers can be obtained from <ftp://ftp.isi.edu/in-notes/iana/assignments/enterprise-numbers>. As an example, Cisco Systems's private enterprise number is 9, so the base OID for its private object space is defined as iso.org.dod.internet.private.enterprises.cisco, or 1.3.6.1.4.1.9. Cisco is free to do as it wishes with this private branch. It's typical for companies such as Cisco that manufacture networking equipment to define their own private enterprise objects. This allows for a richer set of management information than can be gathered from the standard set of managed objects defined under the mgmt branch.

^[4] The term "private enterprise" will be used throughout this book to refer to the enterprises branch.

Companies aren't the only ones who can register their own private enterprise numbers. Anyone can do so, and it's free. The web-based form for registering private enterprise numbers can be found at <http://www.isi.edu/cgi-bin/iana/enterprise.pl>. After you fill in the form, which asks for information such as your organization's name and contact information, your request should be approved in about a week. Why would you want to register your own number? When you become more conversant in SNMP, you'll find things you want to monitor that aren't covered by any MIB, public or private. With your own enterprise number, you can create your own private MIB that allows you to monitor exactly what you want. You'll need to be somewhat clever in extending your agents so that they can look up the information you want, but it's very doable.

2.3.2 Defining OIDs

The SYNTAX attribute provides for definitions of managed objects through a subset of ASN.1. SMIV1 defines several datatypes that are paramount to the management of networks and network devices. It's important to keep in mind that these datatypes are simply a way to define what kind of information a managed object can hold. The types we'll be discussing are similar to those that you'd find in a computer programming language like C. [Table 2-1](#) lists the supported datatypes for SMIV1.

Table 2-1. SMIV1 Datatypes	
Datatype	Description
INTEGER	A 32-bit number often used to specify enumerated types within the context of a single managed object. For example, the operational status of a router interface can be up, down, or testing. With

	enumerated types, 1 would represent up, 2 down, and 3 testing. The value zero (0) must not be used as an enumerated type, according to RFC 1155.
OCTET STRING	A string of zero or more octets (more commonly known as bytes) generally used to represent text strings, but also sometimes used to represent physical addresses.
Counter	A 32-bit number with minimum value 0 and maximum value $2^{32} - 1$ (4,294,967,295). When the maximum value is reached, it wraps back to zero and starts over. It's primarily used to track information such as the number of octets sent and received on an interface or the number of errors and discards seen on an interface. A Counter is monotonically increasing, in that its values should never decrease during normal operation. When an agent is rebooted, all Counter values should be set to zero. Deltas are used to determine if anything useful can be said for successive queries of Counter values. A delta is computed by querying a Counter at least twice in a row, and taking the difference between the query results over some time interval.
OBJECT IDENTIFIER	A dotted-decimal string that represents a managed object within the object tree. For example, 1.3.6.1.4.1.9 represents Cisco Systems's private enterprise OID.
NULL	Not currently used in SNMP.
SEQUENCE	Defines lists that contain zero or more other ASN.1 datatypes.
SEQUENCE OF	Defines a managed object that is made up of a SEQUENCE of ASN.1 types.
IpAddress	Represents a 32-bit IPv4 address. Neither SMIV1 nor SMIV2 discusses 128-bit IPv6 addresses; this problem will be addressed by the IETF's SMI Next Generation (SMING) working group (see http://www.ietf.org/html.charters/sming-charter.html).
NetworkAddress	Same as the IpAddress type, but can represent different network address types.
Gauge	A 32-bit number with minimum value 0 and maximum value $2^{32} - 1$ (4,294,967,295). Unlike a Counter , a Gauge can increase and decrease at will, but it can never exceed its maximum value. The interface speed on a

	router is measured with a Gauge.
TimeTicks	A 32-bit number with minimum value 0 and maximum value $2^{32} - 1$ (4,294,967,295). TimeTicks measures time in hundredths of a second. Uptime on a device is measured using this datatype.
Opaque	Allows any other ASN.1 encoding to be stuffed into an OCTET STRING.

The goal of all these object types is to define managed objects. In [Chapter 1](#), we said that a MIB is a logical grouping of managed objects as they pertain to a specific management task, vendor, etc. The MIB can be thought of as a specification that defines the managed objects a vendor or device supports. Cisco, for instance, has literally hundreds of MIBs defined for its vast product line. For example, its Catalyst device has a separate MIB from its 7000 series router. Both devices have different characteristics that require different management capabilities. Vendor-specific MIBs typically are distributed as human-readable text files that can be inspected (or even modified) with a standard text editor such as vi.



Most modern NMS products maintain a compact form of all the MIBs that define the set of managed objects for all the different types of devices they're responsible for managing. NMS administrators will typically compile a vendor's MIB into a format the NMS can use. Once a MIB has been loaded or compiled, administrators can refer to managed objects using either the numeric or human-readable object ID.

It's important to know how to read and understand MIB files. The following example is a stripped-down version of MIB-II (anything preceded by -- is a comment):

```
RFC1213-MIB DEFINITIONS ::= BEGIN

    IMPORTS
        mgmt, NetworkAddress, IpAddress, Counter, Gauge,
        TimeTicks
            FROM RFC1155-SMI
    OBJECT-TYPE
        FROM RFC 1212;

    mib-2      OBJECT IDENTIFIER ::= { mgmt 1 }

    -- groups in MIB-II

    system     OBJECT IDENTIFIER ::= { mib-2 1 }
    interfaces OBJECT IDENTIFIER ::= { mib-2 2 }
    at         OBJECT IDENTIFIER ::= { mib-2 3 }
```

```

ip          OBJECT IDENTIFIER ::= { mib-2 4 }
icmp        OBJECT IDENTIFIER ::= { mib-2 5 }
tcp         OBJECT IDENTIFIER ::= { mib-2 6 }
udp         OBJECT IDENTIFIER ::= { mib-2 7 }
egp         OBJECT IDENTIFIER ::= { mib-2 8 }
transmission OBJECT IDENTIFIER ::= { mib-2 10 }
snmp        OBJECT IDENTIFIER ::= { mib-2 11 }

```

```
-- the Interfaces table
```

```

-- The Interfaces table contains information on the entity's
-- interfaces. Each interface is thought of as being
-- attached to a 'subnetwork.' Note that this term should
-- not be confused with 'subnet,' which refers to an
-- addressing-partitioning scheme used in the Internet
-- suite of protocols.

```

```
ifTable OBJECT-TYPE
```

```
    SYNTAX SEQUENCE OF IfEntry
```

```
    ACCESS not-accessible
```

```
    STATUS mandatory
```

```
    DESCRIPTION
```

```
        "A list of interface entries. The number of entries is
        given by the value of ifNumber."
```

```
    ::= { interfaces 2 }
```

```
ifEntry OBJECT-TYPE
```

```
    SYNTAX IfEntry
```

```
    ACCESS not-accessible
```

```
    STATUS mandatory
```

```
    DESCRIPTION
```

```
        "An interface entry containing objects at the subnetwork
        layer and below for a particular interface."
```

```
    INDEX { ifIndex }
```

```
    ::= { ifTable 1 }
```

```
IfEntry ::=
```

```
    SEQUENCE {
```

```
        ifIndex
```

```
        INTEGER,
```

```
        ifDescr
```

```
        DisplayString,
```

```
        ifType
```

```
        INTEGER,
```

```
        ifMtu
```

```
        INTEGER,
```

```
        ifSpeed
```

```
        Gauge,
```

```
        ifPhysAddress
```

```
        PhysAddress,
```

```
        ifAdminStatus
```

```
        INTEGER,
```

```
        ifOperStatus
```

```
        INTEGER,
```

```

        ifLastChange
            TimeTicks,
        ifInOctets
            Counter,
        ifInUcastPkts
            Counter,
        ifInNUcastPkts
            Counter,
        ifInDiscards
            Counter,
        ifInErrors
            Counter,
        ifInUnknownProtos
            Counter,
        ifOutOctets
            Counter,
        ifOutUcastPkts
            Counter,
        ifOutNUcastPkts
            Counter,
        ifOutDiscards
            Counter,
        ifOutErrors
            Counter,
        ifOutQLen
            Gauge,
        ifSpecific
            OBJECT IDENTIFIER
    }

ifIndex OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "A unique value for each interface. Its value ranges
        between 1 and the value of ifNumber. The value for each
        each interface must remain constant at least from one
        reinitialization of the entity's network-management
        system to the next reinitialization."

 ::= { ifEntry 1 }

ifDescr OBJECT-TYPE
    SYNTAX  DisplayString (SIZE (0..255))
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "A textual string containing information about the
        interface. This string should include the name of
        the manufacturer, the product name, and the version
        of the hardware interface."
 ::= { ifEntry 2 }

```

END

The first line of this file defines the name of the MIB, in this case `RFC1213-MIB`. (RFC 1213 is the RFC that defines MIB-II; many of the MIBs we refer to are defined by RFCs). The format of this definition is always the same. The `IMPORTS` section of the MIB is sometimes referred to as the linkage section. It allows you to import datatypes and OIDs from other MIB files using the `IMPORTS` clause. This MIB imports the following items from `RFC1155-SMI` (RFC 1155 defines `SMIv1`, which we discussed earlier in this chapter):

- `mgmt`
- `NetworkAddress`
- `IpAddress`
- `Counter`
- `Gauge`
- `TimeTicks`

It also imports `OBJECT-TYPE` from RFC 1212, the Concise MIB Definition, which defines how MIB files are written. Each group of items imported using the `IMPORTS` clause uses a `FROM` clause to define the MIB file from which the objects are taken.

The OIDs that will be used throughout the remainder of the MIB follow the linkage section. This group of lines sets up the top level of the `mib-2` subtree. `mib-2` is defined as `mgmt` followed by `.1`. We saw earlier that `mgmt` was equivalent to `1.3.6.1.2`. Therefore, `mib-2` is equivalent to `1.3.6.1.2.1`. Likewise, the interfaces group under `mib-2` is defined as `{ mib-2 2 }`, or `1.3.6.1.2.1.2`.

After the OIDs are defined, we get to the actual object definitions. Every object definition has the following format:

```
<name> OBJECT-TYPE
    SYNTAX <datatype>
    ACCESS <either read-only, read-write, write-only, or not-accessible>
    STATUS <either mandatory, optional, or obsolete>
    DESCRIPTION
        "Textual description describing this particular managed object."
    ::= { <Unique OID that defines this object> }
```

The first managed object in our subset of the MIB-II definition is `ifTable`, which represents a table of network interfaces on a managed device (note that object names are defined using mixed case, with the first letter in lowercase). Here is its definition using ASN.1 notation:

```
ifTable OBJECT-TYPE
    SYNTAX SEQUENCE OF IfEntry
    ACCESS not-accessible
    STATUS mandatory
```

DESCRIPTION

"A list of interface entries. The number of entries is given by the value of ifNumber."

::= { interfaces 2 }

The SYNTAX of ifTable is SEQUENCE OF IfEntry. This means that ifTable is a table containing the columns defined in IfEntry. The object is not-accessible, which means that there is no way to query an agent for this object's value. Its status is mandatory, which means an agent must implement this object in order to comply with the MIB-II specification. The DESCRIPTION describes what exactly this object is. The unique OID is 1.3.6.1.2.1.2.2, or iso.org.dod.internet.mgmt.interfaces.2.

Let's now look at the SEQUENCE definition from the MIB file earlier in this section, which is used with the SEQUENCE OF type in the ifTable definition:

```
IfEntry ::=
  SEQUENCE {
    ifIndex
      INTEGER,
    ifDescr
      DisplayString,
    ifType
      INTEGER,
    ifMtu
      INTEGER,
    .
    .
    .
    ifSpecific
      OBJECT IDENTIFIER
  }
```

Note that the name of the sequence (IfEntry) is mixed-case, but the first letter is capitalized, unlike the object definition for ifTable. This is how a sequence name is defined. A sequence is simply a list of columnar objects and their SMI datatypes, which defines a conceptual table. In this case, we expect to find variables defined by ifIndex, ifDescr, ifType, etc. This table can contain any number of rows; it's up to the agent to manage the rows that reside in the table. It is possible for an NMS to add rows to a table. This operation is covered later, in [Section 2.6.4](#).

Now that we have IfEntry to specify what we'll find in any row of the table, we can look back to the definition of ifEntry (the actual rows of the table) itself:

```
ifEntry OBJECT-TYPE
  SYNTAX  IfEntry
  ACCESS  not-accessible
  STATUS  mandatory
  DESCRIPTION
    "An interface entry containing objects at the subnetwork layer
```

```

        and below for a particular interface."
INDEX    { ifIndex }
::= { ifTable 1 }

```

ifEntry defines a particular row in the ifTable. Its definition is almost identical to that of ifTable, except we have introduced a new clause, INDEX. The index is a unique key used to define a single row in the ifTable. It's up to the agent to make sure the index is unique within the context of the table. If a router has six interfaces, ifTable will have six rows in it. ifEntry's OID is 1.3.6.1.2.1.2.2.1, or iso.org.dod.internet.mgmt.interfaces.ifTable.ifEntry. The index for ifEntry is ifIndex, which is defined as:

```

ifIndex OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "A unique value for each interface. Its value ranges between
        1 and the value of ifNumber. The value for each interface
        must remain constant at least from one reinitialization of the
        entity's network-management system to the next reinitialization."
    ::= { ifEntry 1 }

```

The ifIndex object is read-only, which means we can see its value, but we cannot change it. The final object our MIB defines is ifDescr, which is a textual description for the interface represented by that particular row in the ifTable. Our MIB example ends with the END clause, which marks the end of the MIB. In the actual MIB-II files, each object listed in the IfEntry sequence has its own object definition. In this version of the MIB we list only two of them, in the interest of conserving space.

2.4 Extensions to the SMI in Version 2

SMIv2 extends the SMI object tree by adding the snmpV2 branch to the internet subtree, adding several new datatypes, and making a number of other changes. [Figure 2-3](#) shows how the snmpV2 objects fit into the bigger picture; the OID for this new branch is 1.3.6.1.6.3.1.1, or iso.org.dod.internet.snmpV2.snmpModules.snmpMIB.snmpMIBObjects. SMIv2 also defines some new datatypes, which are summarized in Table 2-2.

Figure 2-3. SMIv2 registration tree for SNMPv2

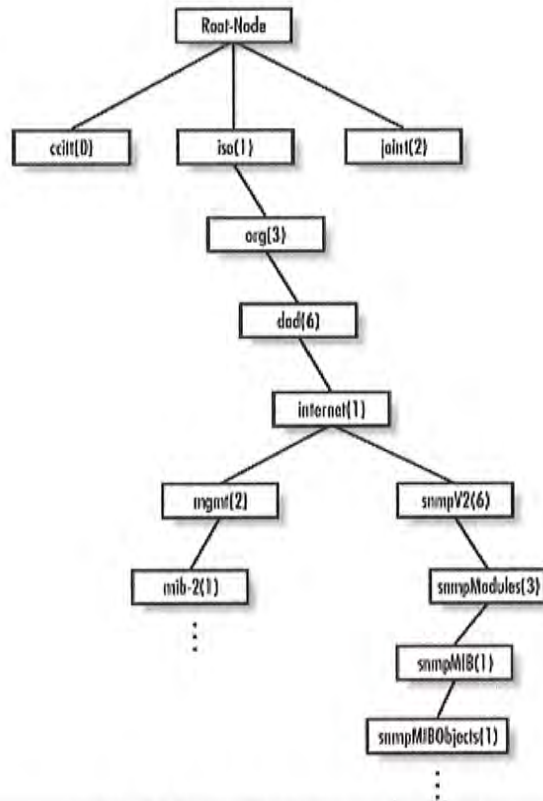


Table 2-2. New Datatypes for SMIV2

Datatype	Description
Integer32	Same as an INTEGER.
Counter32	Same as a Counter.
Gauge32	Same as a Gauge.
Unsigned32	Represents decimal values in the range of 0 to $2^{32} - 1$ inclusive.
Counter64	Similar to Counter32, but its maximum value is 18,446,744,073,709,551,615. Counter64 is ideal for situations in which a Counter32 may wrap back to 0 in a short amount of time.
BITS	An enumeration of nonnegative named bits.

The definition of an object in SMIV2 has changed slightly from SMIV1. There are some new optional fields, giving you more control over how an object is accessed, allowing you to augment a table by adding more columns, and letting you give better descriptions. Here's the syntax of an object definition for SMIV2. The changed parts are in bold:

```

<name> OBJECT-TYPE
    SYNTAX <datatype>
  
```

UnitsParts <Optional, see below>
MAX-ACCESS <See below>
STATUS <See below>
DESCRIPTION
"Textual description describing this particular managed object."
AUGMENTS { <name of table> }
::= { <Unique OID that defines this object> }

Table 2-3 briefly describes the object definition enhancements made in SMIV2.

Table 2-3. SMIV2 Object Definition Enhancements	
Object Definition Enhancement	Description
UnitsParts	A textual description of the units (i.e., seconds, milliseconds, etc.) used to represent the object.
MAX-ACCESS	An OBJECT-TYPE's ACCESS can be MAX-ACCESS in SNMPv2. The valid options for MAX-ACCESS are read-only, read-write, read-create, not-accessible, and accessible-for-notify.
STATUS	This clause has been extended to allow the current, obsolete, and deprecated keywords. current in SNMPv2 is the same as mandatory in an SNMPv1 MIB.
AUGMENTS	In some cases it is useful to add a column to an existing table. The AUGMENTS clause allows you to extend a table by adding one or more columns, represented by some other object. This clause requires the name of the table the object will augment.

SMIV2 defines a new trap type called NOTIFICATION-TYPE, which we will discuss later in [Section 2.6.7](#). SMIV2 also introduces new textual conventions that allow managed objects to be created in more abstract ways. RFC 2579 defines the textual conventions used by SNMPv2, which are listed in [Table 2-4](#).

Table 2-4. Textual Conventions for SMIV2	
Textual Convention	Description
DisplayString	A string of NVT ASCII characters. A DisplayString can be no more than 255 characters in length.
PhysAddress	A media- or physical-level address, represented as an OCTET STRING.
MacAddress	Defines the media-access address for IEEE 802 (the

	standard for local area networks) in canonical ^[5] order. (In everyday language, this means the Ethernet address.) This address is represented as six octets.
TruthValue	Defines both true and false Boolean values.
TestAndIncr	Used to keep two management stations from modifying the same managed object at the same time.
AutonomousType	An OID used to define a subtree with additional MIB-related definitions.
VariablePointer	A pointer to a particular object instance, such as the ifDescr for interface 3. In this case, the VariablePointer would be the OID ifDescr.3.
RowPointer	A pointer to a row in a table. For example, ifIndex.3 points to the third row in the ifTable.
RowStatus	Used to manage the creation and deletion of rows in a table, since SNMP has no way of doing this via the protocol itself. RowStatus can keep track of the state of a row in a table, as well as receive commands for creation and deletion of rows. This textual convention is designed to promote table integrity when more than one manager is updating rows. The following enumerated types define the commands and state variables: active(1), notInService(2), notReady(3), createAndGo(4), createAndWait(5), and destroy(6).
TimeStamp	Measures the amount of time elapsed between the device's system uptime and some event or occurrence.
TimeInterval	Measures a period of time in hundredths of a second. TimeInterval can take any integer value from 0-2147483647.
DateAndTime	An OCTET STRING used to represent date-and-time information.
StorageType	Defines the type of memory an agent uses. The possible values are other(1), volatile(2), nonVolatile(3), permanent(4), and readOnly(5).
TDomain	Denotes a kind of transport service.
TAddress	Denotes the transport service address. TAddress is defined to be from 1-255 octets in length.

^[5] Canonical order means that the address should be represented with the least-significant bit first.

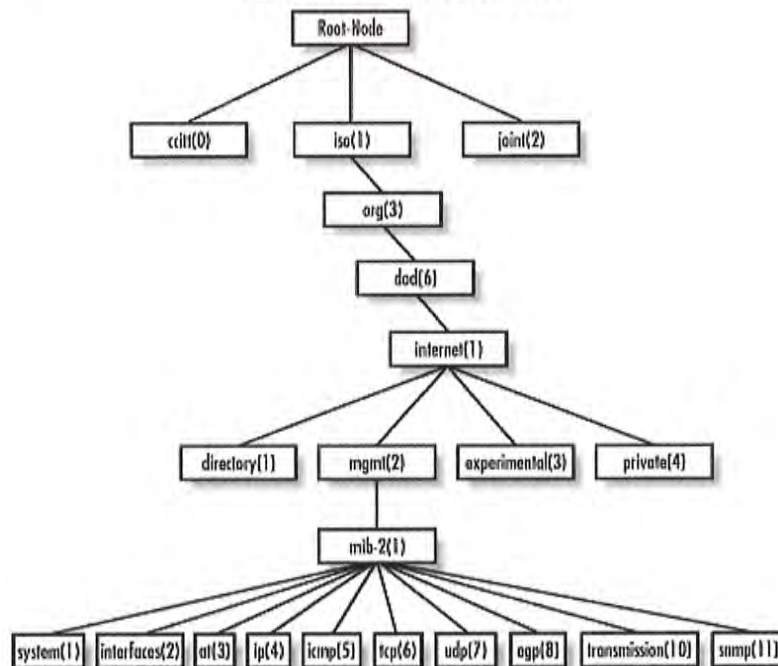
2.5 A Closer Look at MIB-II

MIB-II is a very important management group, because every device that supports SNMP must also support MIB-II. Therefore, we will use objects from MIB-II in our examples throughout this book. We won't go into detail about every object in the MIB; we'll simply define the subtrees. The section of RFC1213-MIB that defines the base OIDs for the mib-2 subtree looks like this:

```
mib-2    OBJECT IDENTIFIER ::= { mgmt 1 }
system   OBJECT IDENTIFIER ::= { mib-2 1 }
interfaces OBJECT IDENTIFIER ::= { mib-2 2 }
at       OBJECT IDENTIFIER ::= { mib-2 3 }
ip       OBJECT IDENTIFIER ::= { mib-2 4 }
icmp     OBJECT IDENTIFIER ::= { mib-2 5 }
tcp      OBJECT IDENTIFIER ::= { mib-2 6 }
udp      OBJECT IDENTIFIER ::= { mib-2 7 }
egp      OBJECT IDENTIFIER ::= { mib-2 8 }
transmission OBJECT IDENTIFIER ::= { mib-2 10 }
snmp     OBJECT IDENTIFIER ::= { mib-2 11 }
```

mib-2 is defined as iso.org.dod.internet.mgmt.1, or 1.3.6.1.2.1. From here, we can see that the system group is mib-2 1, or 1.3.6.1.2.1.1, and so on. [Figure 2-4](#) shows the MIB-II subtree of the mgmt branch.

Figure 2-4. MIB-II subtree



[Table 2-5](#) briefly describes each of the management groups defined in MIB-II. We don't go into great detail about each

group, since you can pull down RFC 1213 and read the MIB yourself.

Table 2-5. Brief Description of the MIB-II Groups		
Subtree Name	OID	Description
system	1.3.6.1.2.1.1	Defines a list of objects that pertain to system operation, such as the system uptime, system contact, and system name.
interfaces	1.3.6.1.2.1.2	Keeps track of the status of each interface on a managed entity. The interfaces group monitors which interfaces are up or down and tracks such things as octets sent and received, errors and discards, etc.
at	1.3.6.1.2.1.3	The address translation (at) group is deprecated and is provided only for backward compatibility. It will probably be dropped from MIB-III.
ip	1.3.6.1.2.1.4	Keeps track of many aspects of IP, including IP routing.
icmp	1.3.6.1.2.1.5	Tracks things such as ICMP errors, discards, etc.
tcp	1.3.6.1.2.1.6	Tracks, among other things, the state of the TCP connection (e.g., closed, listen, synSent, etc.).
udp	1.3.6.1.2.1.7	Tracks UDP statistics, datagrams in and out, etc.
egp	1.3.6.1.2.1.8	Tracks various statistics about EGP and keeps an EGP neighbor table.
transmission	1.3.6.1.2.1.10	There are currently no objects defined for this group, but other media-specific MIBs are defined using this subtree.
snmp	1.3.6.1.2.1.11	Measures the performance of the underlying SNMP implementation on the managed entity and tracks things such as the number of SNMP packets sent and received.

2.6 SNMP Operations

We've discussed how SNMP organizes information, but we've left out how we actually go about gathering management information. Now, we're going to take a look under the hood to see how SNMP does its thing.

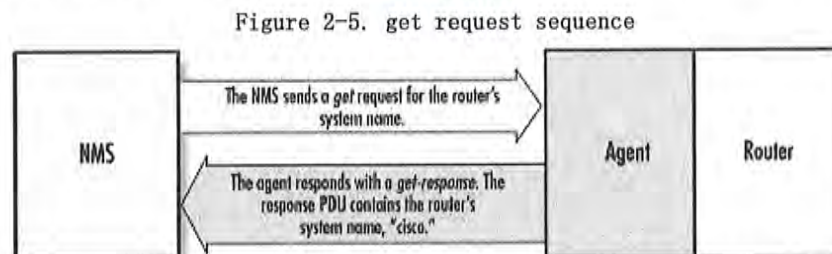
The *Protocol Data Unit* (PDU) is the message format that managers and agents use to send and receive information. There is a standard PDU format for each of the following SNMP operations:

- *get*
- *get-next*
- *get-bulk* (SNMPv2 and SNMPv3)
- *set*
- *get-response*
- *trap*
- *notification* (SNMPv2 and SNMPv3)
- *inform* (SNMPv2 and SNMPv3)
- *report* (SNMPv2 and SNMPv3)

Let's take a look at each of these operations.

2.6.1 The get Operation

The get request is initiated by the NMS, which sends the request to the agent. The agent receives the request and processes it to best of its ability. Some devices that are under heavy load, such as routers, may not be able to respond to the request and will have to drop it. If the agent is successful in gathering the requested information, it sends a *get-response* back to the NMS, where it is processed. This process is illustrated in [Figure 2-5](#).



How did the agent know what the NMS was looking for? One of the items in the *get* request is a *variable binding*. A variable binding, or varbind, is a list of MIB objects that allows a request's recipient to see what the originator wants to know. Variable bindings can be thought of as OID=value pairs that

make it easy for the originator (the NMS, in this case) to pick out the information it needs when the recipient fills the request and sends back a response. Let's look at this operation in action:

```
$ snmpget cisco.ora.com public .1.3.6.1.2.1.1.6.0
system.sysLocation.0 = ""
```



All the Unix commands presented in this chapter come from the Net-SNMP agent package (formerly the UCD-SNMP project), a freely available Unix and Windows NT agent. [Chapter 5](#) provides a URL from which you can download the package. The commands in this package are summarized in [Appendix C](#).

Several things are going on in this example. First, we're running a command on a Unix host. The command is called *snmpget*. Its main job is to facilitate the gathering of management data using a get request. We've given it three arguments on the command line: the name of the device we would like to query (cisco.ora.com), the read-only community string (public), and the OID we would like gathered (.1.3.6.1.2.1.1.6.0). If we look back at [Table 2-5](#) we see that 1.3.6.1.2.1.1 is the system group, but there are two more integers at the end of the OID: .6 and .0. The .6 is actually the MIB variable that we wish to query; its human-readable name is sysLocation. In this case, we would like to see what the system location is set to on the Cisco router. As you can see by the response (system.sysLocation.0 = ""), the system location on this router currently is not set to anything. Also note that the response from *snmpget* is in variable binding format, OID=value.

There is one more thing to look at. Why does the MIB variable have a .0 tacked on the end? In SNMP, MIB objects are defined by the convention x.y, where x is the actual OID of the managed object (in our example, 1.3.6.1.2.1.1.6) and y is the instance identifier. For scalar objects (that is, objects that aren't defined as a row in a table) y is always 0. In the case of a table, the instance identifier lets you select a specific row of the table; 1 is the first row, 2 is the second row, etc. For example, consider the ifTable object we looked at earlier in this chapter. When looking up values in the ifTable, we would use a nonzero instance identifier to select a particular row in the table (in this case, a particular network interface).



Graphical NMS applications, which include most commercial packages, do not use command-line programs to retrieve management information. We use these commands to give you a feel for how the retrieval commands work and what they

typically return. The information a graphical NMS retrieves and its retrieval process are identical to these command-line programs; the NMS just lets you formulate queries and displays the results using a more convenient GUI.

The *getcommand* is useful for retrieving a single MIB object at a time. Trying to manage anything in this manner can be a waste of time, though. This is where the *get-next* command comes in. It allows you to retrieve more than one object from a device, over a period of time.

2.6.2 The *get-next* Operation

The *get-next* operation lets you issue a sequence of commands to retrieve a group of values from a MIB. In other words, for each MIB object we want to retrieve, a separate *get-next* request and *get-response* are generated. The *get-next* command traverses a subtree in lexicographic order. Since an OID is a sequence of integers, it's easy for an agent to start at the root of its SMI object tree and work its way down until it finds the OID it is looking for. When the NMS receives a response from the agent for the *get-next* command it just issued, it issues another *get-next* command. It keeps doing this until the agent returns an error, signifying that the end of the MIB has been reached and there are no more objects left to get.

If we look at another example, we can see this behavior in action. This time we'll use a command called *snmpwalk*. This command simply facilitates the *get-next* procedure for us. It's invoked just like the *snmpget* command, except this time we specify which branch to start at (in this case, the system group):

```
$snmpwalk cisco.ora.com public system
system.sysDescr.0 = "Cisco Internetwork Operating System Software
..IOS (tm) 2500 Software (C2500-I-L), Version 11.2(5), RELEASE
SOFTWARE (fc1)..Copyright (c) 1986-1997 by cisco Systems, Inc...
Compiled Mon 31-Mar-97 19:53 by ckralik"
system.sysObjectID.0 = OID: enterprises.9.1.19
system.sysUpTime.0 = Timeticks: (27210723) 3 days, 3:35:07.23
system.sysContact.0 = ""
system.sysName.0 = "cisco.ora.com"
system.sysLocation.0 = ""
system.sysServices.0 = 6
```

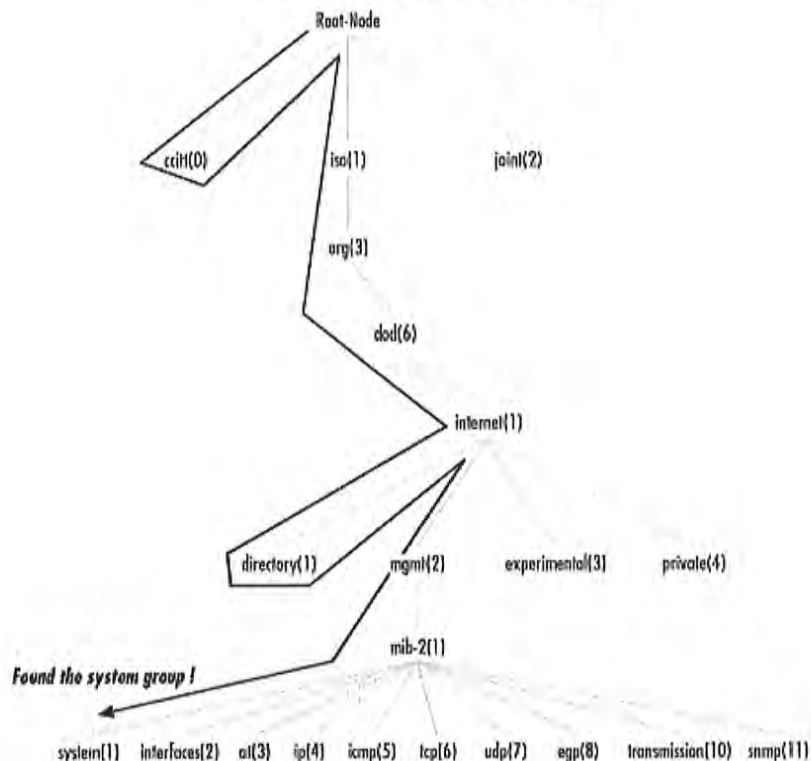
The *get-next* sequence returns seven MIB variables. Each of these objects is part of the system group as it's defined in RFC 1213. We see a system object ID, the amount of time the system has been up, the contact person, etc.

Given that you've just looked up some object, how does *get-next* figure out which object to look up next? *get-next* is based on the concept of the lexicographic ordering of the MIB's object

tree. This order is made much simpler because every node in the tree is assigned a number. To understand what this means, let's start at the root of the tree and walk down to the system node.

To get to the system group (OID 1.3.6.1.2.1.1), we start at the root of the object tree and work our way down. [Figure 2-6](#) shows the logical progression from the root of the tree all the way to the system group. At each node in the tree, we visit the lowest-numbered branch. Thus, when we're at the root node, we start by visiting ccitt. This node has no nodes underneath it, so we move to the iso node. Since iso does have a child we move to that node, org. The process continues until we reach the system node. Since each branch is made up of ascending integers (ccitt(0) iso(1) join(2), for example), the agent has no problem traversing this tree structure all the way down to the system(1) group. If we were to continue this walk, we'd proceed to system.1 (system.sysLocation), system.2, and the other objects in the system group. Next, we'd go to interfaces(2), and so on.

Figure 2-6. Walking the MIB tree

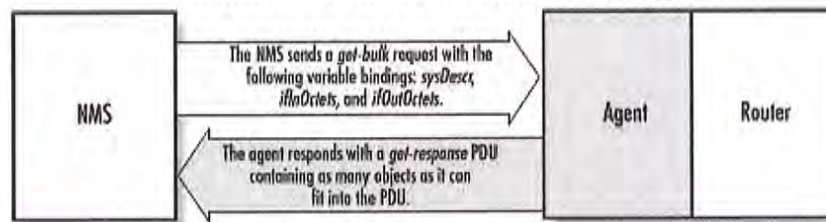


2.6.3 The get-bulk Operation

SNMPv2 defines the *get-bulk* operation, which allows a management application to retrieve a large section of a table at once. The standard *get* operation can attempt to retrieve more than one MIB object at once, but message sizes are limited by the agent's capabilities. If the agent can't return all the requested responses, it returns an error message with no data.

The *get-bulk* operation, on the other hand, tells the agent to send as much of the response back as it can. This means that incomplete responses are possible. Two fields must be set when issuing a *get-bulk* command: *nonrepeaters* and *max-repetitions*. *Nonrepeaters* tells the *get-bulk* command that the first N objects can be retrieved with a simple *get-next* operation. *Max-repetitions* tells the *get-bulk* command to attempt up to M *get-next* operations to retrieve the remaining objects. [Figure 2-7](#) shows the *get-bulk* command sequence.

Figure 2-7. *get-bulk* request sequence



In [Figure 2-7](#), we're requesting three bindings: *sysDescr*, *ifInOctets*, and *ifOutOctets*. The total number of variable bindings that we've requested is given by the formula $N + (M * R)$, where N is the number of *nonrepeaters* (i.e., scalar objects in the request -- in this case 1, because *sysDescr* is the only scalar object), M is *max-repetitions* (in this case, we've set it arbitrarily to 3), and R is the number of nonscalar objects in the request (in this case 2, because *ifInOctets* and *ifOutOctets* are both nonscalar). Plugging in the numbers from this example, we get $1 + (3 * 2) = 7$, which is the total number of variable bindings that can be returned by this *get-bulk* request.

The Net-SNMP package comes with a command for issuing *get-bulk* queries. If we execute this command using all the parameters previously discussed, it will look like the following:

```

$ snmpbulkget -v2c -B 1 3 linux.ora.com public sysDescr ifInOctets ifOutOctets
system.sysDescr.0 = "Linux linux 2.2.5-15 #3 Thu May 27 19:33:18 EDT 1999 i686"
interfaces.ifTable.ifEntry.ifInOctets.1 = 70840
interfaces.ifTable.ifEntry.ifOutOctets.1 = 70840
interfaces.ifTable.ifEntry.ifInOctets.2 = 143548020
interfaces.ifTable.ifEntry.ifOutOctets.2 = 111725152
interfaces.ifTable.ifEntry.ifInOctets.3 = 0
interfaces.ifTable.ifEntry.ifOutOctets.3 = 0
  
```

Since *get-bulk* is an SNMPv2 command, you have to tell *snmpgetbulk* to use an SNMPv2 PDU with the *-v2c* option. The *nonrepeaters* and *max-repetitions* are set with the *-B 1 3* option. This sets *nonrepeaters* to 1 and *max-repetitions* to 3. Notice that the command returned seven variable bindings: one for *sysDescr* and three each for *ifInOctets* and *ifOutOctets*.

2.6.4 The set Operation

The set command is used to change the value of a managed object or to create a new row in a table. Objects that are defined in the MIB as read-write or write-only can be altered or created using this command. It is possible for an NMS to set more than one object at a time.

Figure 2-8. set request sequence

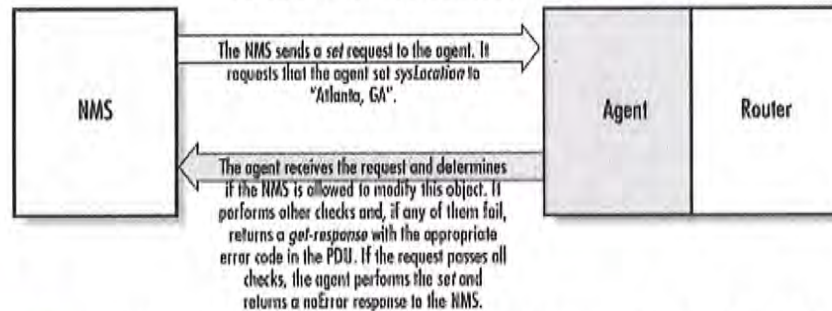


Figure 2-8 shows the set request sequence. It's similar to the other commands we've seen so far, but it is actually changing something in the device's configuration, as opposed to just retrieving a response to a query. If we look at an example of an actual set, you will see the command take place. The following example queries the sysLocation variable, then sets it to a value:

```

$ snmpget cisco.ora.com public system.sysLocation.0
system.sysLocation.0 = ""
$ snmpset cisco.ora.com private system.sysLocation.0 s "Atlanta, GA"
system.sysLocation.0 = "Atlanta, GA"
$ snmpget cisco.ora.com public system.sysLocation.0
system.sysLocation.0 = "Atlanta, GA"
  
```

The first command is the familiar *get* command, which displays the current value of *sysLocation*. In one of the previous examples we saw that it was undefined; this is still the case. The second command is *snmpset*. For this command, we supply the hostname, the read-write community string (*private*), and the variable we want to set (*system.sysLocation.0*), together with its new value (*s "Atlanta, GA"*). The *s* tells *snmpset* that we want to set the value of *sysLocation* to a string; and "Atlanta, GA" is the new value itself. How do we know that *sysLocation* requires a string value? The definition of *sysLocation* in RFC 1213 looks like this:

```

sysLocation OBJECT-TYPE
    SYNTAX  DisplayString (SIZE (0..255))
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
        "The physical location of this node (e.g., 'telephone closet,
        3rd floor')."
    ::= { system 6 }
  
```

The SYNTAX for *sysLocation* is *DisplayString (SIZE (0..255))*, which means that it's a string with a maximum length of 255 characters. The

snmpset command succeeds and reports the new value of *sysLocation*. But just to confirm, we run a final *snmpget*, which tells us that the *set* actually took effect. It is possible to set more than one object at a time, but if any of the sets fail, they all fail (i.e., no values are changed). This behavior is intended.

2.6.5 *get*, *get-next*, *get-bulk*, and *set* Error Responses

Error responses help you determine whether your *get* or *set* request was processed correctly by the agent. The *get*, *get-next*, and *set* operations can return the error responses shown in [Table 2-6](#). The error status for each error is shown in parentheses.

Table 2-6. SNMPv1 Error Messages	
SNMPv1 Error Message	Description
<i>noError</i> (0)	There was no problem performing the request.
<i>tooBig</i> (1)	The response to your request was too big to fit into one response.
<i>noSuchName</i> (2)	An agent was asked to get or set an OID that it can't find; i.e., the OID doesn't exist.
<i>badValue</i> (3)	A read-write or write-only object was set to an inconsistent value.
<i>readOnly</i> (4)	This error is generally not used. The <i>noSuchName</i> error is equivalent to this one.
<i>genErr</i> (5)	This is a catch-all error. If an error occurs for which none of the previous messages is appropriate, a <i>genError</i> is issued.

The SNMPv1 error messages are not very robust. In an attempt to fix this problem, SNMPv2 defines additional error responses that are valid for *get*, *set*, *get-next*, and *get-bulk* operations, provided that both the agent and NMS support SNMPv2. These responses are listed in [Table 2-7](#).

Table 2-7. SNMPv2 Error Messages	
SNMPv2 Error Message	Description
<i>noAccess</i> (6)	A <i>set</i> to an inaccessible variable was attempted. This typically occurs when the variable has an <i>ACCESS</i> type of <i>not-accessible</i> .

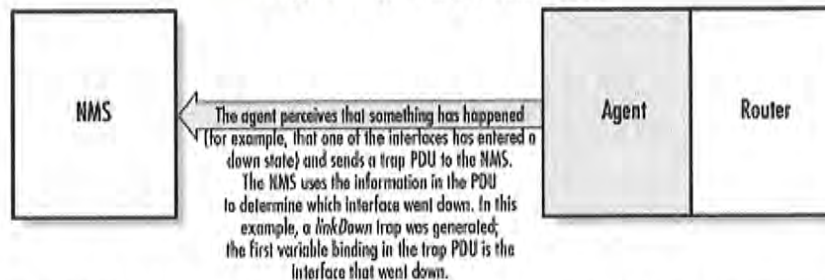
wrongType(7)	An object was set to a type that is different from its definition. This error will occur if you try to set an object that is of type <code>INTEGER</code> to a string, for example.
wrongLength(8)	An object's value was set to something other than what it calls for. For instance, a string can be defined to have a maximum character size. This error occurs if you try to set a string object to a value that exceeds its maximum length.
wrongEncoding(9)	A <i>set</i> operation was attempted using the wrong encoding for the object being set.
wrongValue(10)	A variable was set to a value it doesn't understand. This can occur when a read-write is defined as an enumeration, and you try to set it to a value that is not one of the enumerated types.
noCreation(11)	You tried to set a nonexistent variable or create a variable that doesn't exist in the MIB.
inconsistentValue	A MIB variable is in an inconsistent state, and is not accepting any <i>set</i> requests.
resourceUnavailable(13)	No system resources are available to perform a <i>set</i> .
commitFailed(14)	This is a catch-all error for <i>set</i> failures.
undoFailed(15)	A <i>set</i> failed and the agent was unable to roll back all the previous <i>sets</i> up until the point of failure.
authorizationError(16)	An SNMP command could not be authenticated; in other words, someone has supplied an incorrect community string.
notWritable(17)	A variable will not accept a <i>set</i> , even though it is supposed to.
inconsistentName(18)	You attempted to set a variable, but that attempt failed because the variable was in some kind of inconsistent state.

2.6.6 SNMP Traps

A trap is a way for an agent to tell the NMS that something bad has happened. In the [Section 1.3](#) of [Chapter 1](#) we explored the

notion of traps at a general level; now we'll look at them in a bit more detail. [Figure 2-9](#) shows the trap-generation sequence.

Figure 2-9. Trap generation



The trap originates from the agent and is sent to the trap destination, as configured within the agent itself. The trap destination is typically the IP address of the NMS. No acknowledgment is sent from the NMS to the agent, so the agent has no way of knowing if the trap makes it to the NMS. Since SNMP uses UDP, and since traps are designed to report problems with your network, traps are especially prone to getting lost and not making it to their destinations. However, the fact that traps can get lost doesn't make them any less useful; in a well-planned environment, they are an integral part of network management. It's better for your equipment to try to tell you that something is wrong, even if the message may never reach you, than simply to give up and let you guess what happened. Here are a few situations that a trap might report:

- A network interface on the device (where the agent is running) has gone down.
- A network interface on the device (where the agent is running) has come back up.
- An incoming call to a modem rack was unable to establish a connection to a modem.
- The fan on a switch or router has failed.

When an NMS receives a trap, it needs to know how to interpret it; that is, it needs to know what the trap means and how to interpret the information it carries. A trap is first identified by its generic trap number. There are seven generic trap numbers (0-6), shown in [Table 2-8](#). Generic trap 6 is a special catch-all category for "enterprise-specific" traps, which are traps defined by vendors or users that fall outside of the six generic trap categories. Enterprise-specific traps are further identified by an enterprise ID (i.e., an object ID somewhere in the enterprises branch of the MIB tree, `iso.org.dod.internet.private.enterprises`) and a specific trap number chosen by the enterprise that defined the trap. Thus, the object ID of an enterprise-specific trap is `enterprise-id.specific-trap-number`. For example, when Cisco defines special traps for its private MIBs, it places them all in its

enterprise-specific MIB tree (iso.org.dod.internet.private.enterprises.cisco). As we'll see in [Chapter 10](#), you are free to define your own enterprise-specific traps; the only requirement is that you register your own enterprise number with IANA.

A trap is usually packed with information. As you'd expect, this information is in the form of MIB objects and their values; as mentioned earlier, these object-value pairs are known as variable bindings. For the generic traps 0 through 5, knowledge of what the trap contains is generally built into the NMS software or trap receiver. The variable bindings contained by an enterprise-specific trap are determined by whomever defined the trap. For example, if a modem in a modem rack fails, the rack's agent may send a trap to the NMS informing it of the failure. The trap will most likely be an enterprise-specific trap defined by the rack's manufacturer; the trap's contents are up to the manufacturer, but it will probably contain enough information to let you determine exactly what failed (for example, the position of the modem card in the rack and the channel on the modem card).

Table 2-8. Generic Traps

Generic Trap Name and Number	Definition
coldStart (0)	Indicates that the agent has rebooted. All management variables will be reset; specifically, Counters and Gauges will be reset to zero (0). One nice thing about the coldStart trap is that it can be used to determine when new hardware is added to the network. When a device is powered on, it sends this trap to its trap destination. If the trap destination is set correctly (i.e., to the IP address of your NMS) the NMS can receive the trap and determine whether it needs to manage the device.
warmStart (1)	Indicates that the agent has reinitialized itself. None of the management variables will be reset.
linkDown (2)	Sent when an interface on a device goes down. The first variable binding identifies which interface went down.
linkUp (3)	Sent when an interface on a device comes back up. The first variable binding identifies which interface came back up.

authenticationFailure (4)	Indicates that someone has tried to query your agent with an incorrect community string; useful in determining if someone is trying to gain unauthorized access to one of your devices.
egpNeighborLoss (5)	Indicates that an <i>Exterior Gateway Protocol</i> (EGP) neighbor has gone down.
enterpriseSpecific (6)	Indicates that the trap is enterprise-specific. SNMP vendors and users define their own traps under the private-enterprise branch of the SMI object tree. To process this trap properly, the NMS has to decode the specific trap number that is part of the SNMP message.

In [Chapter 1](#) we mentioned that RFC 1697 is the RDBMS MIB. One of traps defined by this MIB is rdbmsOutOfSpace :

```
rdbmsOutOfSpace TRAP-TYPE
    ENTERPRISE rdbmsTraps
    VARIABLES { rdbmsSrvInfoDiskOutOfSpaces }
    DESCRIPTION
        "An rdbmsOutOfSpace trap signifies that one of the database
        servers managed by this agent has been unable to allocate
        space for one of the databases managed by this agent. Care
        should be taken to avoid flooding the network with these traps."
    ::= 2
```

The enterprise is rdbmsTraps and the specific trap number is 2. This trap has one variable binding, rdbmsSrvInfoDiskOutOfSpaces. If we look elsewhere in the MIB, we will find that this variable is a scalar object. Its definition is:

```
rdbmsSrvInfoDiskOutOfSpaces OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The total number of times the server has been unable to obtain
        disk space that it wanted, since server startup. This would be
        inspected by an agent on receipt of an rdbmsOutOfSpace trap."
    ::= { rdbmsSrvInfoEntry 9 }
```

The DESCRIPTION for this object indicates why the note about taking care to avoid flooding the network (in the DESCRIPTION text for the TRAP-TYPE) is so important. Every time the RDBMS is unable to allocate space for the database, the agent will send a trap. A busy (and full) database could end up sending this trap thousands of times a day.

Some commercial RDBMS vendors, such as Oracle, provide an SNMP agent with their database engines. Agents such as these typically have functionality above and beyond that found in the RDBMS MIB.

2.6.7 SNMP Notification

In an effort to standardize the PDU format of SNMPv1 traps (recall that SNMPv1 traps have a different PDU format from *get* and *set*), SNMPv2 defines a **NOTIFICATION-TYPE**. The PDU format for **NOTIFICATION-TYPE** is identical to that for *get* and *set*. RFC 2863 redefines the *linkDown* generic notification type like so:

linkDown NOTIFICATION-TYPE

OBJECTS { *ifIndex*, *ifAdminStatus*, *ifOperStatus* }

STATUS *current*

DESCRIPTION

"A *linkDown* trap signifies that the SNMPv2 entity, acting in an agent role, has detected that the *ifOperStatus* object for one of its communication links left the down state and transitioned into some other state (but not into the *notPresent* state). This other state is indicated by the included value of *ifOperStatus*."

::= { *snmpTraps* 3 }

The list of bindings is called **OBJECTS** rather than **VARIABLES**, but little else has changed. The first object is the specific interface (*ifIndex*) that transitioned from the *linkDown* condition to some other condition. The OID for this trap is 1.3.6.1.6.3.1.1.5.3, or *iso.org.dod.internet.snmpV2.snmpModules.snmpMIB.snmpMIBObjects.snmpTraps.linkDown*.

2.6.8 SNMP inform

Finally, SNMPv2 provides an *inform* mechanism, which allows for manager-to-manager communication. This operation can be useful when the need arises for more than one NMS in the network. When an *inform* is sent from one NMS to another, the receiver sends a response to the sender acknowledging receipt of the event. This behavior is similar to that of the *get* and *set* requests. Note that an SNMP *inform* can be used to send SNMPv2 traps to an NMS. If you use an *inform* for this purpose, the agent will be notified when the NMS receives the trap.

2.6.9 SNMP report

The *report* operation was defined in the draft version SNMPv2 but never implemented. It is now part of the SNMPv3 specification and is intended to allow SNMP engines to communicate with each other (mainly to report problems with processing SNMP messages).

2.7 Host Management Revisited

Managing your hosts is an important part of network management. You would think that the Host Resources MIB would be part of every host-based SNMP agent, but this isn't the case. Some SNMP agents implement this MIB, but many don't. A few agents go further and implement proprietary extensions based upon this MIB. This is mainly due to the fact that this MIB was intended to serve as a basic, watered-down framework for host management, designed mainly to foster wide deployment.

The Host Resources MIB defines the following seven groups:

```
host          OBJECT IDENTIFIER ::= { mib-2 25 }

hrSystem      OBJECT IDENTIFIER ::= { host 1 }
hrStorage     OBJECT IDENTIFIER ::= { host 2 }
hrDevice      OBJECT IDENTIFIER ::= { host 3 }
hrSWRun       OBJECT IDENTIFIER ::= { host 4 }
hrSWRunPerf   OBJECT IDENTIFIER ::= { host 5 }
hrSWInstalled OBJECT IDENTIFIER ::= { host 6 }
```

The host OID is 1.3.6.1.2.1.25 (iso.org.dod.internet.mgmt.mib-2.host). The remaining six groups define various objects that provide information about the system.

The hrSystem (1.3.6.1.2.1.25.1) group defines objects that pertain to the system itself. These objects include uptime, system date, system users, and system processes.

The hrDevice (1.3.6.1.2.1.25.3) and hrStorage (1.3.6.1.2.1.25.2) groups define objects pertaining to filesystems and system storage, such as total system memory, disk utilization, and CPU nonidle percentage. They are particularly helpful, since they can be used to manage the disk partitions on your host. You can even use them to check for errors on a given disk device.

The hrSWRun (1.3.6.1.2.1.25.4), hrSWRunPerf (1.3.6.1.2.1.25.5), and hrSWInstalled (1.3.6.1.2.1.25.6) groups define objects that represent various aspects of software running or installed on the system. From these groups, you can determine what operating system is running on the host, as well as what programs the host is currently running. The hrSWInstalled group can be used to track which software packages are installed.

As you can see, the Host Resources MIB provides some necessary system-management objects that can be utilized by almost anyone who needs to manage critical systems.

2.8 Remote Monitoring Revisited

A thorough treatment of RMON is beyond the scope of this book, but it's worth discussing the groups that make up RMONv1. RMON probes are typically stand-alone devices that watch traffic on the network segments to which they are attached. Some vendors

implement at least some kind of RMON probe in their routers, hubs, or switches. [Chapter 9](#) provides an example of how to configure RMON on a Cisco router.

The RMON MIB defines the following 10 groups:

rmon	OBJECT IDENTIFIER ::= { mib-2 16 }
statistics	OBJECT IDENTIFIER ::= { rmon 1 }
history	OBJECT IDENTIFIER ::= { rmon 2 }
alarm	OBJECT IDENTIFIER ::= { rmon 3 }
hosts	OBJECT IDENTIFIER ::= { rmon 4 }
hostTopN	OBJECT IDENTIFIER ::= { rmon 5 }
matrix	OBJECT IDENTIFIER ::= { rmon 6 }
filter	OBJECT IDENTIFIER ::= { rmon 7 }
capture	OBJECT IDENTIFIER ::= { rmon 8 }
event	OBJECT IDENTIFIER ::= { rmon 9 }

RMONv1 provides packet-level statistics about an entire LAN or WAN. The rmon OID is 1.3.6.1.2.1.16
(iso.org.dod.internet.mgmt.mib-2.rmon). RMONv1 is made up of nine groups:

statistics (1.3.6.1.2.1.16.1)

Contains statistics about all the Ethernet interfaces monitored by the probe

history (1.3.6.1.2.1.16.2)

Records periodic statistical samples from the statistics group

alarm (1.3.6.1.2.1.16.3)

Allows a user to configure a polling interval and a threshold for any object the RMON probe records

hosts (1.3.6.1.2.1.16.4)

Records traffic statistics for each host on the network

hostTopN (1.3.6.1.2.1.16.5)

Contains host statistics used to generate reports on hosts that top a list ordered by a parameter in the host table

matrix (1.3.6.1.2.1.16.6)

Stores error and utilization information for sets of two addresses

filter (1.3.6.1.2.1.16.7)

Matches packets based on a filter equation; when a packet matches the filter, it may be captured or an event may be generated

capture (1.3.6.1.2.1.16.8)

Allows packets to be captured if they match a filter in the filter group

event (1.3.6.1.2.1.16.9)

Controls the definition of RMON events

RMONv2 enhances RMONv1 by providing network- and application-level statistical gathering. Since the only example of RMON in

this book uses RMONv1, we will stop here and not go into RMONv2. However, we encourage you to read RFC 2021 to get a feel for what enhancements this version of RMON brings to network monitoring.

Chapter 3. NMS Architectures

Now that you understand the basic concepts behind how management stations (NMSs) and agents communicate, it's time to introduce the concept of a network-management architecture. Before rushing out to deploy SNMP management, you owe it to yourself to put some effort into developing a coherent plan. If you simply drop NMS software on a few of your favorite desktop machines, you're likely to end up with something that doesn't work very well. By NMS architecture, we mean a plan that helps you use NMSs effectively to manage your network. A key component of network management is selecting the proper hardware (i.e., an appropriate platform on which to run your NMS) and making sure that your management stations are located in such a way that they can observe the devices on your network effectively.

3.1 Hardware Considerations

Managing a reasonably large network requires an NMS with substantial computing power. In today's complex networked environments, networks can range in size from a few nodes to thousands of nodes. The process of polling and receiving traps from hundreds or thousands of managed entities can be taxing on the best of hardware. Your NMS vendor will be able to help you determine what kind of hardware is appropriate for managing your network. Most vendors have formulas for determining how much RAM you will need to achieve the level of performance you want, given the requirements of your network. It usually boils down to the number of devices you want to poll, the amount of information you will request from each device, and the interval at which you want to poll them. The software you want to run is also a consideration. NMS products such as OpenView are large, heavyweight applications; if you want to run your own scripts with Perl, you can get away with a much smaller management platform.

Is it possible to say something more helpful than "ask your vendor"? Yes. First, although we've become accustomed to thinking of NMS software as requiring a midrange workstation or high-end PC, desktop hardware has advanced so much in the past year or two that running this software is within the range of any modern PC. Specifically, surveying the recommendations of a number of vendors, we have found that they suggest a PC with at least a 300 MHz CPU, 128 MB of memory, and 500 MB of disk

space. Requirements for Sun SPARC and HP workstations are similar.

Let's look at each of these requirements:

300 MHz CPU

This is well within the range of any modern desktop system, but you probably can't bring your older equipment out of retirement to use as a management station.

128 MB of memory

You'll probably have to add memory to any off-the-shelf PC; Sun and HP workstations come with more generous memory configurations. Frankly, vendors tend to underestimate memory requirements anyway, so it won't hurt to upgrade to 256 MB. Fortunately, RAM is cheap these days. (Memory prices fluctuate from day to day, but we recently found 256 MB DIMMs for under \$100.)

500 MB of disk space

This recommendation is probably based on the amount of space you'll need to store the software, and not on the space you'll need for log files, long-term trend data, etc. But again, disk space is cheap these days, and skimping is counterproductive.

Let's think a bit more about how long-term data collection affects your disk requirements. First, you should recognize that some products have only minimal data-collection facilities, while others exist purely for the purpose of collecting data (for example, MRTG). Whether or not you can do data collection effectively depends to some extent on the NMS product you've selected. Therefore, before deciding on a software product, you should think about your data-collection requirements. Do you want to do long-term trend analysis? If so, that will affect both the software you choose and the hardware on which you run it.

For a starting point, let's say that you have 1,000 nodes, you want to collect data every minute, and you're collecting 1 KB of data per node. That's 1 MB per minute, 1.4 GB per day -- you'd fill a 40 GB disk in about a month. That's bordering on extravagant. But let's look at the assumptions:

- Collecting data every minute is certainly excessive; every 10 minutes should do. Now your 40 GB disk will store almost a year's worth of data.
- 1,000 nodes isn't that big a network. But do you really want to store trend data for all your users' PCs? Much of this book is devoted to showing you how to control the amount of data you collect. Instead of 1,000 nodes, let's first count interfaces. And let's forget about desktop systems -- we really care only about trend data for our network backbone: key servers, routers, switches, etc. Even on a midsize network, we're probably talking about 100 or 200 interfaces.

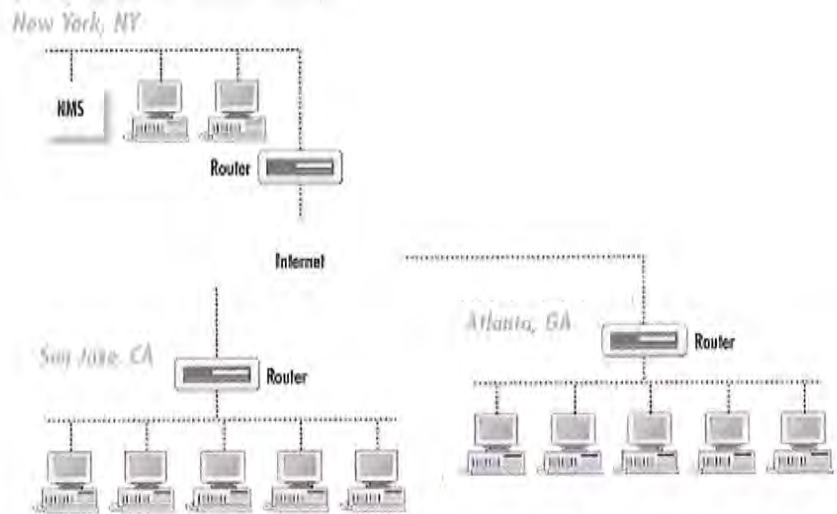
- The amount of data you collect per interface depends on many factors, not the least of which is the format of the data. An interface's status may be up or down -- that's a single bit. If it's being stored in a binary data structure, it may be represented by a single bit. But if you're using syslog to store your log data and writing Perl scripts to do trend analysis, your syslog records are going to be 80 bytes or so even if you are storing only one bit of information. Data-storage mechanisms range from syslog to fancy database schemes -- you obviously need to understand what you're using, and how it will affect your storage requirements. Furthermore, you need to understand how much information you really want to keep per interface. If you want to track only the number of octets going in and out of each interface and you're storing this data efficiently, your 40 GB disk could easily last the better part of a century.

Seriously, it's hard to talk about what your storage requirements will be when they vary over two or three orders of magnitude. But the lesson is that no vendor can tell you what your storage requirements will be. A gigabyte should be plenty for log data on a moderately large network, if you're storing data only for a reasonable subset of that network, not polling too often, and not saving too much data. But that's a lot of variables, and you're the only one in control of them. Keep in mind, though, that the more data you collect, the more time and CPU power will be required to grind through all that data and produce meaningful results. It doesn't matter whether you're using expensive trend-analysis software or some homegrown scripts -- processing lots of data is expensive. At least in terms of long-term data collection, it's probably better to err by keeping too little data around than by keeping too much.

3.2 NMS Architectures

Before going out and buying all your equipment, it's worth spending some time coming up with an architecture for your network that will make it more manageable. The simplest architecture has a single management station that is responsible for the entire network, as shown in [Figure 3-1](#).

Figure 3-1. Single NMS architecture

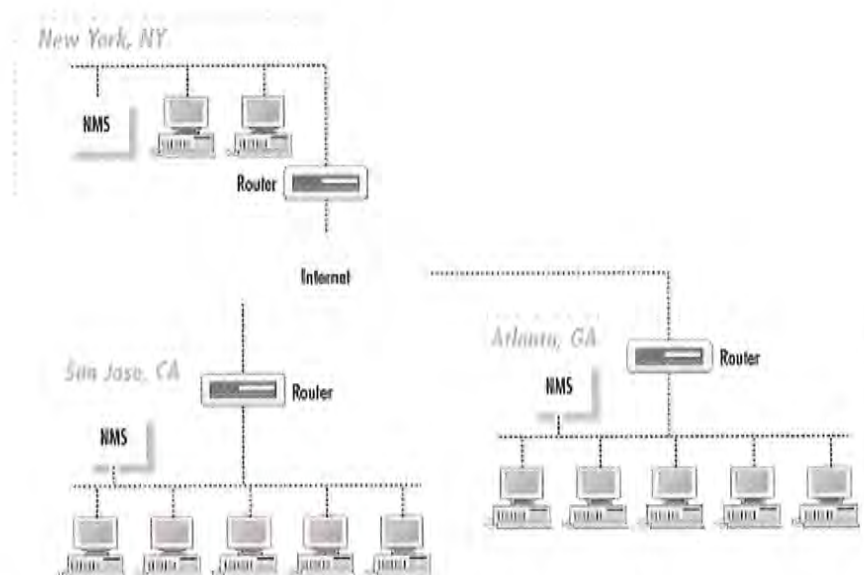


The network depicted in [Figure 3-1](#) has three sites: New York, Atlanta, and San Jose. The NMS in New York is responsible for managing not only the portion of the network in New York, but also those in Atlanta and San Jose. Traps sent from any device in Atlanta or San Jose must travel over the Internet to get to the NMS in New York. The same thing goes for polling devices in San Jose and Atlanta: the NMS in New York must send its requests over the Internet to reach these remote sites. For small networks, an architecture like this can work well. However, when the network grows to the point that a single NMS can no longer manage everything, this architecture becomes a real problem. The NMS in New York can get behind in its polling of the remote sites, mainly because it has so much to manage. The result is that when problems arise at a remote site, they may not get noticed for some time. In the worst case, they might not get noticed at all.

It's also worth thinking about staffing. With a single NMS, your primary operations staff would be in New York, watching the health of the network. But problems frequently require somebody onsite to intervene. This requires someone in Atlanta and San Jose, plus the coordination that entails. You may not need a full-time network administrator, but you will need someone who knows what to do when a router fails.

When your network grows to a point where one NMS can no longer manage everything, it's time to move to a distributed NMS architecture. The idea behind this architecture is simple: use two or more management stations and locate them as close as possible to the nodes they are managing. In the case of our three-site network, we would have an NMS at each site. [Figure 3-2](#) shows the addition of two NMSs to the network.

Figure 3-2. Distributed NMS architecture



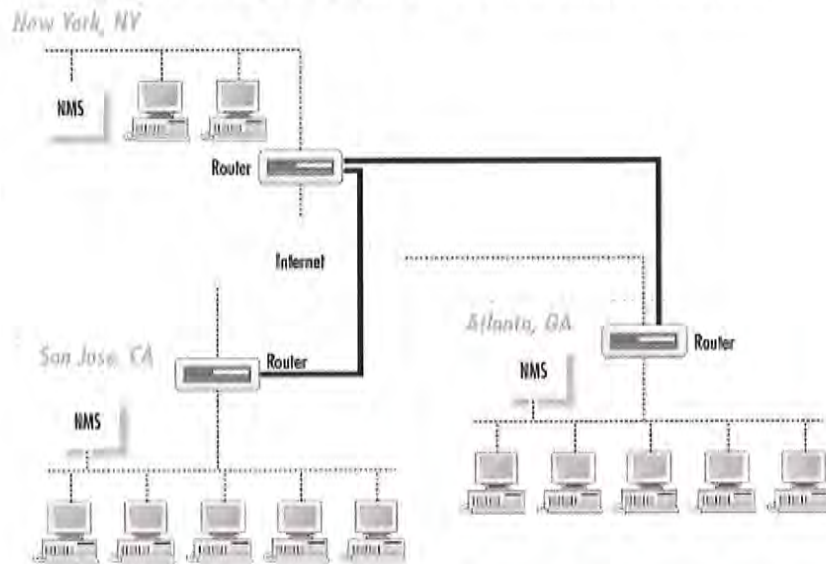
This architecture has several advantages, not the least of which is flexibility. With the new architecture, the NMSs in Atlanta and San Jose can act as standalone management stations, each with a fully self-sufficient staff, or they can forward events to the NMS in New York. If the remote NMSs forward all events to the NMS in New York, there is no need to put additional operations staff in Atlanta and San Jose. At first glance this looks like we've returned to the situation of [Figure 3-1](#), but that isn't quite true. Most NMS products provide some kind of client interface to viewing the events currently in the NMS (traps received, responses to polls, etc.). Since the NMS that forwards events to New York has already discovered the problem, we're simply letting the NMS in New York know about it so it can be dealt with appropriately. The New York NMS didn't have to use valuable resources to poll the remote network to discover that there was a problem.

The other advantage is that if the need arises you can put operations staff in Atlanta and San Jose to manage each of these remote locations. If New York loses connectivity to the Internet, events forwarded from Atlanta or San Jose will not make it to New York. With operations staff in Atlanta and San Jose, and the NMSs at these locations acting in standalone mode, a network outage in New York won't matter. The remote-location staff will continue on as if nothing has happened.

Another possibility with this architecture is a hybrid mode: you staff the operations center in New York 24 hours a day, 7 days a week, but you staff Atlanta and San Jose only during business hours. During off-hours, they rely on the NMS and operations staff in New York to notice and handle problems that arise. But during the critical (and busiest) hours of the day, Atlanta and San Jose don't have to burden the New York operators.

Both of the architectures we have discussed use the Internet to send and receive management traffic. This poses several problems, mainly dealing with security and overall reliability. A better solution is to use private links to perform all your network-management functions. [Figure 3-3](#) shows how the distributed NMS architecture can be extended to make use of such links.

Figure 3-3. Using private links for network management



Let's say that New York's router is the core router for the network. We establish private (but not necessarily high-speed) links between San Jose and New York, and between New York and Atlanta. This means that San Jose will not only be able to reach New York, but it will also be able to reach Atlanta via New York. Atlanta will use New York to reach San Jose, too. The private links (denoted by thicker router-to-router connections) are primarily devoted to management traffic, though we could put them to other uses. Using private links has the added benefit that our community strings are never sent out over the Internet. The use of private network links for network management works equally well with the single NMS architecture, too. Of course, if your corporate network consists entirely of private links and your Internet connections are devoted to external traffic only, using private links for your management traffic is the proverbial "no-brainer."

One final item worth mentioning is the notion of trap-directed polling. This doesn't really have anything to do with NMS architecture, but it can help to alleviate an NMS's management strain. The idea behind trap-directed polling is simple: the NMS receives a trap and initiates a poll to the device that generated the trap. The goal of this scenario is to determine if there is indeed a problem with the device, while allowing the NMS to ignore (or devote few resources to) the device in

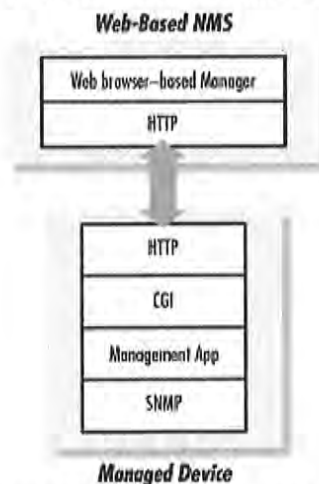
normal operation. If an organization relies on this form of management, it should implement it in such a way that non-trap-directed polling is almost done away with. That is, it should avoid polling devices at regular intervals for status information. Instead, the management stations should simply wait to receive a trap before polling a device. This form of management can significantly reduce the resources needed by an NMS to manage a network. However, it has an important disadvantage: traps can get lost in the network and never make it to the NMS. This is a reality of the connectionless nature of UDP and the imperfect nature of networks.

3.3 A Look Ahead

Web-based network management entails the use of the *HyperText Transport Protocol* (HTTP) and the *Common Gateway Interface* (CGI) to manage networked entities. It works by embedding a web server in an SNMP-compatible device, along with a CGI engine to convert SNMP-like requests (from a web-based NMS) to actual SNMP operations, and vice versa. Web servers can be embedded into such devices at very low monetary and operating cost.

Figure 3-4 is a simplified diagram of the interaction between a web-based NMS and a managed device. The CGI application bridges the gap between the management application and the SNMP engine. In some cases, the management application can be a collection of Java applets that are downloaded to the web browser and executed on the web-based manager. Current versions of OpenView ship with a web-based GUI.

Figure 3-4. Web-based network management



Web-based network management could eliminate, or at least reduce, the need for traditional NMS software. NMS software can be expensive to purchase, set up, and maintain. Most of today's major NMS vendors support only a few popular versions of Unix, and have only recently begun to support Windows 9x/NT/2000, thus limiting your operating-system choices. With a web-based

NMS, however, these two concerns are moot. For the most part web-browser technology is free, and Netscape Communications (now AOL Time Warner) supports many flavors of Unix, as well as the Wintel and Apple platforms.

Web-based network management should not be viewed as a panacea, though. It is a good idea, but it will take some time for vendors to embrace this technology and move toward web-integration of their existing products. There is also the issue of standardization, or the lack of it. The *Web-Based Enterprise Management* (WBEM) consortium addresses this by defining a standard for web-based management. Industry leaders such as Cisco and BMC Software are among the original founders of WBEM. You can learn more about this initiative at the Distributed Management Task Force's web page, <http://www.dmtf.org/wbem/>.

Chapter 4. SNMP-Compatible Hardware

Determining if you have devices that are manageable by SNMP is a good place to start down the path to network-management Zen. Before we get into how to determine if what you already have is manageable, we will briefly discuss what makes a device SNMP-compatible.

Vendors do not have to implement all the MIBs SNMP provides,^[1] but SNMP-manageable devices must support at least MIB-II. It also behooves the vendors to implement some of the more useful MIBs, as well as their own private MIBs, since the ability to manage a product effectively using SNMP is an increasingly important selling point.

^[1] You can find a few examples of these standard MIBs in [Chapter 1](#).

4.1 What Does SNMP-Compatible Really Mean?

Many vendors claim that their products are SNMP-compatible or compliant. For the most part this is true. What they actually mean is that their product supports a set of SNMP operations, as well as MIB-II. For SNMPv1 compatibility, the supported operations include:

- *get*
- *get-next*
- *set*
- *get-response*
- *trap*

Additionally, if the product is SNMPv2 and SNMPv3 compatible, it must support the following operations:

- *get-bulk*
- *inform*
- *notification*
- *report*

Vendors can choose to support SNMPv1, SNMPv2, SNMPv2, or all three. An SNMP agent that supports two versions of SNMP is called "bilingual." In recent years, this was restricted to devices supporting SNMPv1 and SNMPv2. Now a device can support all three versions, which technically makes it trilingual. It is possible for an agent to speak all versions of SNMP because SMIV2 is a superset of SMIV1, and SMIV2 is used, for the most part, with SNMPv3.

Supporting these operations, however, is only one piece to the puzzle of providing a manageable product. The other piece is providing a private MIB that is comprehensive enough to give network managers the information they need to manage their networks intelligently. In today's complex network environments, it does not pay to purchase equipment that has a minimal or poorly implemented private MIB. For instance, it is important to measure ambient temperature inside devices such as routers, hubs, and switches. Cisco and others provide this information via their private MIBs; other vendors do not. If you're in the process of purchasing a high-end router, you might want to look into the vendors' private MIBs to see which vendors provide more relevant information.

Another factor that affects vendor MIB support is product definition. Concord Communications (vendors of an SNMP agent for Unix and Windows) will probably not support the RS-232 MIB (RFC 1659), since their product is geared toward providing system- and application-management information. 3Com, on the other hand, implemented this MIB for their line of Dual Speed Hubs, since these hubs have RS-232 ports.

4.2 Is My Device SNMP-Compatible?

Your product documentation should be helpful in determining hardware or software compatibility with SNMP. You can also consult your sales representative, or customer support, if applicable. Another way to tell if a product is SNMP-compatible is to perform an *snmpget* query against the device in question.^[2] Issuing a diagnostic *get* against any device is easy. The most common way to accomplish this is to find a Unix host that has the *snmpget* binary command installed.^[3] There are several varieties of this command, so consult your manpage or

system administrator for help. The easiest variable to query for is sysDescr, which provides a description of the system being queried. Here's what happens when you use the Net-SNMP *snmpget* command to look at sysDescr on a typical Linux host:

^[2] With this method, we can try to guess what the community string is. In our case, we try public or private. If we don't get a response, it might mean either that we guessed wrong or that the agent isn't set up/configured.

^[3] [Chapter 7](#) discusses installing the Net-SNMP agent and toolkit, which comes with utilities such as *snmpget*.

```
$ snmpget linuxserver.ora.com public system.sysDescr.0
system.sysDescr.0 = "Linux version 2.0.34 (root@porky.redhat.com)
(gcc version 2.7.2.3) #1 Fri May 8 16:05:57 EDT 1998"
```

The response from linuxserver.ora.com is typical of most managed devices. Note, however, that there's nothing sacred about the actual description; the text you retrieve will vary from vendor to vendor. Issuing an *snmpget* against a Cisco 2503 router should return something like this:

```
$ snmpget orarouter.ora.com public system.sysDescr.0
system.sysDescr.0 = "Cisco Internetwork Operating System Software
..IOS (tm) 2500 Software (C2500-I-L), Version 11.2(5), RELEASE
SOFTWARE (fc1)..Copyright (c) 1986-1997 by cisco Systems, Inc...
Compiled Mon 31-Mar-97 19:53 by ckralik"
```

This router's system description tells us that it is running Version 11.2(5) of the Cisco IOS. This sort of information is generally useless, but it does tell us that the device is running an SNMP agent. Here's what happens when something goes wrong:

```
$ snmpget linuxserver.ora.com public system.sysDescr.0
Timeout: No Response from linuxserver.ora.com.
```

This message means that the Net-SNMP *snmpget* command did not receive a response from linuxserver.ora.com. A number of things could be wrong, one of which is that there is no SNMP agent running on the target host. But it's also possible that linuxserver has crashed, that there's some sort of network problem, or that everything is running correctly but you aren't using the correct community string. It's even possible that the device you're querying has SNMP capabilities, but the SNMP agent won't be enabled until you explicitly configure it.

If you suspect you have manageable equipment but are not sure, it is good to know that most vendors ship their products with the read and write community strings set to public and private, respectively. (The Net-SNMP tools we're using here use private as the default for both community strings.^[4])

^[4] Since our agents use public for the community string and Net-SNMP defaults to private, we needed to specify the community string public on the command line.

Once you verify that the device you're testing is SNMP-manageable, you should immediately change the community strings. Leaving the community strings set to well-known values like public and private is a serious security problem.

Once you've established that your device supports SNMP, you can go further to check if it supports Version 2. A good way to do that is to make a request that can be answered only by a Version 2 agent, such as the *bulk-get* request. You can use the *snmpbulkget* command we demonstrated in [Chapter 2](#) to make such a request:

```
$ snmpbulkget -v2c -B 1 3 linux.ora.com public sysDescr ifInOctets ifOutOctets
system.sysDescr.0 = "Linux linux 2.2.5-15 #3 Thu May 27 19:33:18 EDT 1999 i686"
interfaces.ifTable.ifEntry.ifInOctets.1 = 70840
interfaces.ifTable.ifEntry.ifOutOctets.1 = 70840
interfaces.ifTable.ifEntry.ifInOctets.2 = 143548020
interfaces.ifTable.ifEntry.ifOutOctets.2 = 111725152
interfaces.ifTable.ifEntry.ifInOctets.3 = 0
interfaces.ifTable.ifEntry.ifOutOctets.3 = 0
```

Now we know that linux.ora.com supports SNMPv2 -- in particular, v2c. Can we go further, and check for Version 3 support? For Version 3, you're better off checking your vendor's documentation. Most vendors don't support Version 3 yet, and we expect adoption to be fairly slow -- many vendors still support only Version 1.

4.3 Upgrading Your Hardware

Now that you know whether or not you have SNMP devices on your network, it might be time to upgrade! You may find that some of the devices you would like to manage don't support SNMP. There are two ways to upgrade: you can retire your existing equipment and buy newer, more manageable hardware, or you can upgrade your equipment's firmware (if provided by the vendor) to a version that supports SNMP. Some vendors, however, will offer to buy back older equipment, or even give a discount for turning in a competitor's equipment.

Of course, updating your equipment may not be necessary. If you have software applications that are used to manage non-SNMP equipment and they work, there is no need to upgrade. If you're reasonably handy with scripts and want to learn about SNMP in some depth, you may find that it's possible to write scripts that allow you to use SNMP to monitor applications that doesn't support SNMP using wrapper/scripts. For an example of this, see [xref linkend="enettgdg-CHP-12-SECT-4"/>](#) in [Chapter 12](#).

Whatever approach you take, realize that SNMP exists to provide a consistent way to manage networked equipment. If you're currently managing your network using a number of legacy management tools, each supporting a few devices from a particular vendor, SNMP provides a way out. You may be comfortable with your old tools -- but it will become increasingly convenient to use SNMP to provide a uniform network-management approach.

4.4 In the End

You may have been purchasing SNMP-compatible devices for years without knowing it. As SNMP has become more popular, it has been incorporated into more and more devices. SNMP compatibility has become a true selling point for most vendors.

It goes without saying that most network devices support SNMP, including routers, bridges, hubs, servers, and desktop PCs.^[5] However, many other kinds of equipment are also manageable via SNMP, including uninterruptible power supplies (UPSs), air-conditioning units, and other important pieces of your infrastructure. After you identify which routers and hubs are SNMP-compatible, keep your eyes open for other devices that may need to be managed. While SNMP is very good at managing your network, hosts, hubs, and routers, it's not limited to only your networking environment.

^[5] Low-end hubs, switches, and routers designed for home use probably won't support SNMP. Hubs and switches that support SNMP usually are advertised as "manageable" and usually cost significantly more. For routers, you'll have to read the literature carefully.

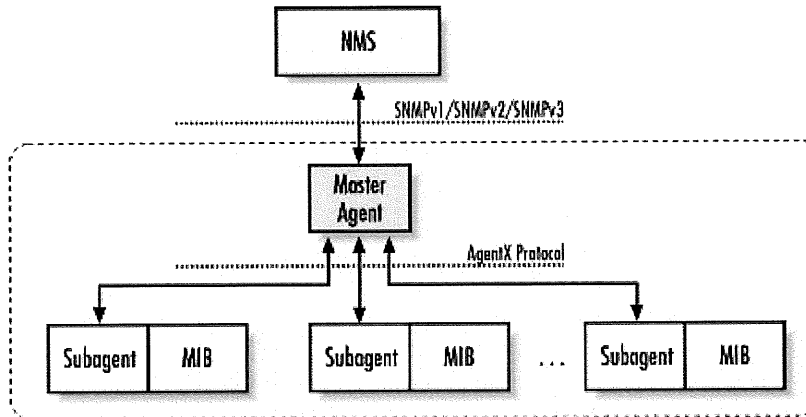
4.5 A Look Ahead

The Internet Engineering Task Force (IETF) is in the process of defining a standards-track technology for SNMP *agent extensibility* (AgentX). As we defined it earlier, an SNMP agent is software that resides on a managed device, replying to SNMP requests and generating asynchronous traps. Information about agent extensibility can be found in RFC 2741, as well as at the AgentX web site, <http://www.scguild.com/agentx/>. The need for AgentX arises from the inability to add and remove MIB objects while an agent is running; in other words, the lack of a standard way to extend an agent's functionality. The *SNMP Multiplexing Protocol* (SMUX, RFC 1227) was an early attempt to provide standardization for agent extensibility, but the protocol was found to be flawed and has since been abandoned.

Figure 4-1 is an overview of the AgentX architecture. With AgentX, the agent consists of a single processing entity called a *master agent* and zero or more processing entities called

subagents. The master agent and subagents can reside on the same device or communicate via a proxy device. The master agent communicates with the NMS, much like a traditional SNMP agent. The subagents have direct access to the MIB, whereas the master agent does not. Consequently, the subagents perform management functions on managed variables, then communicate this information to the master agent via the AgentX protocol, which is not SNMP-based.

Figure 4-1. AgentX architecture



Without a standardized approach to extensibility, it is very difficult for vendors to keep track of extensions to agents for the various platforms they support. AgentX tries to address this by giving vendors a consistent interface for extending agents. It also establishes the notion of MIB regions, or sets of managed variables. A subagent is responsible for registering these MIBs with a single master agent. In practice this means that vendors will have a subagent for each MIB they implement; for example, an RMON subagent, a MIB-II subagent, a Host Resources subagent, and others. This helps vendors because it gives them a way to add and remove MIB instances from an agent without disturbing the actual operation between an NMS and the agent.

Chapter 5. Network-Management Software

Many SNMP software packages are available, ranging from programming libraries that let you build your own utilities (using Perl, C/C++ or Java) to expensive, complete network-management platforms. This chapter presents some pros and cons for many of the most commonly used packages. This will not only give you an idea of what packages are out there, but also help you decide what might be right for you (keep in mind, though, that these pros and cons are merely our opinions). Whenever possible, we present both open source solutions and commercial products.

Management software falls into five categories:

- SNMP agents

- NMS suites
- Element managers (vendor-specific management)
- Trend-analysis software
- Supporting software

Unfortunately, deciding what you need isn't as simple as picking one program from each category. If you have a small network and are interested in building your own tools, you probably don't need a complex NMS suite. Whether or not you need trend-analysis software depends, obviously, on if you're interested in analyzing trends in your network usage. The products available depend in part on the platforms in which you're interested. The minimum you can get by with is an SNMP agent on a device and some software that can retrieve a value from that device (using an SNMP *get*). Although this is minimal, it's enough to start working, and you can get the software for free.

This chapter presents a broad survey of some of the leading products in each of these categories. Since there are more packages than we can cover in this book, be sure to check the Network Management Server (<http://netman.cit.buffalo.edu/Products.html>) for network-management product listings.

5.1 SNMP Agents

As we explained in [Chapter 1](#), the agent is the software that controls all the SNMP communication to and from any SNMP-compatible device. In some devices, such as Cisco routers, the agent software is built into the device itself and requires no installation. On other platforms, such as Windows NT, you may have to install the agent as part of an additional software package.

Before you can look at what types of agents you need, you must research what types of devices you have on your network and what types of information you would like to receive from each. Some agents are very basic and return only a limited amount of information, while others can return a wealth of information. To start, determine whether you need to receive information from servers (Unix, Windows NT, etc.) or network devices (routers, switches, etc.). Generally, out-of-the-box network-type devices provide more information than their server counterparts. On the other hand, network devices do not extend very easily, if at all, in part because network hardware usually doesn't have a disk-based operating environment.^[1] This keeps the end user from accessing the agent to make modifications or extend it. The rest of this section provides

information about some of the software packages that are currently available for use as SNMP agents.

^[1] See [Chapter 11](#) for a discussion of extensible agents.



Make sure that you understand what kind of software is running on your servers (email systems, accounting packages, etc.). Many applications will not listen or respond to SNMP requests, but will send out traps. Traps can be very useful for monitoring some of these applications. Also, there are applications for virus scanners, remote logins (pcAnywhere), and UPSs that will send informative traps when an error has been found. Look for this feature the next time you purchase any package or software suite.

HP Extensible SNMP Agent

<http://www.openview.hp.com>

Platforms

Solaris, HP-UX

Pros

Includes an *snmptrap* program and an HP agent that gives some additional functionality (mostly for HP systems). The agent is extensible using a subset of ASN.1.

Cons

Cost is per device. You have to keep track of multiple daemons.

Sun Microsystems

<http://www.sun.com>

Platforms

Solaris

Pros

Available free for most recent versions of Solaris. Comes bundled with Solaris (Versions 2.6 and up). The agent is extensible.

Cons

Very minimal; supports only MIB-II.

Concord SystemEDGE

<http://www.empire.com>

Platforms

Many flavors of Unix, Windows NT

Pros

Provides very detailed information about the system (CPU, disk space, filesystems, installed apps, etc.). Integrates with the Windows NT SNMP service. Log watcher for Unix and NT. The agent is fully extensible. Works with Concord's Network Health package and Trinagy's TREND suite.

Cons

Can be expensive unless you purchase in quantity.

Microsoft

<http://www.microsoft.com>

Platforms

Windows 9x/NT/2000

Pros

Built into the operating-system kernel. Can be controlled by NT services.

Cons

Meets only the minimal requirements of an SNMP-compatible agent. You must install the latest service pack after you install the software.

Net-SNMP (Formerly the UCD-SNMP project.)

<http://net-snmp.sourceforge.net>

Platforms

Many flavors of Unix, Windows 9x/NT

Pros

Free and fairly robust. Easily extensible using shell or Perl scripts. Includes a trap daemon.

Cons

Documentation is minimal, which means it can be difficult for first-time users to get it running the way they want.

SNMP Research

<http://www.int.snmp.com>

Platforms

Unix, Windows NT

Pros

Good toolkit for writing an agent, if this is the functionality you're looking for.

Cons

Does not integrate with Windows SNMP Service. Mostly a toolkit product; requires extensive work to make it useful.

5.2 NMS Suites

We use the term "suite" to mean a software package that bundles multiple applications into one convenient product. In this section, we discuss NMS software, which is one of the more important pieces of the network-management picture. Without it, the agent software in the previous section is virtually useless. NMS products allow you to have a total network view of your servers, routers, switches, etc. In most cases, this view is a graphical representation of your network, with lots of neat labels and icons. These packages are very configurable and work in almost any network environment. With this freedom, however, often comes a big price tag and a confusing setup process. Some of the products focus more on the network side of management (i.e., devices such as routers, hubs, and switches). Others go a step beyond this and allow you to customize server and workstation agents to integrate nicely into your NMSs. Keep in mind that the bigger packages are for larger, more complicated networks and require extensive training. Be sure to take some time to research the packages before purchasing; if at all possible, get trial versions. The rest of this section lists some of the more common NMS packages.

HP OpenView NNM

<http://www.openview.hp.com>

Platforms

Solaris, HP-UX, Windows NT/2000

Pros

Great mid- to large-business SNMP suite. While it can be complicated, it is manageable with a little help from OpenView support. Has a nice graphical map and event-monitoring system. Can do some historical trend analysis. Price seems right and can be trimmed by obtaining a license for a limited number of managed nodes.

Cons

Not many third-party application plug-ins available.

HP OpenView ITO

<http://www.openview.hp.com>

Platforms

Solaris, HP-UX, Windows NT/2000

Pros

If you're a Fortune 500 company looking to implement OpenView on steroids, ITO is your product. It is very user-centered. Maps, events, and more can be displayed or hidden based on a user's profile. The event system is more like a ticket center. A wealth of third-party "smart plug-ins" are available.

Cons

Price can be very high. Made for serious enterprise companies. Not many people can properly implement this without training or outside consulting help.

Tivoli Netview

<http://www.tivoli.com/products/index/netview/>

Platforms

OS/390, Solaris, AIX, Digital UNIX, Windows NT (Intel and Alpha)

Pros

A truly distributed network-management solution. It has the ability to detect problems at the source before they affect users.

Cons

This is a heavyweight management system that requires extensive investment and resources to implement and operate.

Castle Rock SNMPc

<http://www.castlerock.com>

Platforms

Windows 98/NT/2000

Pros

Great for small to midsize companies. Contains everything you need to get an NMS up and running in your environment. Price is very reasonable, and it's loaded with features.

Cons

Network map could use a little work. Doesn't give a realistic representation of your network.

BMC

<http://www.bmc.com>

Platforms

Many platforms, including Unix and Windows NT

Pros

BMC has developed knowledge bases for managing most aspects of the enterprise, including networks, databases, and servers.

Cons

The knowledge modules are useful, but proprietary. The cost tends to be on the high side. Does not use SNMP as its native language.

Computer Associates Unicenter TNG Framework

<http://www.cai.com>

Platforms

Unix, Windows NT/2000

Pros

Can help you manage your entire IT business -- everything from traditional network management to your Oracle database system.

Cons

This is another heavyweight management system that can take substantial time, resources, and money to implement.

Veritas NerveCenter

<http://www.veritas.com>

Platforms

Solaris, HP-UX, Windows NT

Pros

Uses behavior models (finite state machines) to model real-world network situations. NerveCenter is designed to be a standalone polling engine, or to be used in conjunction with OpenView's graphical map. Perl subroutines can be compiled into the polling engine for later use.

Cons

Takes more effort to maintain than OpenView, and tends to be more complicated to operate.

OpenRiver

<http://www.riversoft.com>

Platforms

Solaris

Pros

RiverSoft, the company behind OpenRiver, boasts that their NMS provides "interventionless network management." They also provide true layer 2 and 3 network discovery. Despite the product's impressive abilities, it is priced very reasonably.

Cons

Currently available only for Solaris (although RiverSoft is planning a Windows NT release).

GxSNMP

<http://www.gxsnmp.org>

Platforms

Any Unix platform with an ANSI C compiler and the GTK/GDK toolkit installed

Pros

This free NMS comes with several nice features, such as a mapping tool (not auto discovery) and integration with SQL.

Cons

This project is still in its infancy (but there are many planned features that will make it a robust NMS solution).

Tkined <http://wwwhome.cs.utwente.nl/~schoenw/scotty/>

Platforms

Most Unix platforms, Windows NT

Pros

Tkined is a free extensible network management platform. It provides a network map and tools to perform discovery of IP networks. It can also perform management of devices with SNMP and non-SNMP standards (*ping*, *traceroute*, etc.). Tcl is used to extend and add functionality to Tkined.

Cons

You must be familiar with Tcl to extend this package.

OpenNMS <http://www.opennms.org>

Platforms

Any platform that supports Java

Pros

OpenNMS is an attempt to provide users with a truly open service and network-management product. It is written in Java and is released under the GNU Public License (GPL). It supports network discovery and distributed polling, among other things.

Cons

This project is still in its infancy.

5.3 Element Managers (Vendor-Specific Management)

These software packages are geared toward a certain type of vendor or function; for example, an element manager might be a product that focuses on managing a modem rack. Before purchasing such a package, take a good look at your present environment, how it's likely to grow, and what vendors you are currently using or are likely to use in the future. Because many of these products are vendor-specific, it's easy to buy something that turns out to be less useful than you expect. For example, CiscoView (part of the CiscoWorks suite) is a great piece of software; it does lots of fancy things, such as showing you the backs of your routers. However, if you purchase a number of Nortel devices a few months after installing this product, it won't be able to give you a unified view of your network. Some packages do allow you to manage their competitor's equipment; for example, an element manager that monitors switches may be able to handle switches from competing vendors. Before buying any of these products, research where your network is headed, and be sure to ask hard questions about the products' capabilities. The remainder of this section lists some of the available element managers.

Sun Management Center

<http://www.sun.com/symon/>

Platforms

Solaris, Windows (Console layer)

Pros

Provides a single point of management for all Sun servers, desktops, storage systems, the Solaris operating environment, applications, and datacenter services. This product scales to thousands of systems on a single, unified management platform and integrates easily with leading third-party platforms for added flexibility. It also has the ability to get real-time system performance and offers a free hardware diagnostic suite (plug-in) that detects hardware faults.

Cons

While it can manage and monitor other vendors, this ability doesn't come easily.

CiscoWorks 2000

<http://www.cisco.com>

Platforms

Solaris, HP-UX, AIX, Windows NT for some

Pros

This suite allows you to do everything from version control on your configuration files to latency graphs and detailed pictures of the backs of your devices. If you have a Cisco shop, stop everything and get this package!

Cons

The maps are a bit hokey. Doesn't produce a very friendly snapshot of your network, and has a hard time returning configurations to the devices. Would be nice if it could restore as easily as it backs up.

3Com Total Control

<http://www.3com.com>

Platforms

Solaris, Windows 9x

Pros

Allows the user to view the status of a modem rack by displaying an image of how the modem rack physically looks -- everything down to the bolts and logo. The user can proceed to reset individual cards or reset the entire chassis, among other things. This is a very slick product and can be very useful when trying to track down equipment problems.

Cons

Since newer 3Com Total Control chassis can have up to 336 modems, this can be a bear to start (it has to query the status of all the modems in the rack). The startup time can be greatly affected by the speed of the network between you and the chassis in question.

Aprisma (Formerly Spectrum for Cabletron hardware.)

<http://www.aprisma.com>

Platforms

Unix, Windows NT

Pros

Very good tool for managing Cabletron equipment, and is starting to add the ability to manage equipment from other vendors.

Cons

Complicated to set up and maintain. Meant for shops that need a high-end platform.

5.4 Trend Analysis

When faced with most network problems, it's nice to have some kind of historical record to give you an idea of when things started going wrong. This allows you to go back and review what happened before a problem appeared, and possibly prevent it from recurring. If you want to be proactive about diagnosing problems before they appear, it is essential to know what "normal" means for your network -- you need a set of baseline statistics that show you how your network normally behaves. While many of the bigger packages do some trend reporting, they can be clunky and hard to use. They might not even provide you with the kind of information you need. Once you see what a dedicated trend-analysis system can do, you will see why it might be worth the time, energy, and money to integrate one into your network-monitoring scheme.

If your environment calls for some serious monitoring, you should look into getting RMON probes. RMON probes are a great addition to trend-analysis packages, since most trend packages can make use of the kind of data these probes gather. The rest of this section lists some trend-analysis packages.

Concord eHealth

<http://www.concord.com>

Platforms

Solaris, HP-UX, Windows NT

Pros

Very professional, web-based graphs. Gives the user the ability to download and view reports in PDF. You can drill down in the web for more detailed reports. Great user management, allowing you to restrict users so that they see only what they need to see. Concord gives a free "Network Health Checkup." This is a great try-before-you-buy program. This program can also

interact with probes and server-based devices to give a full trend analysis of your network and servers.

Cons

Some people may get sticker-shock when they see the price. Licensing is done on a per-element basis.

Trinagy (Formerly DeskTalk Systems, Inc.) TREND

<http://www.desktalk.com>

Platforms

Unix, Windows 9x/NT

Pros

An excellent product for use in capacity planning. Out of the box, Trinagy supports 30, 60, and 90-day forecasts, among other calculations. Its report viewer is written in Java, so it is usable on most platforms. The report architecture is open, in that you can build your own reports.

Cons

Requires two weeks of training to run and administer. The pricing scheme is somewhat similar to eHealth's, since the size of the database depends on how many devices you query, how long you keep rate data around, etc. You can get more for less by tweaking polling intervals and retention times.

MRTG

<http://www.mrtg.org>

Platforms

Most Unix platforms, Windows NT

Pros

Free, easy to set up and use, very well documented. In addition to polling devices on your network, MRTG can receive input from non-SNMP sources.

Cons

Have to install multiple packages, which may be difficult to do on some platforms. For example, MRTG needs a specific SNMP Perl module to perform all of its polling duties. Not very scalable.

Cricket

<http://cricket.sourceforge.net>

Platforms

Most Unix platforms

Pros

Great tool that picks up where MRTG leaves off. It uses RRDTool, the next-generation version of MRTG.

Cons

Cricket is single-threaded, so collecting data from a good-sized network can take a while, especially if you gather usage data frequently. Having said that, it is fairly efficient, so you should not see any problem for quite some time.

InfoVista

<http://www.infovista.com>

Platforms

Unix, Windows NT

Pros

Very flexible and comes with some great reporting right out of the box.

Cons

Requires in-depth knowledge of network management and programming (Perl) in order to customize it to do anything beyond its out-of-the-box capabilities.

5.5 Supporting Software

Supporting software is a grab-bag that includes all sorts of things that are used in conjunction with the software packages listed earlier. Some of these packages can be used to write standalone SNMP applications. The rest of this section outlines several supporting software packages. Most of these are freely available and can be used with little or no previous experience.

Perl

<http://www.perl.com> <http://www.perl.org>

Platforms

Unix, Windows NT, Mac OS

Pros

The *Practical Extraction and Report Language* (Perl) is a versatile, all-purpose scripting language that is the tool of choice for system administrators and network engineers, among others. Both MRTG and Cricket make use of Perl to perform their behind-the-scenes work.

Cons

Some people say that there are no cons to Perl. The most common complaint about the language is that it's interpreted and not compiled, like the C programming language.

SNMP Support for Perl

<http://www.switch.ch/misc/leinen/snmp/perl/>
<http://www.cpan.org>

Platforms

Unix, Windows NT, Mac OS

Pros

Supplies easy-to-use subroutines that give access to the core SNMP functions. Widely tested, as it's the fundamental SNMP engine for the MRTG package.

Cons

Doesn't seem to have a lot of market exposure.

WILMA

<ftp://ftp.ldv.e-technik.tu-muenchen.de/dist/WILMA/INDEX.html>

Platforms

Most Unix platforms

Pros

Contains the core SNMP functions as well as a MIB compiler and browser.

Cons

Functions could be a bit more streamlined and user-friendly.

Net-SNMP C Library

<http://net-snmp.sourceforge.net>

Platforms

Unix, Windows 9x/NT

Pros

This library can be used to develop your own SNMP applications. The library is very easy to use, once you figure it out. The nice thing about the package is that it comes with the source code to commands such as *snmpget*, *snmpset*, and *snmpwalk*, which can be used to see how these sorts of operations are accomplished.

Cons

The documentation on how to use the library is poor to the point of nonexistence.

Net-SNMP Perl Module

<http://www.cpan.org/authors/id/GSM/>

Platforms

Unix, Windows 9x/NT

Pros

This library provides identical functionality to the Net-SNMP C library, except in Perl.

Cons

During installation, this module needs to have access to the Net-SNMP C library in order to work properly.

A3Com

<http://www.kernel.org/software/A3Com/>

Platforms

Unix, Windows NT

Pros

A simple set of modules that can be used to manage 3Com SuperStack II 3900/9300 and CoreBuilder 3500 LAN switches. This can be a good start for management on a budget.

Cons

The functionality is limited.

SNMP++

<http://rosegarden.external.hp.com/snmp++/>

Platforms

Unix (Linux, HP-UX, and Solaris), Windows

Pros

If you need to use C++ for SNMP application development, this is the package to get. You can create powerful applications with minimal programming. This library has been released into the open source community and is freely available.

Cons

Requires knowledge of C++.

Netcool

<http://www.micromuse.com>

Platforms

Unix, Windows NT

Pros

An event-correlation and deduplication engine, used to cut down on the management events that traditional NMS platforms tend to generate by showing the end user only what she needs to know to fix network problems. It is designed to receive events from NMSs such as OpenView or NerveCenter, but it can receive events from almost any kind of management source. Micromuse sells probes that can interface with everything from popular NMS platforms to telephone switch equipment.

Cons

Requires a bit of initial setup (but after that it's easy to use and maintain).

Platforms

Windows 95/NT

Pros

Easy to use and can be configured to perform actions on received traps.

Cons

Doesn't run on any flavor of Unix.

Chapter 6. Configuring Your NMS

Now that you have picked out some software to use in your environment, it's time to talk about installing and running it. In this chapter we will look at a few NMS packages in detail. While we listed several packages in [Chapter 5](#), we will dig into only a few packages here, and we'll use these packages in examples throughout the rest of the book. These examples should allow you to get most other SNMP-based network-management packages up and running with very little effort.

6.1 HP's OpenView Network Node Manager

Network Node Manager (NNM) is a licensed software product. The package includes a feature called "Instant-On" that allows you to use the product for a limited time (60 days) while you are waiting for your real license to arrive. During this period, you are restricted to a 250-managed-node license, but the product's capabilities aren't limited in any other way. When you install the product, the Instant-On license is enabled by default.



Check out the OpenView scripts located in OpenView's *bin* directory (normally */opt/OV/bin*). One particularly important group of scripts sets environment variables that allow you to traverse OpenView's directory structure much more easily. These scripts are named *ov.envvars.csh*, *ov.envvars.sh*, etc. (that is, *ov.envvars* followed by the name of the shell you're using). When you run the appropriate script for your

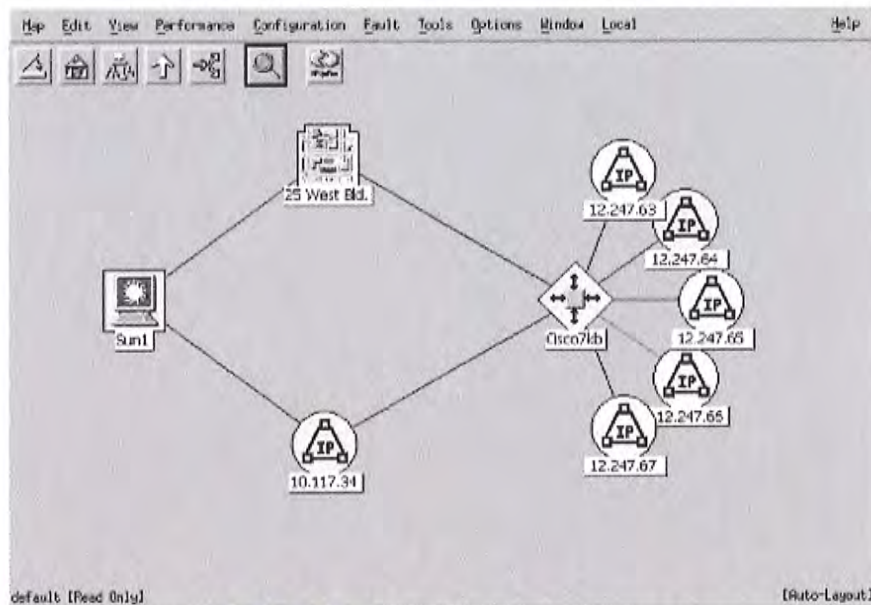
shell, it defines environment variables such as `$OV_BIN`, `$OV_MAN`, and `$OV_TMP`, which point to the OpenView *bin*, *man*, and *tmp* directories. Thus, you can easily go to the directory containing OpenView's manual pages with the command `cd $OV_MAN`. These environment variables are used throughout this book and in all of OpenView's documentation.

6.1.1 Running NNM

To start the OpenView GUI on a Unix machine, define your `DISPLAY` environment variable and run the command `$OV_BIN/ovw`. This starts OpenView's NNM. If your NNM has performed any discovery, the nodes it has found should appear under your Internet (top-level) icon. If you have problems starting NNM, run the command `$OV_BIN/ovstatus -c` and then `$OV_BIN/ovstart` or `$OV_BIN/ovstop`, respectively, to start or stop it. By default, NNM installs the necessary scripts to start its daemons when the machine boots. OpenView will perform all of its functions in the background, even when you aren't running any maps. This means that you do not have to keep a copy of NNM running on your console at all times and you don't have to start it explicitly when your machine reboots.

When the GUI starts, it presents you with a clickable high-level map. This map, called the Root map, provides a top-level view of your network. The map gives you the ability to see your network without having to see every detail at once. If you want more information about any item in the display, whether it's a subnet or an individual node, click on it. You can drill down to see any level of detail you want -- for example, you can look at an interface card on a particular node. The more detail you want, the more you click. [Figure 6-1](#) shows a typical NNM map.

Figure 6-1. A typical NNM map

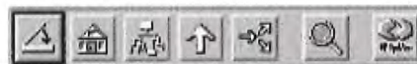


The menu bar (see [Figure 6-2](#)) allows you to traverse the map with a bit more ease. You have options such as closing NNM (the leftmost button), going straight to the Home map (second from the left),^[1] the Root map (third-left), the parent or previous map (fourth-left), or the quick navigator.^[2] There is also a button that lets you pan through the map or zoom in on a portion of it.

^[1] You can set any map as your Home map. When you've found the map you'd like to use, go to "Map → Submap → Set This Submap as Home."

^[2] This is a special map in which you can place objects that you need to watch frequently. It allows you to access them quickly without having to find them by searching through the network map.

Figure 6-2. OpenView NNM menu bar



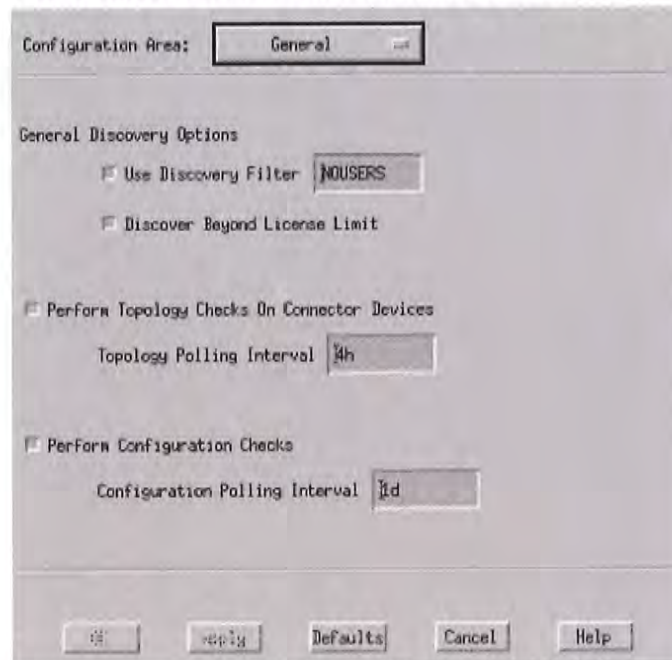
Before you get sick looking at your newly discovered network, keep in mind that you can add some quick and easy customizations that will transform your hodgepodge of names, numbers, and icons into a coordinated picture of your network.

6.1.2 The netmon Process

NNM's daemon process (netmon) starts automatically when the system boots and is responsible for discovering nodes on your network, in addition to a few other tasks. In NNM's menu, go to

"Options → Network Polling Configurations: IP." A window should appear that looks similar to [Figure 6-3](#).

Figure 6-3. OpenView's General network polling configuration options



[Figure 6-3](#) shows the General area of the configuration wizard. The other areas are IP Discovery, Status Polling, and Secondary Failures. The General area allows us to specify a filter (in this example, **NOUSERS**) that controls the discovery process -- we might not want to see every device on the network. We discuss the creation of filters later in this chapter, in [Section 6.1.5](#). We elected to discover beyond the license limit, which means that NNM will discover more objects on our network than our license allows us to manage. "Excess" objects (objects past the license's limit) are placed in an unmanaged state, so that you can see them on your maps but can't control them through NNM. This option is useful when your license limits you to a specific number of managed nodes.

The IP Discovery area ([Figure 6-4](#)) lets us enable or disable the discovery of IP nodes. Using the "auto adjust" discovery feature allows NNM to figure out how often to probe the network for new devices. The more new devices it finds, the more often it polls; if it doesn't find any new devices it slows down, eventually waiting one day (1d) before checking for any new devices. If you don't like the idea that the discovery interval varies (or perhaps more realistically, if you think that probing the network to find new devices will consume more resources than you like, either on your network-management station or the network itself), you can specify a fixed discovery interval. Finally, the "Discover Level-2 Objects" button tells NNM to discover and report devices that are at the second layer of the OSI network model. This category includes

things such as unmanaged hubs and switches, many AppleTalk devices, and so on.

Figure 6-4. OpenView's IP Discovery network polling configuration options

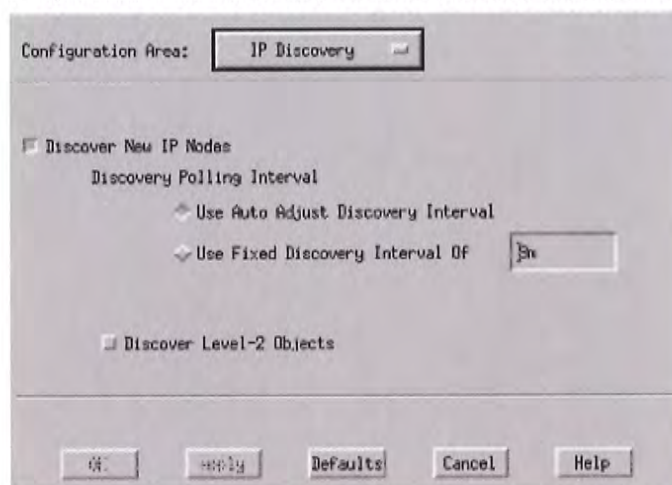
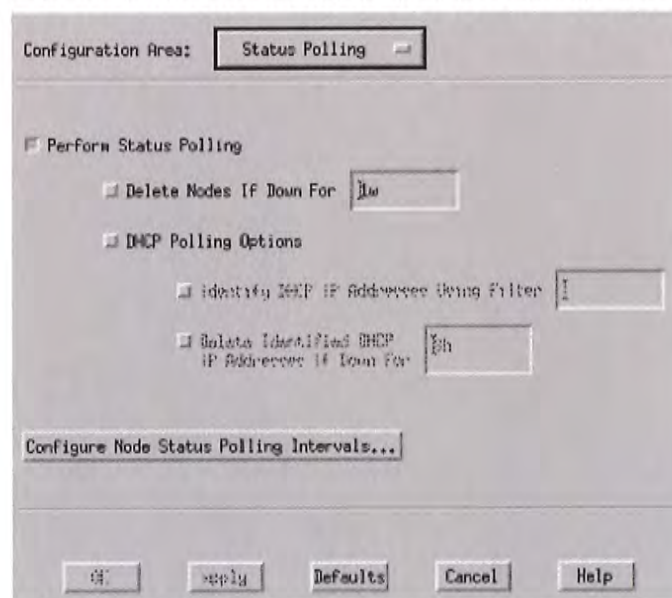


Figure 6-5 shows the Status Polling configuration area. Here you can turn status polling on or off, and delete nodes that have been down or unreachable for a specified length of time. This example is configured to delete nodes after they've been down for one week (1w).

Figure 6-5. OpenView's Status Polling network polling configuration options

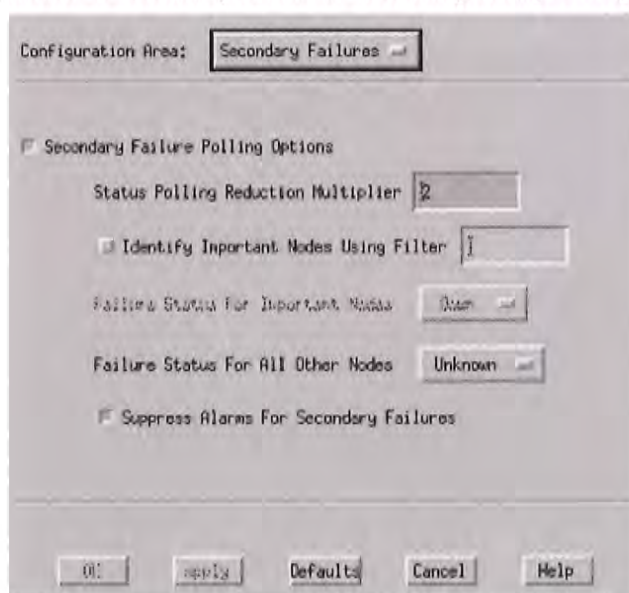


The DHCP polling options are, obviously, especially useful in environments that use DHCP. They allow you to establish a relationship between polling behavior and IP addresses. You can specify a filter that selects addresses that are assigned by DHCP. Then you can specify a time after which netmon will delete non-responding DHCP addresses from its map of your network. If a device is down for the given amount of time, netmon disassociates the node and IP address. The rationale for

this behavior is simple: in a DHCP environment, the disappearance of an IP address often means that the node has received a new IP address from a DHCP server. In that case, continuing to poll the old address is a waste of effort and is possibly even misleading, since the address may be reassigned to a different host.

Finally, the Secondary Failures configuration area shown in [Figure 6-6](#) allows you to tell the poller how to react when it sees a secondary failure. This occurs when a node beyond a failed device is unreachable; for example, when a router goes down, making the file server that is connected via one of the router's interfaces unreachable. In this configuration area, you can state whether to show alarms for the secondary failures or suppress them. If you choose to suppress them, you can set up a filter that identifies important nodes in your network that won't get suppressed even if they are deemed secondary failures.

Figure 6-6. OpenView's Secondary Failures network polling configuration options



Once your map is up, you may notice that nothing is getting discovered. Initially, netmon won't discover anything beyond the network segment to which your NMS is attached. If your NMS has an IP address of 24.92.32.12, you will not discover your devices on 123.67.34.0. NNM finds adjacent routers and their segments, as long as they are SNMP-compatible, and places them in an unmanaged (tan colored) state on the map.^[3] This means that anything in and under that icon will not be polled or discovered. Selecting the icon and going to "Edit → Manage Objects" tells NNM to begin managing this network and allows netmon to start discovering nodes within it. You can quit managing nodes at any time by clicking on UnManage instead of Manage.

^[a] In NNM, go to "Help → Display Legend" for a list of icons and their colors.

If your routers do not show any adjacent networks, you should try testing them with "Fault → Test IP/TCP/SNMP." Add the name of your router, click "Restart," and see what kind of results you get back. If you get "OK except for SNMP," review [Chapter 7](#) and read [Section 6.1.3](#), on setting up the default community names within OpenView.

netmon also allows you to specify a seed file that helps it to discover objects faster. The seed file contains individual IP addresses, IP address ranges, or domain names that narrow the scope of hosts that are discovered. You can create the seed file with any text editor -- just put one address or hostname on each line. Placing the addresses of your gateways in the seed file sometimes makes the most sense, since gateways maintain ARP tables for your network. netmon will subsequently discover all the other nodes on your network, thus freeing you from having to add all your hosts to the seed file. For more useful information, see the documentation for the -s switch to netmon and the Local Registration Files (LRF).

NNM has another utility, called loadhosts, that lets you add nodes to the map one at a time. Here is an example of how you can add hosts, in a sort of freeform mode, to the OpenView map. Note the use of the -m option, which sets the subnet to 255.255.255.0:

```
$ loadhosts -m 255.255.255.0
10.1.1.12 gwrouter1
```

Once you have finished adding as many nodes as you'd like, press Ctrl-d to exit the command.

6.1.3 Configuring Polling Intervals

The SNMP Configuration page is located off of the main screen in "Options → SNMP Configuration." A window similar to the one in [Figure 6-7](#) should appear. This window has four sections: Specific Nodes, IP Address Wildcards, Default, and the entry area (chopped off for viewing purposes). Each section contains the same general areas: Node or IP Address, Get Community, Set Community, Proxy (if any), Timeout, Retry, Port, and Polling. The Default area, which unfortunately is at the bottom of the screen, sets up the default behavior for SNMP on your network -- that is, the behavior (community strings, etc.) for all hosts that aren't listed as "specific nodes" or that match one of the wildcards. The Specific Nodes section allows you to specify exceptions, on a per node basis. IP Address Wildcards allows you to configure properties for a range of addresses. This is especially useful if you have networks that have different get and set community names.^[4] All areas allow you to specify a Timeout in seconds and a Retry value. The Port field gives you

the option of inserting a different port number (the default port is 161). Polling is the frequency at which you would like to poll your nodes.

[4] These community names are used in different parts throughout NNM. For example, when polling an object with *xnmbrowser*, you won't need to enter (or remember) the community string if it (or its network) is defined in the SNMP configurations.

Figure 6-7. OpenView's SNMP Configuration page

SNMP Configuration for net							
Specific Nodes							
Node	Get Community	Set Community	Proxy	Timeout	Retry	Port	Polling
208.166.230.1	netopublic	-	<none>	0.9	2	-	3s
IP Address Wildcards							
IP Wildcard	Get Community	Set Community	Proxy	Timeout	Retry	Port	Polling
203.126.4.*	public	-	<none>	3.0	3	-	6s
202.109.230.*	public	-	<none>	3.0	3	-	6s
Default							
Default	Get Community	Set Community	Proxy	Timeout	Retry	Port	Polling
Global Default	public	-	<none>	0.8	2	-	4.30s

It's important to understand how timeouts and retries work. If we look at Specific Nodes, we see a Timeout of .9 seconds and a Retry of 2 for 208.166.230.1. If OpenView doesn't get a response within .9 seconds, it tries again (the first retry) and waits 1.8 seconds. If it still doesn't get anything back, it doubles the timeout period again to 3.6 seconds (the second retry); if it still doesn't get anything back it declares the node unreachable and paints it red on the NNM's map. With these Timeout and Retry values, it takes about 6 seconds to identify an unreachable node.

Imagine what would happen if we had a Timeout of 4 seconds and a Retry of 5. By the fifth try we would be waiting 128 seconds, and the total process would take 252 seconds. That's over four minutes! For a mission-critical device, four minutes can be a long time for a failure to go unnoticed.

This example shows that you must be very careful about your Timeout and Retry settings -- particularly in the Default area, because these settings apply to most of your network. Setting your Timeout and Retry too high and your Polling periods too low will make netmon fall behind; it will be time to start over before the poller has worked through all your devices.^[5] This is a frequent problem when you have many nodes, slow networks, small polling times, and high numbers for Timeout and Retry.^[6] Once a system falls behind, it will take a long time to discover problems with the devices it is currently monitoring, as well as to discover new devices. In some cases, NNM may not discover problems with downed devices at all! If your Timeout

and Retry values are set inappropriately, you won't be able to find problems and will be unable to respond to outages.

^[5] Keep in mind that most of NNM's map is polled using regular pings and not SNMP.

^[6] Check the manpage for netmon for the `-a` switch, especially around `-a12`. You can try to execute netmon with an `-a | ?`, which will list all the valid `-a` options. If you see any negative numbers in `netmon.trace` after running `netmon -a12`, your system is running behind.

Falling behind can be very frustrating. We recommend starting your Polling period very high and working your way down until you feel comfortable. Ten to twenty minutes is a good starting point for the Polling period. During your initial testing phase, you can always set a wildcard range for your test servers, etc.

6.1.4 A Few Words About NNM Map Colors

By now discovery should be taking place, and you should be starting to see some new objects appear on your map. You should see a correlation between the colors of these objects and the colors in NNM's Event Categories (see [Chapter 10](#) for more about Event Categories). If a device is reachable via *ping*, its color will be green. If the device cannot be reached, it will turn red. If something "underneath" the device fails, the device will become off-green, indicating that the device itself is okay, but something underneath it has reached a nonnormal status. For example, a router may be working, but a web server on the LAN behind it may have failed. The status source for an object like this is Compound or Propagated. (The other types of status source are Symbol and Object.) The Compound status source is a great way to see if there is a problem at a lower level while still keeping an eye on the big picture. It alerts you to the problem and allows you to start drilling down until you reach the object that is under duress.

It's always fun to shut off or unplug a machine and watch its icon turn red on the map. This can be a great way to demonstrate the value of the new management system to your boss. You can also learn how to cheat and make OpenView miss a device, even though it was unplugged. With a relatively long polling interval, it's easy to unplug a device and plug it back in before OpenView has a chance to notice that the device isn't there. By the time OpenView gets around to it, the node is back up and looks fine. Long polling intervals make it easy to miss such temporary failures. Lower polling intervals make it less likely that OpenView will miss something, but more likely that netmon will fall behind, and in turn miss other failures. Take

small steps so as not to crash or overload netmon or your network.

6.1.5 Using OpenView Filters

Your map may include some devices you don't need, want, or care about. For example, you may not want to poll or manage users' PCs, particularly if you have many users and a limited license. It may be worthwhile for you to ignore these user devices to open more slots for managing servers, routers, switches, and other more important devices. netmon has a filtering mechanism that allows you to control precisely which devices you manage. It lets you filter out unwanted devices, cleans up your maps, and can reduce the amount of management traffic on your network.

In this book, we warn you repeatedly that polling your network the wrong way can generate huge amounts of management traffic. This happens when people or programs use default polling intervals that are too fast for the network or the devices on the network to handle. For example, a management system might poll every node in your 10.1.0.0 network -- conceivably thousands of them -- every two minutes. The poll may consist of SNMP *get* or *set* requests, simple *pings*, or both. OpenView's NNM uses a combination of these to determine if a node is up and running. Filtering saves you (and your management) the trouble of having to pick through a lot of useless nodes and reduces the load on your network. Using a filter allows you to keep the critical nodes on your network in view. It allows you to poll the devices you care about and ignore the devices you don't care about. The last thing you want is to receive notification each time a user turns off his PC when he leaves for the night.

Filters also help network management by letting you exclude DHCP users from network discovery and polling. DHCP and BOOTP are used in many environments to manage large IP address pools. While these protocols are useful, they can make network management a nightmare, since it's often hard to figure out what's going on when addresses are being assigned, deallocated, and recycled.

In my environment we use DHCP only for our users. All servers and printers have hardcoded IP addresses. With our setup, we can specify all the DHCP clients and then state that we want everything but these clients in our discovery, maps, etc. The following example should get most users up and running with some pretty good filtering. Take some time to review OpenView's "A Guide to Scalability and Distribution for Network Node Manager" manual for more in-depth information on filtering.

The default filter file, which is located in `$OV_CONF/C`, is broken up into three sections:

- Sets

- Filters
- FilterExpressions

In addition, lines that begin with `//` are comments. `//` comments can appear anywhere; some of the other statements have their own comment fields built in.

Sets allow you to place individual nodes into a group. This can be useful if you want to separate users based on their geographic locations, for example. You can then use these groups or any combination of IP addresses to specify your Filters, which are also grouped by name. You then can take all of these groupings and combine them into FilterExpressions. If this seems a bit confusing, it is! Filters can be very confusing, especially when you add complex syntax and not so logical logic (`&&`, `||`, etc.). The basic syntax for defining Sets, Filters, and FilterExpressions looks like this:

name "comments or description" { contents }

Every definition contains a name, followed by comments that appear in double quotes, and then the command surrounded by brackets. Our default filter,^[7] named `filters`, is located in `$OV_CONF/C` and looks like this:

^[7] Your filter, if right out of the box, will look much different. The one shown here is trimmed to ease the pains of writing a filter.

```
// lines that begin with // are considered COMMENTS and are ignored!
// Begin of MyCompanyName Filters
```

```
Sets {
```

```
    dialupusers "DialUp Users" { "dialup100", " dialup101", \
                                " dialup102" }
```

```
}
```

```
Filters {
```

```
    ALLIPRouters "All IP Routers" { isRouter }
```

```
    SinatraUsers "All Users in the Sinatra Plant" { \
        ("IP Address" ~ 199.127.4.50-254) || \
        ("IP Address" ~ 199.127.5.50-254) || \
        ("IP Address" ~ 199.127.6.50-254) }
```

```
    MarketUsers "All Users in the Market Plant" { \
        ("IP Address" ~ 172.247.63.17-42) }
```

```
    DialAccess "All DialAccess Users" { "IP Hostname" in dialupusers }
}
```

```
FilterExpressions
{
```

```

    ALLUSERS "All Users" { SinatraUsers || MarkeUsers || DialAccess }

    NOUSERS "No Users " { !ALLUSERS }
}

```

Now let's break this file down into pieces to see what it does.

6.1.5.1 Sets

First, we defined a Set^[8] called `dialupusers` containing the hostnames (from DNS) that our dial-up users will receive when they dial into our facility. These are perfect examples of things we don't want to manage or monitor in our OpenView environment.

^[8] These Sets have nothing to do with the `snmpset` operation with which we have become familiar.

6.1.5.2 Filters

The Filters section is the only nonoptional section. We defined four filters: `ALLIPRouters`, `SinatraUsers`, `MarkeUsers`, and `DialAccess`. The first filter says to discover nodes that have field value `isRouter`. OpenView can set the object attribute for a managed device to values such as `isRouter`, `isHub`, `isNode`, etc.^[9] These attributes can be used in Filter expressions to make it easier to filter on groups of managed objects, as opposed to IP address ranges, for example.

^[9] Check out the `$OV_FIELDS` area for a list of fields.

The next two filters specify IP address ranges. The `SinatraUsers` filter is the more complex of the two. In it, we specify three IP address ranges, each separated by logical OR symbols (`||`). The first range (`("IP Address" ~ 199.127.6.50-254)`) says that if the IP address is in the range 199.127.6.50-199.127.6.254, then filter it and ignore it. If it's not in this range, the filter looks at the next range to see if it's in that one. If it's not, the filter looks at the final IP range. If the IP address isn't in any of the three ranges, the filter allows it to be discovered and subsequently managed by NNM. Other logical operators should be familiar to most programmers: `&&` represents a logical AND, and `!` represents a logical NOT.

The final filter, `DialAccess`, allows us to exclude all systems that have a hostname listed in the `dialupusers` set, which was defined at the beginning of the file.

6.1.5.3 FilterExpressions

The next section, `FilterExpressions`, allows us to combine the filters we have previously defined with additional logic. You can use a `FilterExpression` anywhere you would use a `Filter`. Think of it like this: you create complex expressions using `Filters`, which in turn can use `Sets` in the contents parts of

their expressions. You can then use FilterExpressions to create simpler yet more robust expressions. In our case, we take all the filters from above and place them into a FilterExpression called ALLUSERS. Since we want our NNM map to contain nonuser devices, we then define a group called NOUSERS and tell it to ignore all user-type devices with the command !ALLUSERS. As you can see, FilterExpressions can also aid in making things more readable. When you have finished setting up your filter file, use the \$OV_BIN/ovfiltercheck program to check your new filters' syntax. If there are any problems, it will let you know so you can fix them.

Now that we have our filters defined, we can apply them by using the ovtopofix command or the polling configuration menu shown in [Figure 6-3](#).

If you want to remove nodes from your map, use \$OV_BIN/ovtopofix -f FILTER_NAME. Let's say that someone created a new DHCP scope without telling you and suddenly all the new users are now on the map. You can edit the filters file, create a new group with the IP address range of the new DHCP scope, add it to the ALLUSERS FilterExpression, run ovfiltercheck, and, if there are no errors, run \$OV_BIN/ovtopofix -f NOUSERS to update the map on the fly. Then stop and restart netmon -- otherwise it will keep discovering these unwanted nodes using the old filter. I find myself running ovtopofix every month or so to take out some random nodes.

6.1.6 Loading MIBs into OpenView

Before you continue exploring OpenView's NNM, take time to load some vendor-specific MIBs.^[10] This will help you later on when you start interacting (polling, graphing, etc.) more with SNMP-compatible devices. Go to "Options → Load/Unload MIBs: SNMP." This presents you with a window in which you can add vendor-specific MIBs to your database. Alternatively, you can run the command \$OV_BIN/xnmloadmib and bypass having to go through NNM directly.

^[10] Some platforms and environments refer to loading a MIB as compiling it.

That's the end of our brief tour of OpenView configuration. It's impossible to provide a complete introduction to configuring OpenView in this chapter, so we tried to provide a survey of the most important aspects of getting it running. There can be no substitute for the documentation and manual pages that come with the product itself.

6.2 Castle Rock's SNMPc Enterprise Edition

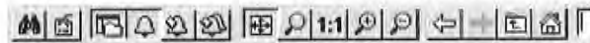
We'll end the chapter with a brief discussion of Castle Rock's SNMPc, Version 5.0, which runs on Windows NT/2000. SNMPc is a simpler product than OpenView in many respects. However, even though it's simpler, it's far from featureless. It's also cheaper than OpenView, which makes it ideal for shops that don't have a lot of money to spend on an NMS platform but need the support and backing that a commercial product provides.

Installation of SNMPc is straightforward. The installer asks for the license number and a discovery seed device. The seed device is similar to a seed file for OpenView's netmon. In the case of SNMPc, we recommend giving it the IP address (or hostname) of your gateway, since this device can be used to discover other segments of your network. Omitting the discovery seed device will not keep SNMPc from performing discovery but will limit it to the devices on the network to which it's directly connected.

6.2.1 SNMPc's Map

Once SNMPc is up and running, you will see any devices it has discovered in the Root map view. [Figure 6-8](#) shows the main button bar. The far right button (the house) gets you to the highest level on the map. The zooming tools allow you to pan in and out of the map, increasing or decreasing the amount of detail it shows. You can also reach the Root submap by selecting "Map View → Root submap" from the View menu.

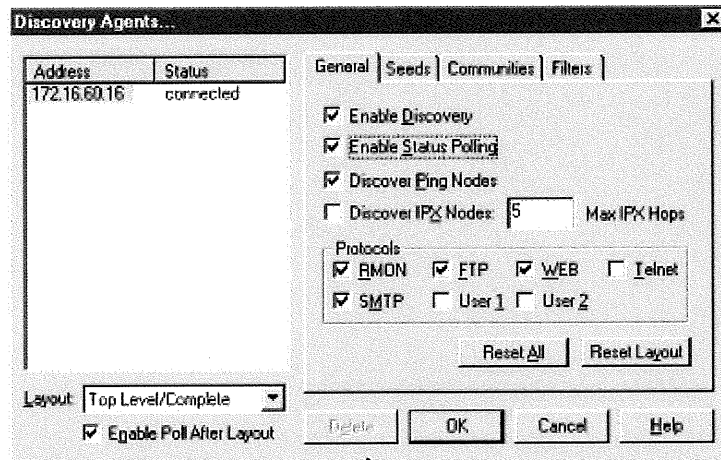
Figure 6-8. SNMPc main button bar



6.2.2 Discovery and Filters

Once you are done playing around with your maps, it's time to start tuning your polling parameters. Go to "Config → Discovery Agents." This should bring up a menu that looks like [Figure 6-9](#). Looking at the menu tabs, it's easy to tell that you will be able to configure your Seeds, Communities, and Filters here. SNMPc filters are equivalent to OpenView filters, but much simpler.

Figure 6-9. SNMPc Discovery Agents menu



The General tab lets you control SNMPc's polling and discovery behavior. The checkbox for enabling and disabling discovery is self-explanatory. The "Enable Status Polling" checkbox determines if SNMPc will *ping* the nodes on your network periodically to determine whether or not they are responding. By default, all nodes are polled every 10 to 30 seconds. To change these default values, you can either edit the properties of each device (one by one), select and highlight multiple devices (using your Ctrl key), or use the object selection tool. You can bring up this tool by using the third button from the left on the main button bar or by going to "View → Selection Tool." The "Discover Ping Nodes" checkbox lets you specify if you want to discover devices that have an IP or IPX entity but do not have an SNMP agent. "Discover IPX Nodes" gives you the option of discovering IPX devices. SNMPc will also check if a device supports various protocols such as SMTP, HTTP, etc. This feature allows you to set up custom menu items based on what services the device is running. The Protocols section of the General tab lets you specify the protocols for which SNMPc will test.

The Seeds tab allows you to specify SNMP devices that will help the discovery process along. This tab allows you to specify more than one seed IP address. (Remember that you're asked for a seed address device when you install the product.)

The Communities tab lets you specify the community strings for your network. You can specify multiple community names; SNMPc will try the different community names when discovering your nodes. Once SNMPc figures out which community is correct for a given device, it inserts the community string in the "Get Community" attribute for that particular device. This simply means the newly discovered device will be saved with its community string.

The final tab, Filters, allows you to exclude certain IP addresses from being discovered. You can specify individual

addresses, or use an asterisk (*) as a wildcard to specify entire networks.

6.2.3 Loading MIBs into SNMPc

Like any reasonably comprehensive network-management product, SNMPc can load and compile new MIBs. To do so, select "Config → MIB Database" from the main menu bar. This window lets you specify the path to the MIB file and gives you full feedback about the status of the compilation, etc. Click on the "Help" button for more information about MIB compilation.

SNMPc is a compact NMS that provides some added features, such as trend reporting. A thorough treatment of its installation is beyond the scope of this book. The online help system that comes with SNMPc is very good, and we recommend you take full advantage of it.

Chapter 7. Configuring SNMP Agents

By this time you should understand what an SNMP agent is: it's nothing more than software that lives on the device you want to monitor. It responds to requests from the NMS and generates traps. This chapter discusses how to configure agents. It starts by defining some standard configuration parameters that are common to all SNMP agents, then goes into some advanced parameters you might run into when configuring your equipment. The bulk of this chapter walks through the configuration for a number of common devices, paying attention to security issues.

7.1 Parameter Settings

All SNMP devices share the following common configurable parameters:

- sysLocation
- sysContact
- sysName
- Read-write and read-only access community strings (and frequently, a trap community string)
- Trap destination

sysLocation is the physical location for the device being monitored. Its definition in RFC 1213 is:

sysLocation OBJECT-TYPE

SYNTAX DisplayString (SIZE (0..255))

ACCESS read-write

STATUS mandatory

DESCRIPTION

"The physical location of this node (e.g., 'telephone closet,

```
    3rd floor')."  
 ::= { system 6 }
```

As you can see, its SYNTAX is DisplayString, which means it can be an ASCII string of characters; its size is declared to be at most 255 characters. This particular object is useful for determining where a device is located. This kind of practical information is essential in a large network, particularly if it's spread over a wide area. If you have a misbehaving switch, it's very convenient to be able to look up the switch's physical location. Unfortunately, sysLocation frequently isn't set when the device is installed and even more often isn't changed when the device is moved. Unreliable information is worse than no information, so use some discipline and keep your devices up to date.

RFC 1213's definition of sysContact is similar to that of sysLocation:

```
sysContact OBJECT-TYPE  
    SYNTAX  DisplayString (SIZE (0..255))  
    ACCESS  read-write  
    STATUS  mandatory  
    DESCRIPTION  
        "The textual identification of the contact person for this managed  
        node, together with information on how to contact this person."  
 ::= { system 4 }
```

sysContact is a DisplayString. It's fairly obvious what it's used for: it identifies the primary contact for the device in question. It is important to set this object with an appropriate value, as it can help your operations staff determine who needs to be contacted in the event of some catastrophic failure. You can also use it to make sure you're notified, if you're responsible for a given device, when someone needs to take your device down for maintenance or repairs. As with sysLocation, make sure to keep this information up to date as your staff changes. It's not uncommon to find devices for which the sysContact is someone who left the company several years ago.

sysName should be set to the *fully-qualified domain name* (FQDN) for the managed device. In other words, it's the hostname associated with the managed device's IP address. The RFC 1213 definition follows:

```
sysName OBJECT-TYPE  
    SYNTAX  DisplayString (SIZE (0..255))  
    ACCESS  read-write  
    STATUS  mandatory  
    DESCRIPTION  
        "An administratively-assigned name for this managed node. By  
        convention, this is the node's fully-qualified domain name."  
 ::= { system 5 }
```

The read-only and read-write parameters are the community strings for read-only and read-write access. Notice that

sysLocation, sysContact, and sysName all have ACCESS values of read-write. With the appropriate read-write community string, anyone can change the definition of these objects and many more objects of significantly greater importance. Ultimately, it's not a huge problem if somebody maliciously makes your router lie about its location -- you probably already know that it isn't located in Antarctica. But someone who can do this can also fiddle with your routing tables and do other kinds of much more serious damage. Someone who has only the read-only community string can certainly find out more information about your network than you would like to reveal to an outsider. Setting the community strings is extremely important to maintaining a secure environment. Most devices are shipped with default community strings that are well known. Don't assume that you can put off setting your community strings until later.

The trap destination parameters specify the addresses to which traps are sent. There's nothing really magical here -- since traps are asynchronous notifications generated by your devices, the agent needs to know who should receive notification. Many devices support authentication-failure traps, which are generated if someone attempts to access them using incorrect community strings. This feature is extremely useful, as it allows you to detect attempts to break into your devices. Many devices also support the ability to include a community string with traps; you can configure the network-management station to respond only to traps that contain the proper community string.

Many devices have additional twists on the access and trap parameters. For example, Cisco devices allow you to create different community strings for different parts of the MIB -- you can use this to allow people to set some variables, but not others. Many vendors allow you to place restrictions on the hosts that are allowed to make SNMP requests. That is, the device will respond only to requests from certain IP addresses, regardless of the community string.

The range of configuration options you're likely to run into is limited only by the imagination of the vendors, so it's obviously impossible for us to describe everything you might encounter. [Section 7.3](#) later in this chapter will give you an idea of how some agents implement the standard configuration parameters and a little insight into what other features might be available.

7.2 Security Concerns

[Chapter 2](#) discussed the security issues with SNMPv1 and SNMPv2. The biggest problem, of course, is that the read-only and read-write community strings are sent as clear-text strings; the agent or the NMS performs no encryption. Therefore, the

community strings are available to anyone with access to a packet sniffer. That certainly means almost anyone on your network with a PC and the ability to download widely available software. Does that make you uncomfortable? It should.

Obviously, you need to take the same precautions with the community strings that you would with your superuser or administrator passwords. Choose community strings that are hard to guess. Mixed-case alphanumeric strings are good choices for community strings; don't use dictionary words. Although someone with the read-only community string can't do as much damage as someone with the read-write string, you might as well take the same precautions for both. Don't forget to change your community strings -- most devices ship with preconfigured community strings that are extremely easy to guess.

That doesn't solve the problems with packet sniffers. When you're configuring an agent, it's a good idea to limit the devices that can make SNMP requests (assuming that your agent allows you to make this restriction). That way, even if someone gets the community strings, he'll have to spoof the IP address of one of your management stations to do any damage.

Of course, many people know how to spoof IP addresses these days, and it's not a really good idea to assume that you can trust your employees. A better solution to the problem is to prevent the SNMP packets from being visible on your external network connections and parts of your network where you don't want them to appear. This requires configuring your routers and firewalls with access lists that block SNMP packets from the outside world (which may include parts of your own network). If you don't trust the users of your network, you may want to set up a separate administrative network to be used for SNMP queries and other management operations. This is expensive and inflexible -- it's hard to imagine extending such a network beyond your core routers and servers -- but it may be what your situation requires.

If you want to use SNMP to monitor your network from home, be extremely careful. You do not want your community strings traveling over the public Internet in an unencrypted form. If you plan to use SNMP tools directly from home, make sure to install VPN software, or some form of tunneling, to keep your SNMP traffic private. A better approach to home monitoring is to use a web interface; by using SSL, you can prevent others from seeing your usage graphs. (No network-management products that we're aware of support SSL out of the box; but they do allow you to integrate with external servers, such as Apache, which do support SSL).

SNMPv3 (discussed in [Appendix F](#)) fixes most of the security problems; in particular, it makes sure that the community strings are always encrypted. Unfortunately, there are very few

implementations of SNMPv3 out there. It's clear what direction you want to head in, but you can't get there yet.

7.3 Agent Configuration Walkthroughs

In the following sections we will walk through the configurations of some typical SNMP agents. We have chosen devices that are found on almost every modern network (x86 PCs, Unix Servers, routers, UPSs, etc.). The point of this discussion isn't to show you how your particular agent is configured -- that would not be practical, given the hundreds of devices and vendors out there. Our intent is to give you a feel for what the common options are, and what steps you'll typically go through to configure an agent.

7.3.1 Windows 95/98 Agent

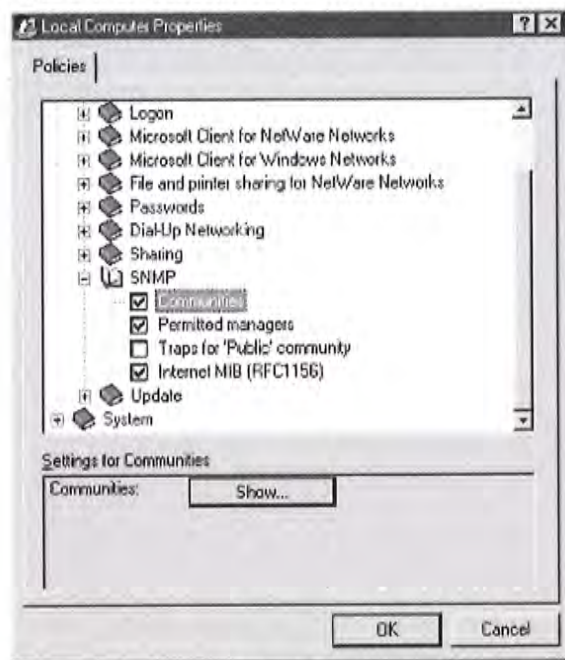
In this section, we'll walk through the SNMP configuration for the Windows 95/98 agent, using the Windows System Policy Editor. The settings are all stored in the registry, so you can also make changes to the configuration using *regedit*, but there's less chance of error if you use the System Policy Editor. It's worth noting that Windows 95, 98, and NT all have the same SNMP entries in the registry, so configuration for these operating systems is similar. It's also worth noting that Microsoft's SNMP agent isn't terribly robust, although it's adequate if you want only basic SNMP functionality. Other agents are available; Concord's SystemEDGE and Castle Rock's SNMPC support the Microsoft operating systems.



Unless you are completely comfortable taking the registry editing leap, we strongly recommend that you use the System Policy Editor to make agent configuration changes. Incorrect settings in the registry can result in serious system problems. Consider yourself warned.

The Windows System Policy Editor comes with the Windows 95/98 Resource Kit, and must be installed before you can configure the SNMP agent. The first time you run the System Policy Editor it will ask you for an *.adm* file. Select *C:\WINDOWS\INF\ADMIN.ADM* as this file. Select "File → Open Registry," then double-click the Local Computer icon. In the Policies tab, click down the plus signs until you reach Network and then SNMP. This should leave you with four SNMP agent configuration items. [Figure 7-1](#) shows what your window should look like. To enable an option, place a check next to it. When you are finished, click "OK," then "File → Save" at the main screen. If you don't follow these steps, your configuration won't be saved to the registry.

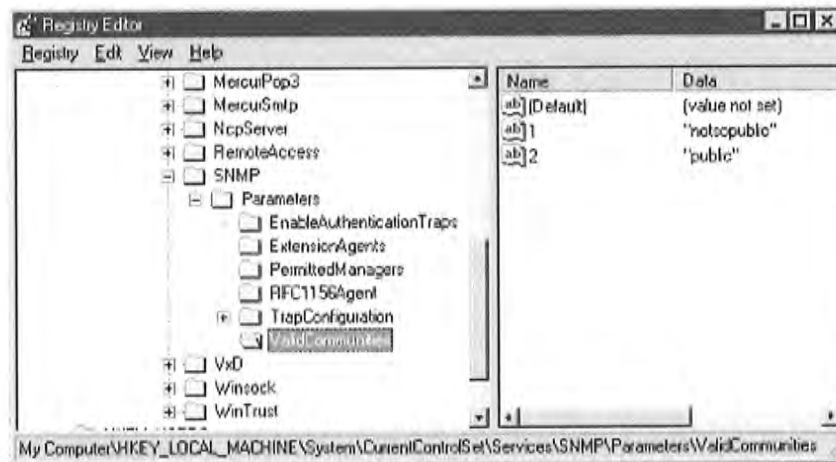
Figure 7-1. Windows 95/98 System Policy Editor



The "Communities" settings allow you to define your community strings. Check the box and then click "Show" in the lower section. This brings up another window showing your community strings. To create a new community, click "Add" and then enter the string. Repeat the steps, if appropriate, for your site. If this option is left unchecked, or if it is checked but no community names are listed, the agent will answer all SNMP requests it receives. The next checkbox item, "Permitted managers," specifies what NMSs can access this agent. You can identify your management stations by IPX addresses, IP addresses, or DNS names. For example, you can use this item to restrict SNMP access to a particular NMS. If the "Permitted managers" box is unchecked or is checked but has no entries, the agent will answer all requests, no matter where they come from. Checking "Traps for 'Public' community" allows you to designate up to five NMSs to receive traps. The last setting, "Internet MIB (RFC1156)," allows you to set the Contact Name (sysContact) and Location (sysLocation) objects.

Remember to save your changes using "File → Save" at the main menu of the System Policy Editor. [Figure 7-2](#) shows what the entries look like in the Registry Editor, after you've used the Policy Editor to set them.

Figure 7-2. Windows 95/98 Registry Editor



7.3.2 Windows NT 4.0 and Windows 2000 Agent

To configure the SNMP service in Windows NT 4.0 and 2000, start in the Control Panel and double-click on the Network icon. Click on the Services tab, select "SNMP Service," and click on the "Properties" button. If "SNMP Service" isn't listed, you need to add it. Press the "Add" button and select "SNMP Service" from the list of services. It will prompt you for your Windows NT system disk, so be sure to have it ready. For Windows 2000, go to the Control Panel and click on "Add/Remove Programs." When the window pops up click on "Add/Remove Windows Components," then select "Management and Monitoring Tools." This should bring up a window with one item in it, "Simple Network Management Protocol." Check the box next to it and press "OK." This will take you back to the Components Wizard window. Click "Next" to begin the installation of the SNMP service. You will probably need your Windows 2000 CD-ROM.

Once you have installed the SNMP service or selected it from the list of installed services, a new window should appear. This window is broken up into three tabs: Agent, Traps, and Security. In the Agent tab, you should configure the Contact (sysContact), Location (sysLocation), and Service (sysServices). We haven't mentioned the sysServices object yet; RFC 1213 defines it like this:

sysServices OBJECT-TYPE

SYNTAX INTEGER (0..127)

ACCESS read-only

STATUS mandatory

DESCRIPTION

"A value which indicates the set of services that this entity primarily offers.

The value is a sum. This sum initially takes the value zero. Then, for each layer, L, in the range 1 through 7, that this node performs transactions for, 2 raised to (L - 1) is added to the sum. For example, a node which performs primarily routing functions would have a value of 4 ($2^{(3-1)}$). In contrast, a node which is a

host offering application services would have a value of 72 ($2^{(4-1)} + 2^{(7-1)}$). Note that in the context of the Internet suite of protocols, values should be calculated accordingly:

layer	functionality
1	physical (e.g., repeaters)
2	datalink/subnetwork (e.g., bridges)
3	internet (e.g., IP gateways)
4	end-to-end (e.g., IP hosts)
7	applications (e.g., mail relays)

For systems including OSI protocols, layers 5 and 6 may also be counted."

::= { system 7 }

The Agent tab provides a checkbox for each of the seven ISO layers sysServices represents. The DESCRIPTION text in the RFC gives a brief definition for each layer. If you so desire, check each service that is offered by your NT machine.

Once you're finished with the Agent tab, select the Traps tab; this allows you to configure the community in which the SNMP agent sends traps. In the "Community Name" box, enter the case-sensitive community name of your choice. Click the "Add" button to the left and then add up to five trap destinations for this community name. The trap destinations can be IPX addresses, IP addresses, or DNS names.

Now click on the Security tab. The top of this tab gives you the option to send authentication-error traps. It's a good idea to check this box, since it can help you detect intruders. The "Accepted Community Names" box lists all the community names to which the agent will respond. Click "Add" and enter your community name of choice. Configuring these communities is important, since someone with the correct community string can wreak havoc on your system. If you leave this box blank, the agent will respond to all requests. The bottom half of the Security menu allows you to specify whether the agent will accept SNMP packets from any host or only from a specified list. To create a list, which we strongly recommend, click "Only Accept SNMP Packets from These Hosts" and then use the "Add" button to add the hostnames or addresses of your monitoring stations. The options for the hosts are the same as for trap destinations; IPX addresses, IP addresses, and DNS names are acceptable.

Finally, click "OK" to save your changes and update the Windows registry. If at any time you make a mistake, click "Cancel." This aborts the configuration process; no changes will be made to your registry.

7.3.3 HP OpenView Agent for HP-UX and Solaris

One text-configuration file controls the parameters for this agent; the file is typically named `/etc/SnmpAgent.d/snmpd.conf`, or `/etc/snmpd.conf` on older systems. You don't need to edit this file for the agent to function normally. If you do edit it, you must stop and restart the master agent by executing the `SnmpMaster` script, first with a `stop` and then a `start` :

```
$ /sbin/init.d/SnmpMaster stop
$ /sbin/init.d/SnmpMaster start
```

7.3.3.1 Simple configuration

The following configuration file configures the agent to respond to `get` requests using the community name `public` and `set` requests using the community name `private`. There are no restrictions on which MIBs can be queried, or which hosts can make the queries. This configuration has no security, since the community strings are set to commonly used defaults and are widely known. The OpenView agent sends authentication-failure traps by default, so you don't have to enable these traps in the configuration file.

```
get-community-name: public
set-community-name: private
trap-dest:         127.0.0.1
contact:           B.Gates
location:          12 Pyramid - Egypt
```

The simplest configuration is to edit the file and place more reasonable community names in the first two lines. We can't say it too much: community names are essentially passwords. Use the same rules for picking community names that you would for choosing the root password. You should always set the destination trap host (`trap-dest`) to the IP address of the host that will receive the trap.

The next example configures several different community names:

```
get-community-name: public
get-community-name: media
set-community-name: hushed
set-community-name: veryprivate
set-community-name: shhhh
```

We have created two `get` (read-only) communities and three `set` (read-write) communities. These communities can be used as you see fit. (In real life, we would have chosen more obscure names.) For example, you might give your operations group in New York public community access and your operations group in Atlanta media community access. The remaining `set` communities can further be subdivided among various administrators and other staff who need read-write access.

7.3.3.2 Advanced configuration

Setting up multiple community strings doesn't sound very useful, and by itself, it isn't. But you can take the concept a

step further and create different communities, each of which consists of a few particular hosts and can access only some of the objects SNMP manages. The next example allows the host 10.123.56.25 to issue *gets* using the community name *comname* and *sets* using the community name *private*. The host 10.123.46.101 can issue *gets* using only the community name *comname*. You cannot use hostnames after the *IP:* directive; you must use IP addresses.

```
get-community-name    comname IP: 10.123.56.25 10.123.46.101
set-community-name    private IP: 10.123.56.25
```

You can also configure the agent to restrict access to MIB subtrees based on IP addresses. The next example allows any host to get any object under *iso.org.dod.internet.mgmt.mib-2*, except for objects in the *interfaces* subtree. The minus sign (-) in front of *interfaces* instructs the agent to disallow access to this subtree.

```
get-community-name    public VIEW: mib-2 -interfaces
```

The final example sets up multiple community names for both *sets* and *gets*. An administrator who is located at host 10.123.46.25 and knows the *admin* community string has read access to the entire MIB tree; with the *adminset* community string, he has write access to the entire tree. Someone with the *operator* community string can sit anywhere and access everything in *mib-2* except for the *interfaces* subtree, but must be sitting at his desk (10.123.56.101) to issue *sets* and is not allowed to set anything in the *mib-2* subtree.

```
get-community-name    operator VIEW: mib-2 -interfaces
get-community-name    admin    IP: 10.123.56.25
set-community-name    operset  IP: 10.123.46.101 VIEW: -mib-2
set-community-name    adminset IP: 10.123.56.25
```

7.3.4 Net-SNMP (Formerly UCD-SNMP)

Net-SNMP is an open source agent that is freely available from <http://net-snmp.sourceforge.net>. We will focus on Net-SNMP Version 4.2, which is the most recent as of this publication. Once you have downloaded and unpacked the distribution, *cd* into the directory in which you unpacked Net-SNMP and read the *README* and *INSTALL* files. These files provide general information on installing the agent and don't require much explanation here.

Net-SNMP uses a configure script to make sure your environment has some key utilities and libraries installed, so it can be compiled successfully. Many configuration options are settable when you run this script. To see a list of them, run the following command:

```
ucd-snmp-4.2/> ./configure --help
```

One common option is *-prefix=PATH*. This specifies an alternate installation directory. By default, Net-SNMP will install in */usr/local/bin*, */usr/local/man*, etc.

We'll be running configure without any options, which means our Net-SNMP build will have default values assigned for various options. For example, the agent binary will be placed in `/usr/local/sbin`. Run the following command to begin the configuration process:

```
ucd-snmp-4.2/> ./configure
```

You will see various messages about what features configure is looking for and whether or not they're found.

After running for a while, configure will ask for some basic SNMP information:

```
***** Configuration Section *****
```

You are about to be prompted by a series of questions. Answer them carefully, as they determine how the snmp agent and related applications are to function.

After the configure script finishes, you can browse the newly created `config.h` file for further - less important - parameters to modify. Be careful if you re-run configure though since `config.h` will be over written.

```
-Press return to continue-
```

When you type Return, you'll be prompted for the system contact information:

```
disabling above prompt for future runs... yes
checking System Contact Information...
```

```
*** System Contact Information:
```

Describes who should be contacted about the host the agent is running on. This information is available in the MIB-II tree. This can also be over-ridden using the "syscontact" syntax in the agent's configuration files.

```
System Contact Information (root@): snmpadmin@ora.com
setting System Contact Information to... snmpadmin@ora.com
checking System Location...
```

We've decided to set our contact information to something useful, but we could have left it blank. The next item you're asked to configure is system location. We've chosen an informative value, but again could have left it blank:

```
*** System Location:
```

Describes the location of the system. This information is available in the MIB-II tree. This can also be over-ridden using the "syslocation" syntax in the agent's configuration files.

```
System Location (Unknown): FTP Server #1, O'Reilly Data Center
setting System Location to... FTP Server #1, O'Reilly Data Center
checking Location to write logfile...
```

The final option you need to configure is the snmpd log file location:

*** Logfile location:

Enter the default location for the snmpd agent to dump information & errors to. If not defined (enter the keyword "none" at the prompt below) the agent will use stdout and stderr instead. (Note: This value can be over-ridden using command line options.)

Location to write logfile (/var/log/snmpd.log):
setting Location to write logfile to... /var/log/snmpd.log

*** snmpd persistent storage location:

Enter a directory for the snmp library to store persistent data in the form of a configuration file.

Location to write persistent information (/var/ucd-snmp):
setting Location to write persistent information to... /var/ucd-snmp
updating cache ./config.cache
creating ./config.status
creating Makefile
creating MakefileMakefile
creating snmplib/Makefile
creating agent/Makefile
creating apps/Makefile
creating apps/snmpnetstat/Makefile
creating agent/mibgroup/Makefile
creating agent/dlmods/Makefile
creating local/Makefile
creating testing/Makefile
creating man/Makefile
creating ov/Makefile
creating mibs/Makefile
creating config.h

The default value is /var/log/snmpd.log, which should work on most Unix systems.

When the configure script finishes, it creates a system-specific file named *config.h*. Before you continue, take a look through this file. It houses many local configuration variables that you may want to change before you start compiling. Here are some snippets from my *config.h* file:

```
/* default list of mibs to load */

#define DEFAULT_MIBS "IP-MIB:IF-MIB:TCP-MIB:UDP-MIB:SNMPv2-MIB: \
RFC1213-MIB:UCD-SNMP-MIB:SNMPv2-PARTY-MIB:SNMPv2-M2M-MIB: \
SNMP-VIEW-BASED-ACM-MIB"

/* default location to look for mibs to load using the above tokens
   and/or those in the MIBS environment variable */

#define DEFAULT_MIBDIRS "/usr/local/share/snmp/mibs"
```

```
/* LOGFILE: If defined it closes stdout/err/in and opens this in out/err's
place. (stdin is closed so that sh scripts won't wait for it) */
```

```
#define LOGFILE "/var/log/snmpd.log"
```

```
/* default system contact */
```

```
#define SYS_CONTACT "snmpadmin@ora.com"
```

```
/* system location */
```

```
#define SYS_LOC "FTP Server #1, O'Reilly Data Center"
```

You can now compile your new package with the *make* command. The compilation process displays many messages, most of which you can ignore. In short, if it completes, you've succeeded and can proceed to installation. If not, you will see errors and should investigate what went wrong. If you tweaked the *config.h* file and your build failed, try recreating *config.h*. Without modifying this new *config.h*, try another build. This will weed out any problems you created within that file.

Install your new package with the command *make install*. By default, this command installs various executables in */usr/local/bin* and other important information in */usr/local/share/snmp*.

At this point, you can configure the agent further by using one of two approaches:

- Running the program */usr/local/bin/snmpconf*, which asks you a lot of questions and creates a configuration file. The configuration script is surprisingly confusing, though, so it's hard to recommend this approach.
- Crafting a configuration by hand. If you're not interested in SNMPv3, this is fairly easy.

7.3.4.1 Running the configuration script

The configuration script is rather long and complex. Here are a few pointers:

- It starts by asking whether you want to create *snmp.conf* or *snmpd.conf*. To configure the agent, select *snmpd.conf*. *snmp.conf* sets up some defaults for command-line tools such as *snmpget*. Strictly speaking, creating *snmp.conf* isn't necessary.
- Most of the configurable options have to do with SNMPv3. Although Version 3 is an important step forward, you can almost certainly ignore this; very few vendors support v3. Version 3 is discussed in [Appendix F](#).
- When you're finished configuring, the script leaves the configuration file in your current directory. You can either place the files in *~/.snmp*, if they're for your own use, or in */usr/local/share/snmp*, if you want this configuration to be used by everyone on the system.

7.3.4.2 Creating a configuration by hand

If you don't want to do anything complex, creating your own configuration file is easy. Here's a very simple configuration file:

```
syslocation    "O'Reilly Data Center"
syscontact     snmpadmin@oreilly.com
rwcommunity    private
rocommunity    public
authtrapenable 1
trapcommunity  trapsRus
trapsink        nmshost.oreilly.com
trap2sink       nmshost.oreilly.com
```

The configuration items should be familiar: we're setting up the system location; the system contact; the read-write, read-only, and trap community strings; and the destination to which traps should be sent. We're also enabling authentication traps. Note that we configured destinations for both SNMP Version 1 and Version 2 traps. The trap destination lines (`trapsink` and `trap2sink`) can also have a trap community string, if the NMS at the given host requires a different community name.

The `rwcommunity` and `rocommunity` lines allow us to be a bit more sophisticated than the example indicates. We're allowed to specify the network or subnet to which the community strings apply, and an object ID that restricts queries to MIB objects that are underneath that OID. For example, if you want to restrict read-write access to management stations on the subnetwork 10.0.15.0/24, you could use the line:

```
rwcommunity    private 10.0.15.0
```

If you take this route, you should certainly look at the `EXAMPLE.conf` file in the directory in which you built Net-SNMP. You can modify this file and install it in the appropriate location (either `~/.snmp/snmpd.conf` or `/usr/local/share/snmp/snmpd.conf`), or you can take ideas from it and use them in your own configuration. It includes some particularly clever tricks that we'll discuss in [Chapter 11](#) but that are well beyond the simple configuration we're discussing here.

7.3.5 Concord SystemEDGE Agent for Unix and NT

Concord SystemEDGE is a commercial product that can be used as a subagent to the standard Windows NT agent. On Unix systems, this agent can be used either as a standalone agent or side-by-side with an existing agent. It runs on Linux, Solaris, and other operating systems. The CD on which the product is shipped includes agents for all the platforms SystemEDGE supports. Whenever possible, SystemEDGE uses the platform's native package manager to make installation easier. Each architecture-dependent version of the agent comes with an easy-to-follow

README file for installation. See [Chapter 11](#) for a discussion of this agent's capabilities.

7.3.5.1 Simple configuration

The SystemEDGE configuration file is located in */etc/sysedge.cf*. Use your editor of choice to make changes to this file. You must stop and restart SystemEDGE for your changes to take effect. The configuration file format is the same for all the versions of SystemEDGE.

For a typical SNMP configuration, *sysedge.cf* looks like this:

```
community public read-only
community veryprivate read-write 127.0.0.1 10.123.56.25
community traps 127.0.0.1
```

Comment lines begin with a # character. The first parameter sets the read-only community to *public*. The read-write community is defined to be *veryprivate*. The two IP addresses following the read-write community string are an access list that tells the agent to allow *set* operations from localhost (127.0.0.1) and 10.123.56.25 only. Always use an access list if possible; without this security feature, any host can execute *set* operations. Note that there is a space between the two addresses, not a Tab character. The third option tells the agent where to send traps; in this case, to localhost (127.0.0.1).

The agent sends authentication-failure traps by default, and we strongly recommend using them. If you don't want authentication-failure traps, include the following line in your configuration file:

```
no_authen_traps
```

7.3.5.2 Advanced configuration

SystemEDGE provides some powerful self-monitoring capabilities. These extensions (found only in Concord's Empire private enterprise MIB) are similar to the Remote Network Monitoring (RMON) MIB, which is discussed in [Chapter 9](#). Empire's extensions can reduce network load by allowing the agent, instead of an NMS, to perform monitoring (polling) of important system objects. For example, the agent can be instructed to make sure the free space available in the root filesystem stays above some predefined threshold. When this threshold is crossed, the agent sends a trap to the NMS so the condition can be dealt with appropriately.

The following line shows how you can monitor and restart *sendmail* if it dies:

```
watch process procAlive 'sendmail' 1 0x100 60 'Watch Sendmail' '/etc/init.d/sendmail
start'
```

This monitor sends a trap to the NMS, defined earlier as *community traps 127.0.0.1*, when the *sendmail* process dies. The agent

then executes `/etc/init.d/sendmail start` to restart the process. The general form of this command is:

```
watch process procAlive 'procname' index flags interv 'description' 'action'
```

The `procname` parameter is a regular expression that SystemEDGE uses to select the processes that it is monitoring; in this case, we're watching processes with the name `sendmail`. Each entry in the process-monitoring table must have a unique index; in this example, we used the value 1. We could have picked any integer, as long as that integer was not already in use in the table. The `flags` parameter is a hexadecimal^[1] flag that changes the behavior of the monitor. We specified a flag of `0x100`, which tells the monitor that the process it's watching spawns child processes; this flag ensures that SystemEDGE will take action only when the parent `sendmail` process dies, not when any of the children die. The use of process-monitor flags is beyond the scope of this chapter; see the manual that comes with SystemEDGE for more information. The `interv` parameter specifies how often (in seconds) the agent checks the process's status. We have set the interval to 60 seconds. The `description` parameter contains information about the process being monitored; it can be up to 128 characters in length. It is a good idea to use a description that indicates what is being monitored, since the agent stores this value in the monitor table for retrieval by an NMS and includes it in the variable bindings when a trap is sent. The final parameter is the action the monitor will take when the process dies; we chose to restart the daemon.

^[1] Generally speaking, there are several ways to represent hexadecimal numbers. SystemEDGE uses the notion of a number prefixed with `0x`, which should be familiar to C and Perl programmers.

SystemEDGE can be extended by using plug-ins. These plug-ins manage and monitor applications such as Apache (web server), Exchange (Microsoft mail), and Oracle (database), to name a few. A "top processes" plug-in named `topprocs` comes with every distribution. The following statement tells SystemEDGE to load this plug-in for 64-bit Solaris (this statement is similar for NT and other Unix platforms):

```
sysedge_plugin /opt/EMPSysedge/plugins/topprocs/topprocs-sol64bit.so
```

The folks at Concord have taken great care to add useful comments to the `sysedge.cf` file. The comments are often all you need to configure the agent.

7.3.6 Cisco Devices

Cisco Systems produces a wide range of routers, switches, and other networking equipment. The configuration process is virtually the same on all Cisco devices, because they share the IOS operating system.^[2] There are some minor differences in the

parameters that can be configured on every device; these generally have to do with the capabilities of the device, rather than the SNMP implementation.

^[2] There are some exceptions to this rule, such as the PIX firewalls. These exceptions usually mean that the product is made by a company that Cisco acquired.

To configure the SNMP parameters, you must be in enable mode. You can use the following commands to see what traps are available:

```
router> enable
Password: mypassword
router# config terminal
router(config)#snmp-server enable traps ?
  bgp          Enable BGP state change traps
  envmon       Enable SNMP environmental monitor traps
  frame-relay  Enable SNMP frame-relay traps
  isdn         Enable SNMP isdn traps
  <cr>
```

The question mark tells the router to respond with the possible completions for the command you're typing. You can use this feature throughout the entire command-line interface. If the part of the command you have already typed has a syntax error, the router will give you the "Unrecognized command" message when you type the question mark. <cr> tells you that you can exit without configuring the command (snmp-server enable traps in this case) by typing a carriage return.

7.3.6.1 Simple configuration

Here's a simple configuration that lets you start using the SNMP agent:

```
router(config)#snmp-server community private RW
router(config)#snmp-server community public RO
router(config)#snmp-server trap-authentication
router(config)#snmp-server location Delta Building - 1st Floor
router(config)#snmp-server contact J Jones
router(config)#snmp-server host 10.123.135.25 public
```

Most of these commands set parameters with which you should be familiar by now. We define two communities, public and private, with read-only (RO) and read-write (RW) permissions, respectively. snmp-server trap-authentication turns on authentication-failure traps. The command snmp-server host 10.123.135.25 public configures the destination to which traps should be sent. The IP address is set to the address of our NMS. The community string public will be included in the traps.

7.3.6.2 Advanced configuration

The following configuration item tells the device what interface it should use when sending out SNMP traps:

```
router(config)#snmp-server trap-source VLAN1
```

Configuring the trap source is useful because routers, by definition, have multiple interfaces. This command allows you to send all your traps out through a particular interface.

There may be times when you want to send only certain traps to your NMS. The next item sends only environmental monitor traps to the specified host, 172.16.52.25 (the envmon option is not available on all Cisco devices):

```
router(config)#snmp-server host 172.16.52.25 public envmon
```

One of the most frightening SNMP sets is the Cisco shutdown, which lets you shut down the router from the NMS. The good news is that you have to include a switch in the configuration before the router will respond to shutdown commands. Issuing the following command disables shutdowns:

```
router(config)#no snmp-server system-shutdown
```

To receive traps about authentication failures (something trying to poll your device with the wrong community name) add the following line:

```
router(config)#snmp-server trap-authentication
```

The final advanced configuration parameter is an access list. The first line sets up access list 15. It states that the IP address 10.123.56.25 is permitted to access the agent. The second line says that anyone that passes access list 15 (i.e., a host with IP address 10.123.56.25) and gives the community name notsopublic has read-only (RO) access to the agent. Access lists are a very powerful tool for controlling access to your network. They're beyond the scope of this book, but if you're not familiar with them, you should be.

```
router(config)#access-list 15 permit 10.123.56.25
```

```
router(config)#snmp-server community notsopublic RO 15
```

That's it! You now have a working SNMP configuration for your Cisco router.

7.3.7 APC Symetra

APC's uninterruptible power supplies (UPSs) are typical of a large class of products that aren't usually considered network devices, but that have incorporated a network interface for the purpose of management.

To configure an APC UPS, you can use its management port (a familiar serial port to which you can connect a console terminal) or, assuming that you've performed basic network configuration, *telnet* to the UPS's IP address. SNMP configuration is the same regardless of the method you use. Either way, you get a Text User Interface (TUI) that presents you with rather old-fashioned menus -- you type your menu selection (usually a number) followed by Enter to navigate through the menus.

We'll assume that you've already performed basic network configuration, such as assigning an IP address for the UPS. To configure SNMP, go to the Network menu and select "5" to go into the SNMP submenu. You should get a menu like this:

```
----- SNMP -----  
  
1- Access Control 1  
2- Access Control 2  
3- Access Control 3  
4- Access Control 4  
5- Trap Receiver 1  
6- Trap Receiver 2  
7- Trap Receiver 3  
8- Trap Receiver 4  
9- System  
10- Summary  
  
?- Help  
<ENTER> Redisplay Menu  
<ESC> Return To Previous Menu
```

>
You need to configure three distinct sections: Access Control, Trap Receiver, and System. To see a summary of the current SNMP settings, use the Summary submenu.

This particular device allows us to specify four IP addresses for access control and four IP addresses to receive traps. The access control items allow you to configure the IP addresses of your management stations -- this is similar to the access lists we've seen in other devices, and is obviously basic to security. The UPS will reply only to queries from the IP addresses you have listed. Configuration is a bit awkward -- you need to go to a separate menu to configure each IP address. Here's what you'll see when configuring the Access Control 1 submenu:

```
----- Access Control 1 -----  
  
Access Control Summary  
# Community Access NMS IP  
-----  
1 public Read 10.123.56.25  
2 private Write 10.123.56.25  
3 public2 Disabled 0.0.0.0  
4 private2 Disabled 0.0.0.0  
  
1- Community : public  
2- Access Type : Read  
3- NMS IP Address : 10.123.56.25  
4- Accept Changes :  
  
?- Help  
<ENTER> Redisplay Menu
```

<ESC> Return To Previous Menu

>

The first part of the menu summarizes the state of access control. On this menu, we can change only the first item on the list. The special address 0.0.0.0 is a wildcard -- it means that the UPS will respond to queries from any IP address. Although addresses 3 and 4 are set to 0.0.0.0, these addresses are currently disabled, and that's how we want to keep them. We want the UPS to respond only to the management stations we explicitly list.

On this menu, we've configured items 1 (the community string), 2 (the access type), and 3 (the IP address). We've set the community string to **public** (not a choice you'd want in a real configuration), the access type to **Read** (allowing various SNMP *get* operations, but no *set* operations), and the NMS IP address to 10.123.56.25. The net effect is that the UPS's SNMP agent will accept *get* requests from IP address 10.123.56.25 with the community name **public**. When you are satisfied with the configuration, enter a 4 to accept your changes.

To configure the second access control item, press Esc to return to the previous menu; then select 2. As you can see, we allow 10.123.56.25 to perform *set* operations. We don't have any other management stations, so we've left items 3 and 4 disabled.

Once the Access Control section is complete, you can start configuring traps. The Trap Receivers section is simply a list of NMSs that receive traps. As with Access Control, four trap receivers can be configured. To get to the first trap receiver, return to the SNMP menu and select menu 5. A typical trap receiver setup looks like this:

----- Trap Receiver 1 -----

Trap Receiver Summary

#	Community	Generation	Authentication	Receiver NMS IP
1	public	Enabled	Enabled	10.123.56.25
2	public	Enabled	Enabled	0.0.0.0
3	public	Enabled	Enabled	0.0.0.0
4	public	Enabled	Enabled	0.0.0.0

- 1- Trap Community Name : public
- 2- Trap Generation : Enabled
- 3- Authentication Traps: Enabled
- 4- Receiver NMS IP : 10.123.56.25
- 5- Accept Changes :

?- Help

<ENTER> Redisplay Menu

<ESC> Return To Previous Menu

>

Once again, the first part of the menu is a summary of the trap receiver configuration. We've already set the first trap receiver to the address of our NMS, enabled trap generation, and enabled the generation of authentication traps -- as always, a good idea. The traps we generate will include the community string public. Note that trap receivers 2, 3, and 4 are set to 0.0.0.0. On this menu, 0.0.0.0 is not a wildcard; it's just an invalid address that means you haven't yet configured the trap receiver's IP address. It's basically the same as leaving the entry disabled.

The final configuration items that should be set are on the System submenu, found under the SNMP main menu:

----- System -----

```
1- sysName      : ups1.ora.com
2- sysContact   : Douglas Mauro
3- sysLocation  : Apache Hilo Deck
4- Accept Changes :
```

?- Help

<ENTER> Redisplay Menu

<ESC> Return To Previous Menu

>

After you have finished configuring all your SNMP parameters, use the Summary submenu for a quick look at what you have done. A typical setup will look something like this:

SNMP Configuration Summary

```
sysName      : ups1.ora.com
sysLocation   : Apache Hilo Deck
sysContact    : Douglas Mauro
```

Access Control Summary

#	Community	Access	NMS IP
1	public	Read	10.123.56.25
2	private	Write	10.123.56.25
3	public2	Disabled	0.0.0.0
4	private2	Disabled	0.0.0.0

Trap Receiver Summary

#	Community	Generation	Authentication	Receiver NMS IP
1	public	Enabled	Enabled	10.123.56.25
2	public	Enabled	Enabled	0.0.0.0
3	public	Enabled	Enabled	0.0.0.0
4	public	Enabled	Enabled	0.0.0.0

Press <ENTER> to continue...

Upon completion and verification, use the Esc key to take you all the way out to the Logout menu.

Chapter 8. Polling and Setting

We've put a lot of work into getting things set up so that we can use SNMP effectively. But now that we've installed a fancy node manager and configured agents on all our devices, what can we do? How can we interact with the devices that are out there?

The three basic SNMP operations are *snmpget*, *snmpset*, and *snmpwalk*. They are fairly self-explanatory: *snmpget* reads a value from a managed device, *snmpset* sets a value on a device, and *snmpwalk* reads a portion of the MIB tree from a device. For example, you can use *snmpget* to query a router and find out its administrative contact (i.e., the person to call if the router appears to be broken), *snmpset* to change this contact information, and *snmpwalk* to traverse a MIB to get an idea of which objects the router has implemented or to retrieve status information on all the router's interfaces.

This chapter shows you how to use these operations in day-to-day network management. First, we will use Perl to demonstrate how you can *set*, *get*, and *walk* objects in a script (the nice thing about using Perl is that you can easily extend the simple scripts in this chapter to fit your needs and environment). We will then use HP OpenView and Net-SNMP to perform the same operations, but from the command line. Finally, as an alternative to the command line, we will demonstrate OpenView's graphical MIB Browser, which has a nice interface for getting, setting and walking MIB data.

8.1 Retrieving a Single MIB Value

Let's start by querying a router for the name of its administrative contact. This operation, called polling, is accomplished with the SNMP *get* command. The following Perl script, *snmpget.pl*, uses an SNMP Perl module to retrieve the information we want ([Chapter 5](#) contains the URL for this module):

```
#!/usr/local/bin/perl
#filename: /opt/local/perl_scripts/snmpget.pl
use BER;
use SNMP_util;
use SNMP_Session;
$MIB1 = ".1.3.6.1.2.1.1.4.0";
$HOST = "orarouter1";
($value) = &snmpget("public@$HOST", "$MIB1");
if ($value) { print "Results :$MIB1: :$value:\n"; }
else { warn "No response from host :$HOST:\n"; }
```

This script is obviously very primitive, but it is also easy to understand, even if you're not an experienced Perl user. It's importance isn't in what it does, which is very little, but as a template you can use to insert SNMP operations into other programs. (If you are not used to writing quick Perl programs, or are unfamiliar with the language, a good starting point is the official Perl web site, <http://www.perl.com>.) The script starts with three use statements, which are similar to #include statements in C. The use statements load Perl modules containing functions and definitions for working with SNMP. The three modules we use are:

BER

Describes how to encode management data into bit patterns for transmission. *Basic Encoding Rules* (BER) is an ISO standard.

SNMP_util

Defines a set of functions that use the `SNMP_Session` module to make it much more programmer-friendly. `SNMP_util` itself uses `BER` and `SNMP_Session`, but in this first script we chose to reference these other modules explicitly. In future programs, we'll just use `SNMP_util`.

SNMP_Session

Provides Perl with core SNMP functionality.

The next two lines specify the data we want to get. We have hardcoded the object ID of a particular piece of data defined by the MIB and the hostname from which we want to retrieve this MIB data. In a more flexible program, you might want to get these values from the command line, or build a user interface to help users specify exactly what they are interested in retrieving. For the time being, however, this will get us started. It is easy enough to replace `orarouter1` with the hostname or IP address of the device you want to poll. The OID we are requesting is stored in the variable `$MIB1`. The value `.1.3.6.1.2.1.1.4.0` requests the device's administrative contact. Again, you can replace this with any OID of your choice. We used the numeric form of this object, but you can also use the textual form for the OID, which is `.org.dod.internet.mgmt.mib-2.system.sysContact.0`. You can abbreviate this further to `sysContact` because `SNMP_util` defines some parts of the OID string for us (for example, `SNMP_util` defines `sysContact` as `1.3.6.1.2.1.1.4.0`), but it's often safer to be explicit and use the entire OID. Don't forget to include the `.0`, which states that we want the first (0) and only instance of `iso.org.dod.internet.mgmt.mib-2.system.sysContact.0`, at the end of your OID.

The next line polls the device. The `snmpget` function retrieves the data from the device specified by the variable `$HOST`. Notice the two arguments to the function. The first is the device we want to poll, preceded by the community name `public`. (If you

need to use another community name -- you did change the community names when you configured the device, didn't you? -- you'll have to modify this line and insert your community name in place of it.) The second argument to `snmpget` is the OID in which we are interested. If you type the code in yourself, do not forget the parentheses around `$value`. If you omit the parentheses, `$value` will be set to the number of items in the array `snmpget` returns.

Once we have polled the device, we print either the output or an error message. I put a colon before and after any output that I print; this makes it easy to see if there are any hidden characters in the output. The decimal integer "16" is very different from "16\n", which is the decimal integer 16 followed by a newline character.

Now let's run the program:

```
$ /opt/local/perl_scripts/snmpget.pl
Results :.1.3.6.1.2.1.1.4.0: :ORA IT Group:
```

`snmpget.pl` prints the OID we requested, followed by the actual value of that object, which is `ORA IT Group`. Don't worry if the return value for `sysContact` is wrong or blank. (The trick of putting colons before and after the output will make it clear if `sysContact` is blank or empty.) This probably means that no one has configured an administrative contact, or that it was configured incorrectly. We'll show you how to fix that when we discuss the set operation. If you get an error, skip to the end of this chapter to see a list of some errors and their appropriate fixes.

We will now modify `snmpget.pl` to poll any host and any OID we want. This is accomplished by passing the host and OID as command-line arguments to the Perl script:

```
#!/usr/local/bin/perl
#filename: /opt/local/perl_scripts/snmpget.pl
use SNMP_util;
$MIB1 = shift;
$HOST = shift;
($MIB1) && ($HOST) || die "Usage: $0 MIB_OID HOSTNAME";
($value) = &snmpget("$HOST", "$MIB1");
if ($value) { print "Results :$MIB1: :$value:\n"; }
else { warn "No response from host :$HOST:\n"; }
```

Now that this program is a little more flexible, it is possible to look up different kinds of information on different hosts. We even left out the community string, which allows us to poll hosts with different community names. Here's how to run the new version of `snmpget.pl`:

```
$ /opt/local/perl_scripts/snmpget.pl .1.3.6.1.2.1.1.1.0 public@orarouter1
Results :.1.3.6.1.2.1.1.1.0: :Cisco Internetwork Operating System Software
IOS (tm) 3000 Software (IGS-I-L), Version 11.0(16), RELEASE SOFTWARE (fc1)
Copyright (c) 1986-1997 by cisco Systems, Inc.
Compiled Tue 24-Jun-97 12:20 by jaturner:
```

In this example, we asked the router to describe itself by looking up the OID .1.3.6.1.2.1.1.1.0 (system.sysDesc.0). The result tells us that orarouter1 is a Cisco router running Version 11.0(16) of the IOS operating system, along with some other useful information.

8.1.1 Using HP OpenView to Retrieve Values

Let's start by looking up our router's administrative contact (system.sysContact.0) and see if we get the same result as we did with our previous Perl script. The arguments to OpenView's snmpget ^[1] are the community name, the hostname of the device we want to poll, and the OID of the data we are requesting; we gave the OID in numeric form, but again, we could have given it as a text string:

^[1] Most OpenView executable files are located in /opt/OV/bin.

```
$ /opt/OV/bin/snmpget -c public orarouter1 .1.3.6.1.2.1.1.4.0
system.sysContact.0 : DISPLAY STRING- (ascii): ORA IT Group
```

Although this looks a little different from the output of the Perl script, it tells us the same thing. snmpget prints the OID we requested on the command line, making it easy to verify that we polled the right object. Again, note that the trailing .0 is important. The output also tells us the object's datatype:

DISPLAY STRING- (ascii). Back in [Chapter 2](#), we discussed the datatypes that SNMP uses; some of the common types are INTEGER, OCTET STRING, Counter, and IpAddress. Finally, the output gives us the information we asked for: the router is administered by the ORA IT Group, which is the value returned from the SNMP get request.

Now let's do the same thing using OpenView's GUI interface. From the Network Node Manager's display, select "Misc → SNMP MIB Browser."^[2] If you don't have NNM running, you can start the MIB Browser from the command line: /opt/OV/bin/xnmbrowser. [Figure 8-1](#) shows the GUI. Its input fields are similar to the variables we have been setting in our Perl scripts: Name or IP Address, Community Name, MIB Object ID, MIB Instance, SNMP Set Value, and MIB Values.

^[2] If you find that the SNMP MIB Browser menu item is grayed out and cannot be clicked on, click on an SNMP object on your NNM map. You should then be able to click on the menu item to start your GUI.

Let's use this browser to run an snmpget. Start by inserting a Name or IP Address and Community Name in the input boxes provided. To enter the object you want to retrieve, use the MIB Object ID field and the text box below it. MIB Object ID shows us that we are currently in the subtree .iso.org.dod.internet. The text area shows the objects at the next level of the tree: directory, mgmt, etc. (To see the numeric OIDs for these

objects, click on their names and then on the "Describe" button.) Then browse down through the MIB by double-clicking mgmt, then mib-2, system, and finally sysContact. Click on sysContact and then on "Start Query." The result that appears in the "MIB Values" field (as shown in [Figure 8-2](#)) should look very similar to the value that was returned in the command-line example.

Figure 8-1. OpenView xnmbrowser default

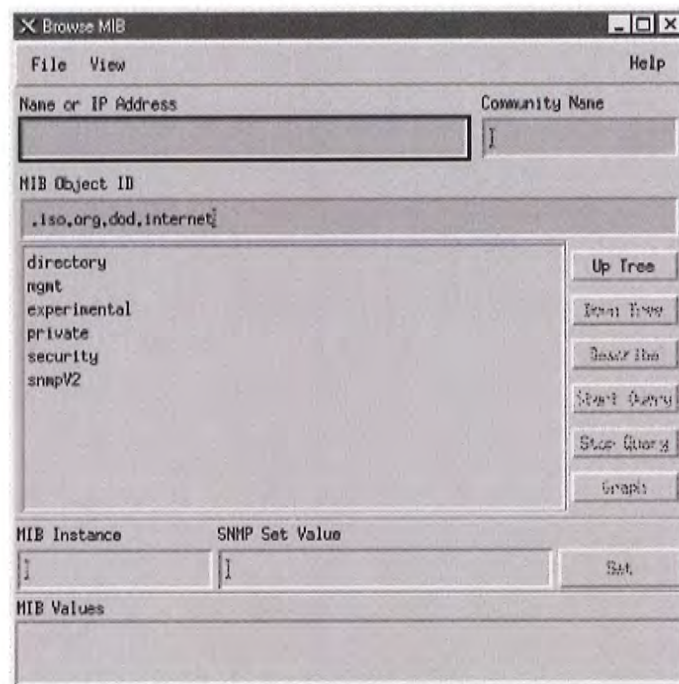
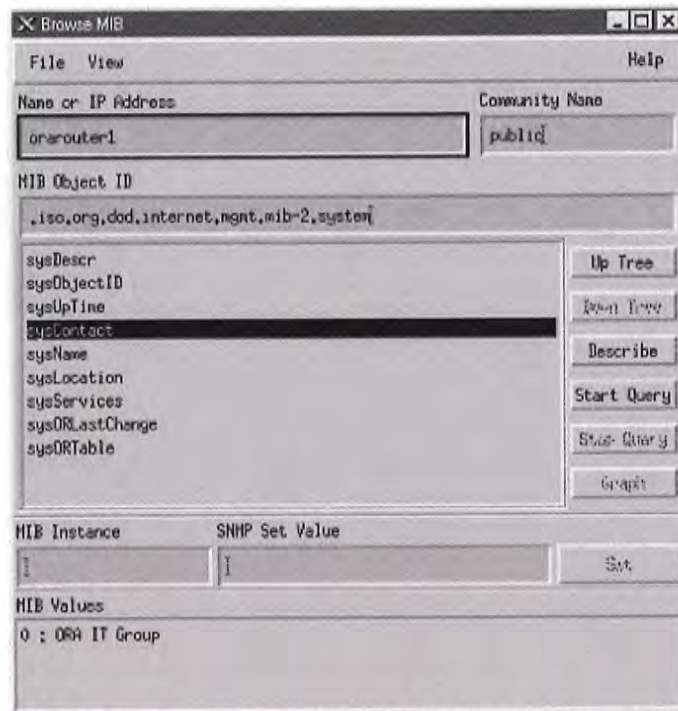


Figure 8-2. OpenView xnmbrowser response



Let's go back to the command line and poll for sysDesc again:

```
$ /opt/OV/bin/snmpget orarouter1 .1.3.6.1.2.1.1.0
system.sysDescr.0 : DISPLAY STRING- (ascii): Cisco Internetwork Operating
System Software IOS (tm) 3000 Software (IGS-I-L), Version 11.0(16), RELEASE
SOFTWARE (fc1)Copyright (c) 1986-1997 by cisco Systems, Inc. Compiled Tue
24-Jun-97 12:20 by jaturner
```

Looks the same, right? Notice that we left out the community string. We can do this because the default get community string is public, which is the correct community string for the target host, orarouter1. You can change your default community strings in OpenView's global settings. Let's see if we can get an object with a different datatype:

```
$ /opt/OV/bin/snmpget orarouter1 .1.3.6.1.2.1.1.3.0
system.sysUpTime.0 : Timeticks: (159857288) 18 days, 12:02:52.88
```

This command returns the system uptime, which is of type TimeTicks. TimeTicks (RFC 1155) represents a nonnegative integer, which counts the time in hundredths of a second since some epoch. Ignoring the number in parentheses, this shows me that my router has been up and operational for 18 days, 12 hours, 02 minutes, and so on. The big number in parentheses is the exact amount of time the machine has been up, in hundredths of seconds. If you do the math, you will see this adds up to 18.501 days, or 18 days, 12 hours, and a little bit: exactly what we expect.

8.1.2 Using Net-SNMP

The Net-SNMP tools provide an excellent command-line interface to SNMP operations. These tools are also commonly known as UCD-

SNMP -- you'll still find this older name in many references, and even in the code itself.

[Chapter 7](#) discussed how to compile, install, and configure the Net-SNMP agent. If you've done that, you've already compiled and installed the SNMP tools. They're shipped in the same package as the SNMP agent, and no real configuration is necessary for them. There is a configuration program, called `snmpconf`, which can be used to generate an `snmp.conf` file that provides default values for some of the options to the commands.^[3] Unless you're using SNMPv3, though, it isn't really necessary. It might be handy to set up a default community string but, in practice, this is of only limited use: you probably have different community strings on different devices, anyway. If you decide to use `snmpconf` to create the tool configuration file, make sure that you place `snmp.conf` in the `.snmp` subdirectory of your home directory or (if you want the options to apply to all users) in `/usr/local/share/snmp`.

^[3] This is the same command used to create `snmpd.conf`, which configures the Net-SNMP agent. The `snmp.conf` configuration file is similar in form to `snmpd.conf`.

We'll assume that you won't do any configuration and will simply use the tools "out of the box." Here's a simple poll that asks a router for its location:

```
$ snmpget orarouter1 public .1.3.6.1.2.1.1.6.0
system.sysLocation.0 = Sebastopol CA
```

It's fairly simple: we provided the hostname of the router we wanted to poll, a community string, and the OID of the object we wanted to retrieve. Instead of using the numeric OID, you can use the lengthy human-readable form. To save typing, `snmpget` assumes everything up to the object name and instance ID. Therefore, the following command is exactly equivalent to the previous one:

```
$ snmpget orarouter1 public sysLocation.0
system.sysLocation.0 = Sebastopol CA
```

We'll take a look at the `snmpwalk` and `snmpset` commands that come with the Net-SNMP package later in this chapter, but the package contains many tools and is well worth a more detailed explanation. One tool that's particularly useful is `snmptranslate`, which converts between the numeric and textual names of MIB objects and can do things such as look up the definition of an object in a MIB file. The software distribution comes with a number of standard MIBs; you can place additional MIB files in `/usr/local/share/snmp/mibs`. [Appendix C](#) gives an overview of the Net-SNMP package.

8.2 Retrieving Multiple MIB Values

The syntax for *snmpwalk* is similar to the syntax for its cousin, *snmpget*. As discussed in [Chapter 2](#), *snmpwalk* traverses a MIB starting with some object, continuously returning values until it gets to the end of that object's branch. For example, the upcoming Perl script begins walking the .iso.org.dod.internet.mgmt.mib-2.interfaces.ifTable.ifEntry.ifDescr object and provides a description of each Ethernet interface on the device it polls.

This new script is a minor modification of *snmpget.pl*. We turned the scalar *\$value* into the array *@values*; ^[4] we need an array because we expect to get multiple values back. We also called the function *snmpwalk* instead of *snmpget* (syntactically, the two functions are the same):

^[4] The Perl program we used earlier could have used the array instead of the scalar as well. This is possible because Perl's version of *snmpget* allows for multiple OIDs, not just one. To specify multiple OIDs, place a comma (,) between each OID. Remember to enclose each OID within its own double quotes.

```
#!/usr/local/bin/perl
#filename: /opt/local/perl_scripts/snmpwalk.pl
use SNMP_util;
$MIB1 = shift;
$HOST = shift;
($MIB1) && ($HOST) || die "Usage: $0 MIB_OID HOSTNAME";
(@values) = &snmpwalk("$HOST", "$MIB1");
if (@values) { print "Results :$MIB1: :@values:\n"; }
else { warn "No response from host :$HOST:\n"; }
```

Here's how to run the script:

```
$ /opt/local/perl_scripts/snmpwalk.pl .1.3.6.1.2.1.2.2.1.2 orarouter1
```

This command walks down the .iso.org.dod.internet.mgmt.mib-2.interfaces.ifTable.ifEntry.ifDescr object, returning information about the interfaces that are on the router. The results look something like this:

```
Results :.1.3.6.1.2.1.2.2.1.2: :1:Ethernet0 2:Serial0 3:Serial1:
```

The output depends on the interfaces on the host or router you are polling. To give some examples, I've run this script against some of the machines on my network. Here are the results.

Cisco 7000 router:

```
Results :.1.3.6.1.2.1.2.2.1.2: :1:Ethernet0/0 2:Ethernet0/1 3:TokenRing1/0
4:TokenRing1/1 5:TokenRing1/2 6:TokenRing1/3 7:Serial2/0 8:Serial2/1
9:Serial2/2 10:Serial2/3 11:Serial2/4 12:Serial2/5 13:Serial2/6 14:Serial2/7
15:FastEthernet3/0 16:FastEthernet3/1 17:TokenRing4/0 18:TokenRing4/1:
```

Sun workstation:

```
Results :.1.3.6.1.2.1.2.2.1.2: :1:lo0 2:hme0:
```

Windows NT PC:

Results :.1.3.6.1.2.1.2.2.1.2: :1:MS TCP Loopback interface
2:PCI2 Token-Ring Network 16/4 Adapter :

APC uninterruptible power supply:

Results :.1.3.6.1.2.1.2.2.1.2: :1:pda:

For each device, we see at least one interface. As you'd expect, the router has many interfaces. The first interface on the router is listed as 1:Ethernet0/0, the second is listed as 2:Ethernet0/1, and so on, up through interface 18. SNMP keeps track of interfaces as a table, which can have many entries. Even single-homed devices usually have two entries in the table: one for the network interface and one for the loopback interface. The only device in the example above that really has a single interface is the APC UPS -- but even in this case, SNMP keeps track of the interface through a table that is indexed by an instance number.

This feature allows you to append an instance number to an OID to look up a particular table element. For example, we would use the OID .1.3.6.1.2.1.2.2.1.2.1 to look at the first interface of the Cisco router, .1.3.6.1.2.1.2.2.1.2.2 to look at the second, and so on. In a more human-readable form, ifDescr.1 is the first device in the interface description table, ifDescr.2 is the second device, and so on.

8.2.1 Walking the MIB Tree with OpenView

Switching over to OpenView's snmpwalk, let's try to get every object in the .iso.org.dod.internet.mgmt.mib-2.system subtree:

```
$ /opt/OV/bin/snmpwalk oraswitch2 .1.3.6.1.2.1.1
system.sysDescr.0 : DISPLAY STRING- (ascii): Cisco Internetwork Operating
System Software IOS (tm) C2900XL Software (C2900XL-H-M), Version 11.2(8)
SA1,RELEASE SOFTWARE (fc1)Copyright (c) 1986-1998 by cisco Systems, Inc.
Compiled Tue 03-Feb-98 14:59 by rheaton
system.sysObjectID.0: OBJECT IDENTIFIER:
.iso.org.dod.internet.private.enterprises.cisco.ciscoProducts.cisco2509
system.sysUpTime.0 : Timeticks: (168113316) 19 days, 10:58:53.16
system.sysContact.0 : DISPLAY STRING- (ascii): J.C.M. Pager 555-1212
system.sysName.0 : DISPLAY STRING- (ascii): oraswitch2.ora.com
system.sysLocation.0 : DISPLAY STRING- (ascii): Sebastopol CA
system.sysServices.0 : INTEGER: 6
```

Let's go to the GUI MIB Browser and try that same walk. Repeat the steps you took for the snmpget using the GUI. This time insert the OID .1.3.6.1.2.1.1 and hit the "Start Query" button. Check out the results.



The GUI figures out whether it needs to perform an `snmpwalk` or `snmpget`. If you give an instance value (being specific), the browser performs an `snmpget`. Otherwise, it does an `snmpwalk`. If you are looking for more speed and less cost to your network, include the instance value.

What will happen if you walk the entire `.iso` subtree? It may hurt or even crash your machine, because in most cases the device can return several thousand values. Each interface on a router can add thousands of values to its MIB tables. If each object takes `.0001` seconds to compute and return, and there are 60,000 values to return, it will take your device 6 seconds to return all the values -- not counting the load on the network or on the monitoring station. If possible, it is always a good idea to perform an `snmpwalk` starting at the MIB subtree that will provide you with the specific information you are looking for, as opposed to walking the entire MIB.

It might be useful to get a feel for how many MIB objects a given device has implemented. One way to do this is to count the number of objects each `snmpwalk` returns. This can be accomplished with the Unix `grep` command. The `-c` switch to `grep` tells it to return the number of lines that matched. The period (`.`) tells `grep` to match everything. Starting from the `.system` object (`.1.3.6.1.2.1.1`), let's go back one and see how many objects are implemented in the `mib-2` subtree. Take off the last `.1` off the object ID and run the `snmpwalk` command again, this time piping the results into `grep -c`:

```
$ /opt/OV/bin/snmpwalk oraswitch2 .1.3.6.1.2.1 | grep -c .
```

The number of objects you see will depend on the type of device and the software running on it. When I tried several different devices, I got results ranging from 164 to 5193.

This command is great when you want to walk a MIB to see all the types of values that a device is capable of returning. When I am trying out a new device or MIB, I often walk some decent-sized portion of the MIB and read through all the returned values, looking for any info that may be of interest. When something catches my eye, I go to the MIB definition and read its description. Many GUI MIB Browsers allow you to check the description with the click of a button. In OpenView's GUI, click on the OID and then on "Describe."

8.2.2 Walking the Tree with Net-SNMP

Net-SNMP's `snmpwalk` is very similar in form and function to OpenView's. Here's how you use it:

```
$ snmpwalk orarouter1 public .1.3.6.1.2.1.1
system.sysDescr.0 = Cisco Internetwork Operating System Software
IOS (tm) C820 Software (C820-Y6-M), Version 12.1(3)XG3, EARLY DEPLOYMENT RELEASE
```

```
SOFTWARE (fc1)
TAC:Home:SW:IOS:Specials for info
Copyright (c) 1986-2000 by cisco Systems, Inc.
Compiled Wed 20-Dec-00 16:21
system.sysObjectID.0 = OID: enterprises.9.1.284
system.sysUpTime.0 = Timeticks: (100946413) 11 days, 16:24:24.13
system.sysContact.0 = thenetworkadministrator@oreilly.com
system.sysName.0 = orarouter1@oreilly.com
system.sysLocation.0 = Sebastopol CA
system.sysServices.0 = 6
system.sysORLastChange.0 = Timeticks: (0) 0:00:00.00
```

There aren't any real surprises. Again, you can use an object name instead of a numerical ID; because you're walking a tree, you don't need to specify an instance number.

8.3 Setting a MIB Value

With *snmpget* and *snmpwalk*, we have retrieved management information only from devices. The next logical step is to change the value of a MIB object via SNMP. This operation is known as *snmpset*, or *set*. In this section we'll read the value of an object, use *snmpset* to change its value, then read the value again to prove that it's been changed.

There's obviously some danger here: what happens if you change a variable that's critical to the state of the system you're monitoring? In this chapter, we'll deal only with some simple objects, such as the administrative contact, that won't damage anything if they're changed incorrectly. Therefore, if you keep the OIDs correct, you shouldn't worry about hurting any of your devices. All the objects we set in this chapter have an **ACCESS** of read-write. It's a good idea to get a feel for which objects are writable by reading the MIB in which the object is defined -- either one of the RFCs or a MIB file provided by your vendor.

Let's get started. Run the following OpenView command (or use one of the other programs we've discussed) to find out the *sysContact* for your chosen device:

```
$ /opt/OV/bin/snmpget -c public orarouter1 .1.3.6.1.2.1.1.4.0
system.sysContact.0 : DISPLAY STRING- (ascii): ORA IT Group
```

The *-c public* switch passes the community string *public* to the *snmpget* command.



Keep in mind that your devices shouldn't use the same (default) community strings that are used within this book. In addition, using the same string for the read-only (*snmpget*) and read-write (*snmpset*) communities is a poor idea.

Now let's run the OpenView *snmpset* command. This command takes the value specified in quotes on the command line and uses it

to set the object indicated by the given OID. Use the same OID (system.sysContact.0). Since the new value for sysContact contains words and possibly numbers, we must also specify the variable type `octetstring`.^[5] Run the OpenView `snmpset` command with the following parameters:

^[5] If you read RFC 1213 (MIB-II) you will note that sysLocation has a SYNTAX of DisplayString. This is really a textual convention of type OCTET STRING with a size of 0..255 octets.

```
$ /opt/OV/bin/snmpset -c private orarouter1 .1.3.6.1.2.1.1.4.0 \
octetstring "Meg A. Byte 555-1212"
system.sysContact.0 : DISPLAY STRING- (ascii): Meg A. Byte 555-1212
```

The result shows that `snmpset` successfully changed the router's contact person to Meg A. Byte 555-1212. If you don't see this result, the set was not successful. [Table 8-2](#) shows some of the common error messages you might receive, and steps you can take to correct the problems. To confirm the value the device has stored in sysContact, we can repeat the `snmpget` command.

If we use OpenView's GUI, things start to get a bit easier to see, set, and confirm. Use the GUI to get the value of sysContact. Once you have confirmed that a value is there, type a description in the SNMP Set Value text box. Since there is only one instance for sysContact, you have to insert a 0 (zero) for the MIB Instance. After you have completed all the required input items, click on the "Set" button located to the right of the "SNMP Set Value" text box. You should see a pop-up window that reads "Set has completed successfully." To verify that the set actually occurred, click on "Start Query." (It should be apparent to you by now that using a GUI such as OpenView's MIB Browser program makes getting and setting MIB objects much easier.)

To show how this can be done programmatically, we will write another small Perl script, named `snmpset.pl`:

```
#!/usr/local/bin/perl
#filename: /opt/local/perl_scripts/snmpset.pl
use SNMP_util;
$MIB1 = ".1.3.6.1.2.1.1.6.0";
$HOST = "oraswitch2";
$LOC = "@ARGV";
($value) = &snmpset("private\@$HOST", "$MIB1", 'string', "$LOC");
if ($value) { print "Results :$MIB1: :$value:\n"; }
else { warn "No response from host :$HOST:\n"; }
```

Let's run this script:

```
$ /opt/local/perl_scripts/snmpset.pl A bld JM-10119 floor 7
Results :.1.3.6.1.2.1.1.6.0: :A bld JM-10119 floor 7:
```

Using the `snmpget.pl` script, we can verify that the set took place:

```
$ /opt/local/perl_scripts/snmpget.pl .1.3.6.1.2.1.1.6.0 public@oraswitch2
Results :.1.3.6.1.2.1.1.6.0: :A bld JM-10119 floor 7:
```

Now we'll use the Net-SNMP *snmpset* utility to change the system contact:

```
$ snmpset oraswitch2 private sysContact.0 s myself
```

```
system.sysContact.0 = myself
```

```
$ snmpget oraswitch2 public sysContact.0
```

```
system.sysContact.0 = myself
```

There's nothing really confusing here. We supplied a community string, a hostname, and an object ID, followed by a datatype (*s* for *String*) and the new value of *sysContact*. Just to convince ourselves that the set actually took place, we followed it with an *snmpget*. The only additional thing you need to know is the mechanism for specifying datatypes. Net-SNMP uses the single-character abbreviations shown in [Table 8-1](#).

Table 8-1. Net-SNMP Datatype Abbreviations	
Abbreviation	Meaning
a	IP address
b ^[6]	Bits
d	Decimal string
D	Double
F	Float
i	Integer
I	Signed int64
n	Null
o	Object ID
s	String
t	Time ticks
u	Unsigned integer
U	Unsigned int64
x	Hexadecimal string

^[6] While the manpages show this as a valid datatype, the help output from the command does not.

8.4 Error Responses

[Table 8-2](#) shows the error responses that a device might return while executing the commands presented in this chapter. Consult your local documentation if these explanations do not cover your exact problem.

Table 8-2. Error Response Table	
Server Responded with	Explanation
Contained under subtree	<i>snmpwalk</i> returns this error if you have tried going down a MIB and are already at the end, or if the tree doesn't exist on the client.
No response arrived before timeout	Possible causes include invalid community name, agent is not running, or the node is inaccessible.
Agent reported error with variable	You are trying to set to an object with a datatype that is not the same as (or close to) the variable's specified type. For example, if the variable wants a <code>DisplayString</code> , you'll get this error if you send it an <code>INTEGER</code> . Read through the MIB to see what <code>SYNTAX</code> type the variable needs.
Missing instance value for . . .	When you are setting a value, you must supply the entire OID and instance. A scalar object will end with zero (0) and a tabular object will end with the instance number of the object in a table. Verify that the instance number you're using with <i>snmpget</i> is correct and retry your <i>set</i> .
Access is denied for variable	This may happen if you are trying to set a value on a read-only object. Review the MIB to see what the object's <code>ACCESS</code> setting is.

Chapter 9. Polling and Thresholds

SNMP gives you the ability to poll your devices regularly, collecting their management information. Furthermore, you can tell the NMS that there are certain thresholds that, if crossed, require some sort of action. For example, you might want to be notified if the traffic at an interface jumps to an extremely high (or low) value; that event might signal a problem with the interface, or insufficient capacity, or even a hostile attack on your network. When such a condition occurs, the NMS can forward an alarm to an event-correlation engine or have an icon on an OpenView map flash. To make this more concrete, let's say that the NMS is polling the status of an interface on a router. If the interface goes down, the NMS

reports what has happened so the problem can be quickly resolved.

SNMP can perform either internal or external polling. *Internal* polling is typically used in conjunction with an application that runs as a daemon or a facility such as *cron* that periodically runs a local application. *External* polling is done by the NMS. The OpenView NMS provides a great implementation of external polling; it can graph and save your data for later retrieval or notify you if it looks like something has gone wrong. Many software packages make good NMSs, and if you're clever about scripting you can throw together an NMS that's fine-tuned to your needs. In this chapter, we will look at a few of the available packages.

Polling is like checking the oil in a car; this analogy may help you to think about appropriate polling strategies. Three distinct items concern us when checking the oil: the physical process (opening the hood, pulling out the dipstick, and putting it back in); the preset gauge that tells us if we have a problem (is the level too high, too low, or just right?); and the frequency with which we check it (once an hour, week, month, or year?).

Let's assume that you ask your mechanic to go to the car and check the oil level. This is like an NMS sending a packet to a router to perform an *snmpget* on some piece of information. When the mechanic is finished, you pay him \$30 and go on your way. Because a low oil level may result in real engine damage, you want to check the oil regularly. So how long should you wait until you send the mechanic out to the car again? Checking the oil has a cost: in this scenario, you paid \$30. In networks, you pay with bandwidth. Like money, you have only so much bandwidth, and you can't spend it frivolously. So the real question is, how long can you wait before checking the oil again without killing your budget?

The answer lies within the car itself. A finely tuned racing car needs to have its fluids at perfect levels. A VW Beetle,^[1] unlike a racecar, can have plus or minus a quart at any time without seriously hindering its performance. You're probably not driving a Beetle, but you're probably not driving a racecar either. So you decide that you can check the oil level about every three weeks. But how will you know what is low, high, or just right?

^[1] The old ones from the 1960s, not the fancy modern ones.

The car's dipstick tells you. Your mechanic doesn't need to know the car model, engine type, or even the amount of oil in the car; he only needs to know what value he gets when he reads the dipstick. On a network, a device's dipstick is called an agent, and the dipstick reading is the SNMP response packet.

All SNMP-compatible devices contain standardized agents (dipsticks) that can be read by any mechanic (NMS). It is important to keep in mind that the data gathered is only as good as the agent, or mechanic, that generated it.

In both cases, some predefined threshold determines the appropriate action. In the oil example, the threshold is "low oil," which triggers an automatic response: add oil. (Crossing the "high oil" threshold might trigger a different kind of response.) If we're talking about a router interface, the possible values we might receive are "up" and "down." Imagine that your company's gateway to the Internet, a port on a router, must stay up 24 hours a day, 7 days a week. If that port goes down, you could lose \$10,000 for each second it stays down. Would you check that port often? Most organizations won't pay someone to check router interfaces every hour, let alone every second. Even if you had the time, that wouldn't be fun, right? This is where SNMP polling comes in. It allows network managers to guarantee that mission-critical devices are up and functioning properly, without having to pay someone to constantly monitor routers, servers, etc.

Once you determine your monitoring needs, you can specify at what interval you would like to poll a device or set of devices. This is typically referred to as the *poll interval*, and can be as granular as you like (e.g., every second, every hour, etc.). The threshold value at which you take action doesn't need to be binary: you might decide that something's obviously wrong if the number of packets leaving your Internet connection falls below a certain level.



Whenever you are figuring out how often to poll a device, remember to keep three things in mind: the device's agent/CPU, bandwidth consumption, and the types of values you are requesting. Some values you receive may be 10-minute averages. If this is the case, it is a waste to poll every few seconds. Review the MIBs surrounding the data for which you are polling. My preference is to start polling fairly often. Once I see the trends and peak values, I back off. This can add congestion to the network but ensures that I don't miss any important information.

Whatever the frequency at which you poll, keep in mind other things that may be happening on the network. Be sure to stagger your polling times to avoid other events if possible. Keep in mind backups, data loads, routing updates, and other events that can cause stress on your networks or CPUs.

9.1 Internal Polling

It may seem like a waste of bandwidth to poll a device just to find out that everything is okay. On a typical day, you may poll dozens of devices hundreds or thousands of times without discovering any failures or outages. Of course, that's really what you want to find out -- and you'll probably conclude that SNMP has served its purpose the first time you discover a failed device and get the device back online before users have had a chance to start complaining. However, in the best of all possible worlds, you'd get the benefits of polling without the cost: that is, without devoting a significant chunk of your network's bandwidth to monitoring its health.

This is where internal polling comes in. As its name implies, internal polling is performed by an agent that is internal, or built in, to the device you want to manage. Since polling is internal to the device, it doesn't require traffic between the agent and your NMS. Furthermore, the agent doing the polling does not have to be an actual SNMP agent, which can allow you to monitor systems (either machines or software) that do not support SNMP. For example, some industrial-strength air-conditioning-equipment vendors provide operational status information via a serial port. If the air-conditioning unit is attached to a terminal server or similar device, it becomes easy to use scripting languages to monitor the unit and generate traps if the temperature exceeds a certain threshold. This internal program can be written in your favorite scripting language, and it can check any status information to which you can get access. All you need is a way to get data from the script to the management station.

One strategy for writing a polling program is to use "hooks" within a program to extract information that can then be fed into an SNMP trap and sent to the NMS. We will cover traps more in [Chapter 10](#). Another way to do internal polling is to use a program (e.g., sh, Perl, or C) that is run at set intervals. (On Unix, you would use *cron* to run a program at fixed intervals; there are similar services on other operating systems.) Hooks and *cron*-driven scripts both allow you to check internal variables and report errors as they are found. Here is a Perl script that checks for the existence of a file and sends a trap if the file is not found:

```
#!/usr/local/bin/perl
# Filename: /opt/local/perl_scripts/check4file.pl

use SNMP_util "0.54"; # This will load the BER and SNMP_Session modules for us

$FILENAME = "/etc/passwd";

#
# if the /etc/passwd file does not exist, send a trap!
#
if(!(-e $FILENAME)) {
    snmptrap("public@nms:162", ".1.3.6.1.4.1.2789", "sunserver1", 6, 1547, \
```

```

        ".1.3.6.1.4.1.2789.1547.1", "string", "File \:$FILENAME\: Could\
        NOT Be Found");
    }

```

Here is what the Sun-style *crontab* looks like:

```
$ crontab -l
```

```

# Check for this file every 15 minutes and report trap if not found
4,19,34,49 * * * * /opt/local/perl_scripts/check4file.pl

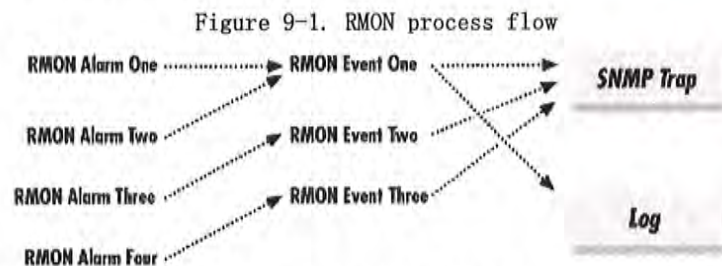
```

Notice that we poll four minutes after each quarter hour, rather than on the quarter hour. The next poll we insert into the *crontab* file may run five minutes after the quarter hour (5,20,35,50). This practice prevents us from starting a huge number of programs at the same time. It's a particularly good idea to avoid polling on the hour -- that's a popular time for random programs and *cron* jobs to start up. Consult the *cron* manpage if you are unfamiliar with its operation.

9.1.1 Remote Monitoring (RMON)

RMON is a supplement to the MIB-II group. This group, if supported by the device's SNMP agent, allows us to do both internal and external polling. We can poll devices through a remote NMS (external polling) or have the local RMON agent check itself periodically and report any errors (internal polling). The RMON agent will send traps when error conditions are found.

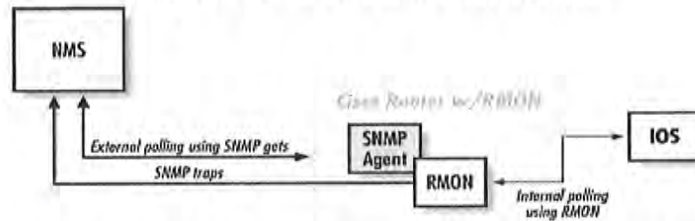
Many devices support RMON, making it an effective mechanism for internal polling. For example, Cisco supports the Events and Alarms RMON categories. You can configure the Alarms category to poll MIBs internally and react in different ways when a rising or falling threshold occurs. Each threshold has the option of calling an internal Event. [Figure 9-1](#) shows the flow that these two RMON categories take.



The distinction between alarms and events is important. Each alarm is tied to a specific event, which defines what action to perform when the alarm goes off. Once a threshold is met, triggering an alarm, the alarm calls the event, which can perform additional functions, including sending traps to the NMS and writing a record in a log. Standard SNMP traps are preconfigured by the agent's vendor, which gives network managers no control over setting any kind of thresholds; however, RMON allows a network manager to set rising and

falling thresholds. [Figure 9-2](#) represents the interaction between a router's RMON agent and an NMS.

Figure 9-2. RMON and NMS interaction



In [Figure 9-2](#), the Cisco router's SNMP agent forwards a trap to the NMS. Notice the direction of communication: RMON trap transmission is unidirectional. The NMS receives the trap from the Cisco router and decides what action to take, if any.

In addition to sending traps, we can also log events; if we so choose, we can even log the event without generating a trap. Logging can be particularly useful when you are initially configuring RMON alarms and events. If you make your alarm conditions too sensitive, you can clog your NMS with trigger-happy RMON events. Logging can help you fine-tune your RMON alarms before they are released into production.

9.1.1.1 RMON configuration

As a practical example of how to configure RMON, we will use Cisco's RMON implementation, starting with events. The following IOS command defines an RMON event:

```
rmon event number [log] [trap community] [description string] [owner string]
```

If you're familiar with IOS, you should be expecting a corresponding `no` command that discards an RMON event:

```
no rmon event number
```

The parameters to these IOS commands are:

number

Specifies the unique identification number for the event. This value must be greater than 0; a value of 0 is not allowed.

log

Tells the agent to log the entry when triggered. This argument is optional.

trap community

Specifies the trap community string; i.e., a community string to be included with the trap. Many network-management programs can be configured to respond only to traps with a particular community string.

description string

Describes the event.

owner string

Ties the event or item to a particular person.

Here are two examples of how to create Cisco RMON events. The first line creates a rising alarm, which facilitates sending a trap to the NMS. The second creates a falling alarm that might indicate that traffic has returned to an acceptable level (this alarm is logged, but doesn't generate a trap):

```
(config)#rmon event 1 log trap public description "High ifInOctets" owner dmauro
(config)#rmon event 2 log description "Low ifInOctets" owner dmauro
```

You can also use logging to keep track of when the events were called. Though you can configure traps without logging, what happens if the line to your NMS goes down? Logging ensures that you don't lose information when the NMS is disabled. We suggest using both `log` and `trap` on all your events. You can view the logs of your RMON events by issuing the following command on the router:

```
orarouter1# show rmon event
```

```
Event 1 is active, owned by dmauro
Description is High ifInOctets
Event firing causes log and trap to community public, last fired 00:05:04
Current log entries:
      index      time  description
        1  00:00:31  High ifInOctets
        2  00:05:04  High ifInOctets
Event 2 is active, owned by dmauro
Description is Low ifInOctets
Event firing causes log, last fired 00:00:11
Current log entries:
      index      time  description
        1  00:00:11  Low ifInOctets
```

The following command walks the rmon event table, which displays the values we just set:

```
$ snmpwalk orarouter1 .iso.org.dod.internet.mgmt.mib-2.rmon.event.eventTable
rmon.event.eventTable.eventEntry.eventIndex.1 : INTEGER: 1
rmon.event.eventTable.eventEntry.eventIndex.2 : INTEGER: 2
rmon.event.eventTable.eventEntry.eventDescription.1
          : DISPLAY STRING- (ascii): High ifInOctets
rmon.event.eventTable.eventEntry.eventDescription.2
          : DISPLAY STRING- (ascii): Low ifInOctets
rmon.event.eventTable.eventEntry.eventType.1 : INTEGER: log-and-trap
rmon.event.eventTable.eventEntry.eventType.2 : INTEGER: log
rmon.event.eventTable.eventEntry.eventCommunity.1 : OCTET STRING- (ascii): public
rmon.event.eventTable.eventEntry.eventCommunity.2 : OCTET STRING- (ascii):
rmon.event.eventTable.eventEntry.eventLastTimeSent.1 : Timeticks: (0) 0:00:00.00
rmon.event.eventTable.eventEntry.eventLastTimeSent.2 : Timeticks: (0) 0:00:00.00
rmon.event.eventTable.eventEntry.eventOwner.1 : DISPLAY STRING- (ascii): dmauro
rmon.event.eventTable.eventEntry.eventOwner.2 : DISPLAY STRING- (ascii): dmauro
rmon.event.eventTable.eventEntry.eventStatus.1 : INTEGER: valid
rmon.event.eventTable.eventEntry.eventStatus.2 : INTEGER: valid
```

Most of the information we set on the command line is available through SNMP. We see two events, with indexes 1 and 2. The

first event has the description `High ifInOctets`; it is logged and a trap is generated; the community string for the event is `public`; the event's owner is `dmauro`; the event is `valid`, which essentially means that it is enabled; and we also see that the event has not yet occurred. Instead of using the command line to define these events, we could have used `snmpset` either to create new events or to modify events we already have. If you take this route, keep in mind that you must set the `eventEntry.eventStatus` to 1, for "valid," for the event to work properly.



You can poll the objects `ifDescr` and `ifType` in the `mgmt.interfaces.ifEntry` subtree to help you identify which instance number you should use for your devices. If you are using a device with multiple ports, you may need to search the `ifType`, `ifAdminStatus`, and `ifOperStatus` to help you identify what's what. In [Section 9.2](#), we will see that it is not necessary to keep track of these MIB variables (the external polling software takes care of this for us).

Now that we have our events configured, let's start configuring alarms to do some internal polling. We need to know what we are going to poll, what type of data is returned, and how often we should poll. Assume that the router is our default gateway to the Internet. We want to poll the router's second interface, which is a serial interface. Therefore, we want to poll `mgmt.interfaces.ifEntry.ifInOctets.2` to get the number of outbound octets on that interface, which is an `INTEGER` type.^[2] To be precise, the `ifInOctets` MIB object is defined as "The total number of octets received on the interface, including framing characters." (The `.2` at the end of the OID indicates the second entry in the `ifEntry` table. On our router, this denotes the second interface, which is the one we want to poll.) We want to be notified if the traffic on this interface exceeds 90,000 octets/second; we'll assume things are back to normal when the traffic falls back under 85,000 octets/second. This gives us the rising and falling thresholds for our alarm. Next, we need to figure out the interval at which we are going to poll this object. Let's start by polling every 60 seconds.

^[2] From RFC 1757, the `alarmVariable` (the object/MIB we are going to poll) needs to resolve to an ASN.1 primitive type of `INTEGER`, `Counter`, `Gauge`, or `TimeTicks`.

Now we need to put all this information into a Cisco `RMON` *alarm* command. Here is the command to create an alarm:

```
rmmon alarm number variable interval {delta | absolute}
    rising-threshold value [event-number]
    falling-threshold value [event-number]
```

[owner string]

The following command discards the alarm:

no rmon alarm number

The parameters to these commands are:

number

Specifies the unique identification number assigned to the alarm.

variable

Specifies which MIB object to monitor.

interval

Specifies the frequency at which the alarm monitors the MIB variable.

delta

Indicates that the threshold values given in the command should be interpreted in terms of the difference between successive readings.

absolute

Indicates that the threshold values given in the command should be interpreted as absolute values; i.e., the difference between the current value and preceding values is irrelevant.

rising-threshold value event-number

Specifies the value at which the alarm should be triggered, calling the event, when the value is rising. event-number is the event that should be called when the alarm occurs. The event number is optional because the threshold doesn't have to be assigned an event. If either of the two thresholds is left blank the event number will be set to 0, which does nothing.

falling-threshold value event-number

Specifies the value at which the alarm should be triggered, calling the event, when the value is falling. event-number is the event that should be called when the alarm occurs. The event number is optional because the threshold doesn't have to be assigned an event. If either of the two thresholds is left blank the event number will be set to 0, which does nothing.

owner string

Ties this alarm to a particular person.

To configure the alarm settings we just described, enter the following command, in configuration mode, on a Cisco console:

```
orarouter1(config)#rmon alarm 25 ifEntry.10.2 60 absolute \  
rising-threshold 90000 1 falling-threshold 85000 2 owner dmauro
```

This command configures alarm number 25, which monitors the object in ifEntry.10.2 (instance 2 of ifEntry.ifInOctets, or the input octets on interface 2) every 60 seconds. It has a rising threshold of 90,000 octets, which has event number 1

tied to it: event 1 is called when traffic on this interface exceeds 90,000 octets/second. The falling threshold is set to 85,000 octets and has event number 2 tied to it. Here's how the alarm looks in the router's internal tables:

```
orarouter1#show rmon alarm
```

```
Alarm 1 is active, owned by dmauro
Monitors ifEntry.10.2 every 60 second(s)
Taking absolute samples, last value was 87051
Rising threshold is 90000, assigned to event 1
Falling threshold is 85000, assigned to event 2
On startup enable rising or falling alarm
```

The last line of output says that the router will enable the alarm upon reboot. As you'd expect, you can also look at the alarm settings through the RMON MIB, beginning with the subtree 1.3.6.1.2.1.16. As with the events themselves, we can create, change, edit, and delete entries using *snmpset*.

One problem with internal polling is that getting trends and seeing the data in a graph or table is difficult. Even if you develop the backend systems to gather MIB objects and display them graphically, retrieving data is sometimes painful. The Multi Router Traffic Grapher (MRTG) is a great program that allows you to do both internal and external polling. Furthermore, it is designed to generate graphs of your data in HTML format. MRTG is covered in [Chapter 13](#).

9.2 External Polling

It is often impossible to poll a device internally, for technical, security, or political reasons. For example, the System Administration group may not be in the habit of giving out the root password, making it difficult for you to install and maintain internal polling scripts. However, they may have no problem with installing and maintaining an SNMP agent such as Concord's SystemEDGE or Net-SNMP. It's also possible that you will find yourself in an environment in which you lack the knowledge to build the tools necessary to poll internally. Despite the situation, if an SNMP agent is present on a machine that has objects worth polling, you can use an external device to poll the machine and read the objects' values.^[3] This external device can be one or more NMSs or other machines or devices. For instance, when you have a decent-sized network it is sometimes convenient, and possibly necessary, to distribute polling among several NMSs.

^[3] Many devices say they are SNMP-compatible but support only a few MIBs. This makes polling nearly impossible. If you don't have the object(s) to poll there is nothing you can do, unless there are hooks for an extensible agent. Even with extensible

agents, unless you know how to program, the Simple in SNMP goes away fast.

Each of the external polling engines we will look at uses the same polling methods, although some NMSs implement external polling differently. We'll start with the OpenView *xnmgraph* program, which can be used to collect and display data graphically. You can even use OpenView to save the data for later retrieval and analysis. We'll include some examples that show how you can collect data and store it automatically and how you can retrieve that data for display. Castle Rock's SNMPc also has an excellent data-collection facility that we will use to collect and graph data.

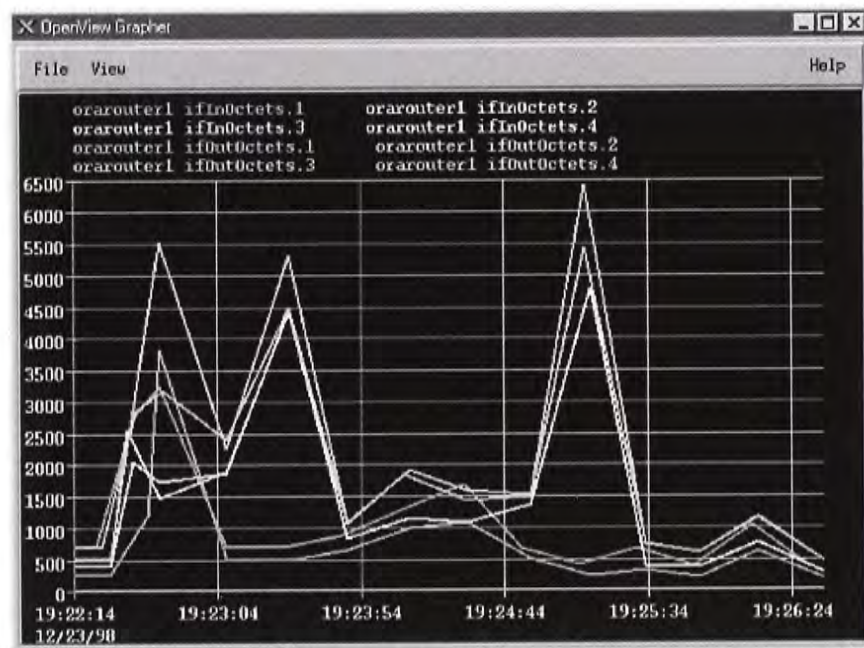
9.2.1 Collecting and Displaying Data with OpenView

One of the easiest ways to get some interesting graphs with OpenView is to use the *xnmgraph* program. You can run *xnmgraph* from the command line and from some of NNM's menus. One practical way to graph is to use OpenView's *xnmbrowser* to collect some data and then click "Graph." It's as easy as that. If the node you are polling has more than one instance (say, multiple interfaces), OpenView will graph all known instances. When an NMS queries a device such as a router, it determines how many instances are in the *ifTable* and retrieves management data for each entry in the table.

9.2.2 OpenView Graphing

[Figure 9-3](#) shows the sort of graph you can create with NNM. To create this graph, we started the browser ([Figure 8-2](#)) and clicked down through the MIB tree until we found the *.iso.org.dod.internet.mgmt.mib-2.interfaces.ifTable.ifEntry* list. Once there, we clicked on *ifInOctets*; then, while holding down the Ctrl key, we clicked on *ifOutOctets*. After both were selected and we verified that the "Name or IP Address" field displayed the node we wanted to poll, we clicked on the "Graph" button.

Figure 9-3. OpenView *xnmgraph* of octets in/out



Once the graph has started, you can change the polling interval and the colors used to display different objects. You can also turn off the display of some or all of the object instances. The menu item "View → Line Configuration" lets you specify which objects you would like to display; it can also set multipliers for different items. For example, to display everything in K, multiply the data by .001. There is also an option ("View → Statistics") that shows a statistical summary of your graph. [Figure 9-4](#) shows some statistics from the graph in [Figure 9-3](#). While the statistics menu is up, you can left-click on the graph; the statistics window will display the values for the specific date and time to which you are pointing with the mouse.

Figure 9-4. xnmgraph statistics

Line	Minimum	Average	Maximum	Last Value
orarouter1 ifInOctets.1	375.16	793.35	2069.66	2069.66
orarouter1 ifInOctets.2	276.98	1106.54	4778.98	1544.64
orarouter1 ifInOctets.3	0.00	0.00	0.00	0.00
orarouter1 ifInOctets.4	275.37	1105.79	4777.44	1540.51
orarouter1 ifOutOctets.1	167.64	473.00	1448.99	1448.99
orarouter1 ifOutOctets.2	452.16	1609.71	5421.61	2586.47
orarouter1 ifOutOctets.3	0.00	0.00	0.00	0.00
orarouter1 ifOutOctets.4	460.98	1495.76	5441.12	2482.46



Starting *xnmgraph* from the command line allows you to start the grapher at a specific polling period and gives you several other options. By default, OpenView polls at 10-second intervals. In most cases this is fine, but if you are polling a multiport router to check if some

ports are congested, a 10-second polling interval may be too quick and could cause operational problems. For example, if the CPU is busy answering SNMP queries every 10 seconds, the router might get bogged down and become very slow, especially if the router is responsible for OSPF or other CPU-intensive tasks. You may also see messages from OpenView complaining that another poll has come along while it is still waiting for the previous poll to return. Increasing the polling interval usually gets rid of these messages.

Some of NNM's default menus let you use the grapher to poll devices depending on their type. For example, you can select the object type "router" on the NNM and generate a graph that includes all your routers. Whether you start from the command line or from the menu, there are times when you will get a message back that reads "Requesting more lines than number of colors (25). Reducing number of lines." This message means that there aren't enough colors available to display the objects you are trying to graph. The only good ways to avoid this problem are to break up your graphs so that they poll fewer objects or to eliminate object instances you don't want. For example, you probably don't want to graph router interfaces that are down (for whatever reason) and other "dead" objects. We will soon see how you can use a regular expression as one of the arguments to the *xnmgraph* command to graph only those interfaces that are up and running.

Although the graphical interface is very convenient, the command-line interface gives you much more flexibility. The following script displays the graph in [Figure 9-3](#) (i.e., the graph we generated through the browser):

```
#!/bin/sh
# filename: /opt/OV/local/scripts/graph0ctets
# syntax: graph0ctets <hostname>
/opt/OV/bin/xnmgraph -c public -mib \
".iso.org.dod.internet.mgmt.mib-2.interfaces.ifTable.ifEntry.ifInOctets:,,,,,,\
.iso.org.dod.internet.mgmt.mib-2.interfaces.ifTable.ifEntry.ifOutOctets:,,,,,,\"
$1
```

You can run this script with the command:

```
$ /opt/OV/local/scripts/graph0ctets orarouter1
```

The worst part of writing the script is figuring out what command-line options you want -- particularly the long strings of nine colon-separated options. All these options give you the ability to refine what you want to graph, how often you want to poll the objects, and how you want to display the data. (We'll discuss the syntax of these options as we go along, but for the complete story, see the *xnmgraph*(1) manpage.) In this script,

we're graphing the values of two MIB objects, `ifInOctets` and `ifOutOctets`. Each OID we want to graph is the first (and in this case, the only) option in the string of colon-separated options. On our network, this command produces eight traces: input and output octets for each of our four interfaces. You can add other OIDs to the graph by adding sets of options, but at some point the graph will become too confusing to be useful. It will take some experimenting to use the `xnmgraph` command efficiently, but once you learn how to generate useful graphs you'll wonder how you ever got along without it.



Keeping your scripts neat is not only good practice, but also aesthetically pleasing. Using a `"\"` at the end of a line indicates that the next line is a continuation of the current line. Breaking your lines intelligently makes your scripts more readable. Be warned that the Unix shells do not like extra whitespace after the `"\"`. The only character after each `"\"` should be one carriage return.

Now, let's modify the script to include more reasonable labels -- in particular, we'd like the graph to show which interface is which, rather than just showing the index number. In our modified script, we've used numerical object IDs, mostly for formatting convenience, and we've added a sixth option to the ugly sequence of colon-separated options: `.1.3.6.1.2.1.2.2.1.2` (this is the `ifDescr`, or interface description, object in the interface table). This option says to poll each instance and use the return value of `snmpget .1.3.6.1.2.1.2.2.1.2.INSTANCE` as the label. This should give us meaningful labels. Here's the new script:

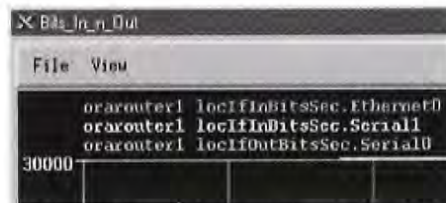
```
#!/bin/sh
# filename: /opt/OV/local/scripts/graphOctets
# syntax: graphOctets <hostname>
/opt/OV/bin/xnmgraph -c public -title Bits_In_n_Out -mib \
".1.3.6.1.4.1.9.2.2.1.1.6:::.1.3.6.1.2.1.2.2.1.2::,\
.1.3.6.1.4.1.9.2.2.1.1.8:::.1.3.6.1.2.1.2.2.1.2::" $1
```

To see what we'll get for labels, here's the result of walking `.1.3.6.1.2.1.2.2.1.2`:

```
$ snmpwalk orarouter1 .1.3.6.1.2.1.2.2.1.2
interfaces.ifTable.ifEntry.ifDescr.1 : DISPLAY STRING- (ascii): Ethernet0
interfaces.ifTable.ifEntry.ifDescr.2 : DISPLAY STRING- (ascii): Serial0
interfaces.ifTable.ifEntry.ifDescr.3 : DISPLAY STRING- (ascii): Serial1
```

[Figure 9-5](#) shows our new graph. With the addition of this sixth option, the names and labels are much easier to read.

Figure 9-5. OpenView `xnmgraph` with new labels



Meaningful labels and titles are important, especially if management is interested in seeing the graphs. A label that contains an OID and not a textual description is of no use. Some objects that are useful in building labels are `ifType` (.1.3.6.1.2.1.2.2.1.3) and `ifOperStatus` (.1.3.6.1.2.1.2.2.1.8). Be careful when using `ifOperStatus`; if the status of the interface changes during a poll, the label does not change. The label is evaluated only once.

One of the most wasteful things you can do is poll a useless object. This often happens when an interface is administratively down or not configured. Imagine that you have 20 serial interfaces, but only one is actually in use. If you are looking for octets in and out of your serial interfaces, you'll be polling 40 times and 38 of the polls will always read 0. OpenView's `xnmgraph` allows you to specify an OID and regular expression to select what should be graphed. To put this feature to use, let's walk the MIB to see what information is available:

```
$ snmpwalk orarcouter1 .1.3.6.1.2.1.2.2.1.8
interfaces.ifTable.ifEntry.ifOperStatus.1 : INTEGER: up
interfaces.ifTable.ifEntry.ifOperStatus.2 : INTEGER: up
interfaces.ifTable.ifEntry.ifOperStatus.3 : INTEGER: down
```

This tells us that only two interfaces are currently up. By looking at `ifDescr`, we see that the live interfaces are Ethernet0 and Serial0; Serial1 is down. Notice that the type of `ifOperStatus` is `INTEGER`, but the return value looks like a string. How is this? RFC 1213 defines string values for each possible return value:

```
ifOperStatus OBJECT-TYPE
    SYNTAX  INTEGER {
        up(1),          -- ready to pass packets
        down(2),
        testing(3)     -- in some test mode
    }
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The current operational state of the interface. The testing(3)
        state indicates that no operational packets can be passed."
    ::= { ifEntry 8 }
```

It's fairly obvious how to read this: the integer value 1 is converted to the string `up`. We can therefore use the value 1 in a regular expression that tests `ifOperStatus`. For every instance we will check the value of `ifOperStatus`; we will poll

that instance and graph the result only if the status returns 1. In pseudocode, the operation would look something like this:

```
if (ifOperStatus == 1) {
    pollForMIBData;
    graphOctets;
}
```

Here's the next version of our graphing script. To put this logic into a graph, we use the OID for ifOperStatus as the fourth colon option, and the regular expression (1) as the fifth option:

```
#!/bin/sh
# filename: /opt/OV/local/scripts/graphOctets
# syntax: graphOctets <hostname>
/opt/OV/bin/xnmgraph -c public \
-title Octets_In_and_Out_For_All_Up_Interfaces \
-mib ".1.3.6.1.2.1.2.2.1.10:::.1.3.6.1.2.1.2.2.1.8:1:::, \
.1.3.6.1.2.1.2.2.1.16:::.1.3.6.1.2.1.2.2.1.8:1:::" $1
```

This command graphs the ifInOctets and ifOutOctets of any interface that has a current operational state equal to 1, or up. It therefore polls and graphs only the ports that are important, saving on network bandwidth and simplifying the graph. Furthermore, we're less likely to run out of colors while making the graph because we won't assign them to useless objects. Note, however, that this selection happens only during the first poll and stays effective throughout the entire life of the graphing process. If the status of any interface changes after the graph has been started, nothing in the graph will change. The only way to discover any changes in interface status is to restart *xnmgraph*.

Finally, let's look at:

- How to add a label to each of the OIDs we graph
- How to multiply each value by a constant
- How to specify the polling interval

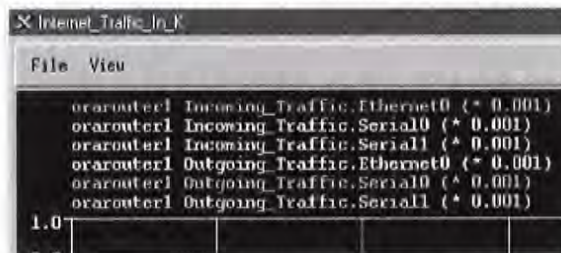
The cropped graph in [Figure 9-6](#) shows how the labels change when we run the following script:

```
#!/bin/sh
# filename: /opt/OV/local/scripts/graphOctets
# syntax: graphOctets <hostname>
/opt/OV/bin/xnmgraph -c public -title Internet_Traffic_In_K -poll 68 -mib \
".1.3.6.1.4.1.9.2.2.1.1.6:Incoming_Traffic:::.1.3.6.1.2.1.2.2.1.2:::001:, \
.1.3.6.1.4.1.9.2.2.1.1.8:Outgoing_Traffic:::.1.3.6.1.2.1.2.2.1.2:::001:" \
$1
```

The labels are given by the second and sixth fields in the colon-separated options (the second field provides a textual label to identify the objects we're graphing and the sixth uses the ifDescr field to identify the particular interface); the

constant multiplier (.001) is given by the eighth field; and the polling interval (in seconds) is given by the `-poll` option.

Figure 9-6. `xnmgraph` with labels and multipliers



By now it should be apparent how flexible OpenView's `xnmgraph` program really is. These graphs can be important tools for troubleshooting your network. When a network manager receives complaints from customers regarding slow connections, he can look at the graph of `ifInOctets` generated by `xnmgraph` to see if any router interfaces have unusually high traffic spikes.

Graphs like these are also useful when you're setting thresholds for alarms and other kinds of traps. The last thing you want is a threshold that is too trigger-y (one that goes off too many times) or a threshold that won't go off until the entire building burns to the ground. It's often useful to look at a few graphs to get a feel for your network's behavior before you start setting any thresholds. These graphs will give you a baseline from which to work. For example, say you want to be notified when the battery on your UPS is low (which means it is being used) and when it is back to normal (fully charged). The obvious way to implement this is to generate an alarm when the battery falls below some percentage of full charge, and another alarm when it returns to full charge. So the question is: what value can we set for the threshold? Should we use 10% to indicate that the battery is being used and 100% to indicate that it's back to normal? We can find the baseline by graphing the device's MIBs.^[5] For example, with a few days' worth of graphs, we can see that our UPS's battery stays right around 94-97% when it is not in use. There was a brief period when the battery dropped down to 89%, when it was performing a self-test. Based on these numbers, we may want to set the "in use" threshold at 85% and the "back to normal" threshold at 94%. This pair of thresholds gives us plenty of notification when the battery's in use, but won't generate useless alarms when the device is in self-test mode. The appropriate thresholds depend on the type of devices you are polling, as well as the MIB data that is gathered. Doing some initial testing and polling to get a baseline (normal numbers) will help you set thresholds that are meaningful and useful.

^[5] Different vendors have different UPS MIBs. Refer to your particular vendor's MIB to find out which object represents low battery power.

Before leaving *xnmgraph*, we'll take a final look at the nastiest aspect of this program: the sequence of nine colon-separated options. In the examples, we've demonstrated the most useful combinations of options. In more detail, here's the syntax of the graph specification:

object:label:instances:match:expression:instance-label:truncator:multiplier:nodes

The parameters are:

object

The OID of the object whose values you want to graph. This can be in either numeric or human-readable form, but it should not have an instance number at the end. It can also be the name of an expression (expressions are discussed in [Appendix A](#)).

label

A string to use in making the label for all instances of this object. This can be a literal string or the OID of some object with a string value. The label used on the graph is made by combining this label (for all instances of the object) with instance-label, which identifies individual instances of an object in a table. For example, in [Figure 9-6](#), the labels are Incoming_Traffic and Outgoing_Traffic; instance-label is 1.3.6.1.2.1.2.2.1.2, or the ifDescr field for each object being graphed.

instances

A regular expression that specifies which instances of object to graph. If this is omitted, all instances are graphed. For example, the regular expression 1 limits the graph to instance 1 of object; the regular expression [4-7] limits the graph to instances 4 through 7. You can use the match and expression fields to further specify which objects to match.

match

The OID of an object (not including the instance ID) to match against a regular expression (the match-expression), to determine which instances of the object to display in the graph.

expression

A regular expression; for each instance, the object given by match is compared to this regular expression. If the two match, the instance is graphed.

instance-label

A label to use to identify particular instances of the object you are graphing. This is used in combination with the label and truncator fields to create a label for each instance of each object being graphed.

truncator

A string that will be removed from the initial portion of the instance label, to make it shorter.

multiplier

A number that's used to scale the values being graphed.

nodes

The nodes to poll to create the graph. You can list any number of nodes, separated by spaces. The wildcard "*" polls all the nodes in OpenView's database. If you omit this field, *xnmgraph* takes the list of nodes from the final argument on the command line.

The only required field is object; however, as we've seen, you must have all eight colons even if you leave some (or most) of the fields empty.

9.2.3 OpenView Data Collection and Thresholds

Once you close the OpenView graphs, the data in them is lost forever. OpenView provides a way to fix this problem with data collection. Data collection allows the user to poll and record data continuously. It can also look at these results and trigger events. One benefit of data collection is that it can watch the network for you while you're not there; you can start collecting data on Friday then leave for the weekend knowing that any important events will be recorded in your absence.

You can start OpenView's Data Collection and Thresholds function from the command line, using the command `$OV_BIN/xnmcollect`, or from NNM under the Options menu. This brings you to the "Data Collection and Thresholds" window, shown in [Figure 9-7](#), which displays a list of all the collections you have configured and a summary of the collection parameters.

Figure 9-7. OpenView's Data Collection and Thresholds window

File Edit Actions			Help
MIB Objects Configured for Collection			
Status	Label	MIB Object ID	
Suspended	ifInOctets	.1.3.6.1.2.1.2.2.1.10	
Suspended	ifOutOctets	.1.3.6.1.2.1.2.2.1.16	
Suspended	ifInErrors	.1.3.6.1.2.1.2.2.1.14	
Suspended	ifOutErrors	.1.3.6.1.2.1.2.2.1.20	
Suspended	15MinLoadAvg	.1.3.6.1.4.1.11.2.3.1.1.5	
Suspended	snmpInPkts	.1.3.6.1.2.1.11.1	
Suspended	IFZutil	IFZutil	
Suspended	DiskZutil	DiskZutil	

MIB Object Collection Summary				
Interval	Store	Threshold	Source	Instances

Configured collections that are in "Suspended" mode appear in a dark or bold font. This indicates that OpenView is not collecting any data for these objects. A "Collecting" status indicates that OpenView is polling the selected nodes for the given object and saving the data. To change the status of a collection, select the object, click on "Actions," and then click on either "Suspend Collection" or "Resume Collection." (Note that you must save your changes before they will take effect.)

9.2.3.1 Designing collections

To design a new collection, click on "Edit → Add MIB Object." This takes you to a new screen. At the top, click on "MIB Object"^[6] and click down through the tree until you find the object you would like to poll. To look at the status of our printer's paper tray, for example, we need to navigate down to `.iso.org.dod.internet.private.enterprises.hp.nm.system.net-peripheral.net-printer.generalDeviceStatus.gdStatusEntry.gdStatusPaperOut (.1.3.6.1.4.1.11.2.3.9.1.1.2.8)`.^[7] The object's description suggests that this is the item we want: it reads "This indicates that the peripheral is out of paper." (If you already know what you're looking for, you can enter the name or OID directly.) Once there, you can change the name of the collection to something that is easier to read. Click "OK" to move forward. This brings you to the menu shown in [Figure 9-8](#).

^[6] You can collect the value of an expression instead of a single MIB object. The topic of expressions is out of the scope of this book but is explained in the *mibExpr.conf* (4) manpage.

^[7] This object is in HP's private MIB, so it won't be available unless you have HP printers and have installed the appropriate MIBs. Note that there is a standard printer MIB, RFC 1759, but HP's MIB has more useful information.

Figure 9-8. OpenView poll configuration menu

The "Source" field is where you specify the nodes from which you would like to collect data. Enter the hostnames or IP addresses you want to poll. You can use wildcards like 198.27.6.* in your IP addresses; you can also click "Add Map" to add any nodes currently selected. We suggest that you start with one node for testing purposes. Adding more nodes to a collection is easy once you have everything set up correctly; you just return to the window in [Figure 9-8](#) and add the nodes to the Source list.

"Collection Mode" lets you specify what to do with the data NNM collects. There are four collection modes: "Exclude Collection," "Store, Check Thresholds," "Store, No Thresholds," and "Don't Store, Check Thresholds." Except for "Exclude Collection," which allows us to turn off individual collections for each device, the collection modes are fairly self-explanatory. ("Exclude Collection" may sound odd, but it is very useful if you want to exclude some devices from collection without stopping the entire process; for example, you may have a router with a hardware problem that is bombarding you with meaningless data.) Data collection without a threshold is easier than collection with a threshold, so we'll start there. Set the Collection Mode to "Store, No Thresholds." This disables (grays out) the bottom part of the menu, which is used for threshold parameters. (Select "Store, Check Thresholds" if you want both data collection and threshold monitoring.) Then click "OK" and save the new collection. You can now watch your collection grow in the `$OV_DB/snmpCollect` directory. Each collection consists of a binary datafile, plus a file with the same name preceded by an exclamation mark (!); this file stores the collection information. The data-collection files will grow

without bounds. To trim these files without disturbing the collector, delete all files that do not contain an "!" mark.

Clicking on "Only Collect on Nodes with sysObjectID:" allows you to enter a value for sysObjectID. sysObjectID (iso.org.dod.internet.mgmt.mib-2.system.sysObjectID) lets you limit polling to devices made by a specific manufacturer. Its value is the enterprise number the device's manufacturer has registered with IANA. For example, Cisco's enterprise number is 9, and HP's is 11 (the complete list is available at <http://www.isi.edu/in-notes/iana/assignments/enterprise-numbers>); therefore, to restrict polling to devices manufactured by HP, set the sysObjectID to 11. RFC 1213 formally defines sysObjectID (1.3.6.1.2.1.1.2) as follows:

sysObjectID OBJECT-TYPE

SYNTAX OBJECT IDENTIFIER

ACCESS read-only

STATUS mandatory

DESCRIPTION

"The vendor's authoritative identification of the network management subsystem contained in the entity. This value is allocated within the SMI enterprises subtree (1.3.6.1.4.1) and provides an easy and unambiguous means for determining what kind of box' is being managed. For example, if vendor 'Flintstones, Inc.' was assigned the subtree 1.3.6.1.4.1.4242, it could assign the identifier 1.3.6.1.4.1.4242.1.1 to its 'Fred Router'."

::= { system 2 }

The polling interval is the period at which polling occurs. You can use one-letter abbreviations to specify units: "s" for seconds, "m" for minutes, "h" for hours, "d" for days. For example, 32s indicates 32 seconds; 1.5d indicates one and a half days. When I'm designing a data collection, I usually start with a very short polling interval -- typically 7s (7 seconds between each poll). You probably wouldn't want to use a polling interval this short in practice (all the data you collect is going to have to be stored somewhere), but when you're setting up a collection, it's often convenient to use a short polling interval. You don't want to wait a long time to find out whether you're collecting the right data.

The next option is a drop-down menu that specifies what instances should be polled. The options are "All," "From List," and "From Regular Expression." In this case we're polling a scalar item, so we don't have to worry about instances; we can leave the setting to "All" or select "From List" and specify instance "0" (the instance number for all scalar objects). If you're polling a tabular object, you can either specify a comma-separated list of instances or choose the "From Regular Expression" option and write a regular expression that selects the instances you want. Save your changes ("File → Save"), and you're done.

9.2.3.2 Creating a threshold

Once you've set all this up, you've configured NNM to periodically collect the status of your printer's paper tray. Now for something more interesting: let's use thresholds to generate some sort of notification when the traffic coming in through one of our network interfaces exceeds a certain level. To do this, we'll look at a Cisco-specific object, `locIfInBitsSec` (more formally `iso.org.dod.internet.private.enterprises.cisco.local.linterface.s.lifTable.lifEntry.locIfInBitsSec`), whose value is the five-minute average of the rate at which data arrives at the interface, in bits per second. (There's a corresponding object called `locIfOutBitsSec`, which measures the data leaving the interface.) The first part of the process should be familiar: start Data Collection and Thresholds by going to the Options menu of NNM; then click on "Edit → Add MIB Object." Navigate through the object tree until you get to `locIfInBitsSec`; click "OK" to get back to the screen shown in [Figure 9-8](#). Specify the IP addresses of the interfaces you want to monitor and set the collection mode to "Store, Check Thresholds"; this allows you to retrieve and view the data at a later time. (I typically turn on the "Store" function so I can verify that the collector is actually working and view any data that has accumulated.) Pick a reasonable polling interval -- again, when you're testing it's reasonable to use a short interval -- then choose which instances you'd like to poll, and you're ready to set thresholds.

The "Threshold" field lets you specify the point at which the value you're monitoring becomes interesting. What "interesting" means is up to you. In this case, let's assume that we're monitoring a T1 connection, with a capacity of 1.544 Mbits/second. Let's say somewhat arbitrarily that we'll start worrying when the incoming traffic exceeds 75% of our capacity. So, after multiplying, we set the threshold to "> 1158000". Of course, network traffic is fundamentally bursty, so we won't worry about a single peak -- but if we have two or three consecutive readings that exceed the threshold, we want to be notified. So let's set "consecutive samples" to 3: that shields us from getting unwanted notifications, while providing ample notification if something goes wrong.

Setting an appropriate consecutive samples value will make your life much more pleasant, though picking the right value is something of an art. Another example is monitoring the `/tmp` partition of a Unix system. In this case, you may want to set the threshold to " ≥ 85 ", the number of consecutive samples to 2, and the poll interval to 5m. This will generate an event when the usage on `/tmp` exceeds 85% for two consecutive polls. This choice of settings means that you won't get a false alarm if a user copies a large file to `/tmp` and then deletes the file a few minutes later. If you set consecutive samples to 1, NNM

will generate a Threshold event as soon as it notices that /tmp is filling up, even if the condition is only temporary and nothing to be concerned about. It will then generate a Rearm event after the user deletes the file. Since we are really only worried about /tmp filling up and staying full, setting the consecutive threshold to 2 can help reduce the number of false alarms. This is generally a good starting value for consecutive samples, unless your polling interval is very high.

The rearm parameters let us specify when everything is back to normal or is, at the very least, starting to return to normal. This state must occur before another threshold is met. You can specify either an absolute value or a percentage. When monitoring the packets arriving at an interface, you might want to set the rearm threshold to something like 926,400 bits per second (an absolute value that happens to be 60% of the total capacity) or 80% of the threshold (also 60% of capacity). Likewise, if you're generating an alarm when /tmp exceeds 85% of capacity, you might want to rearm when the free space returns to 80% of your 85% threshold (68% of capacity). You can also specify the number of consecutive samples that need to fall below the rearm point before NNM will consider the rearm condition met.

The final option, "Configure Threshold Event," asks what OpenView events you would like to execute for each state. You can leave the default event, or you can refer to [Chapter 10](#) for more on how to configure events. The "Threshold" state needs a specific event number that must reside in the HP enterprise. The default Threshold event is OV_DataCollectThresh - 58720263. Note that the Threshold event is always an odd number. The Rearm event is the next number after the Threshold event: in this case, 58720264. To configure events other than the default, click on "Configure Threshold Event" and, when the new menu comes up, add one event (with an odd number) to the HP section and a second event for the corresponding Rearm. After making the additions, save and return to the Collection windows to enter the new number.

When you finish configuring the data collection, click "OK." This brings you back to the Data Collection and Thresholds menu. Click "File → Save" to make your current additions active. On the bottom half of the "MIB Object Collection Summary" window, click on your new object and then on "Actions → Test SNMP." This brings up a window showing the results of an SNMP test on that collection. After the test, wait long enough for your polling interval to have expired once or twice. Then click on the object collection again, but this time click on "Actions → Show Data." This window shows the data that has been gathered so far. Try blasting data through the interface to see if you can trigger a Threshold event. If the Threshold events are not occurring, verify that your threshold and polling intervals are set correctly. After you've seen a Threshold

event occur, watch how the Rearm event gets executed. When you're finished testing, go back and set up realistic polling periods, add any additional nodes you would like to poll, and turn off storing if you don't want to collect data for trend analysis. Refer to the `$OV_LOG/snmpCol.trace` file if you are having any problems getting your data collection rolling. Your HP OpenView manual should describe how to use this trace file to troubleshoot most problems.

Once you have collected some data, you can use `xnmgraph` to display it. The `xnmgraph` command to use is similar to the ones we saw earlier; it's an awkward command that you'll want to save in a script. In the following script, the `-browse` option points the grapher at the stored data:

```
#!/bin/sh
# filename: /opt/OV/local/scripts/graphSavedData
# syntax: graphSavedData <hostname>
/opt/OV/bin/xnmgraph -c public -title Bits_In_n_Out_For_All_Up_Interfaces \
-browse -mib \
".1.3.6.1.4.1.9.2.2.1.1.6:::1.3.6.1.2.1.2.2.1.8:1:1.3.6.1.2.1.2.2.1.2:::\
.1.3.6.1.4.1.9.2.2.1.1.8:::1.3.6.1.2.1.2.2.1.8:1:1.3.6.1.2.1.2.2.1.2::" \
$1
```

Once the graph has started, no real (live) data will be graphed; the display is limited to the data that has been collected. You can click on "File → Update Data" to check for and insert any data that has been gathered since the start of the graph. Another option is to leave off `-browse`, which allows the graph to continue collecting and displaying the live data along with the collected data.

Finally, to graph all the data that has been collected for a specific node, go to NNM and select the node you would like to investigate. Then select "Performance → Graph SNMP Data → Select Nodes" from the menus. You will get a graph of all the data that has been collected for the node you selected. Alternately, select the "All" option in "Performance → Graph SNMP Data." With the number of colors limited to 25, you will usually find that you can't fit everything into one graph.

9.2.4 Castle Rock's SNMPc

The workgroup edition of Castle Rock's SNMPc program has similar capabilities to the OpenView package. It uses the term "trend reporting" for its data collection and threshold facilities. The enterprise edition of SNMPc even allows you to export data to a web page. In all our examples we use the workgroup edition of SNMPc.

To see how SNMPc works, let's graph the `snmpOutPkts` object. This object's OID is 1.3.6.1.2.1.11.2 (iso.org.dod.internet.mgmt.mib-2.snmp.snmpOutPkts). It is defined in RFC 1213 as follows:

snmpOutPkts OBJECT-TYPE

SYNTAX Counter

ACCESS read-only

STATUS mandatory

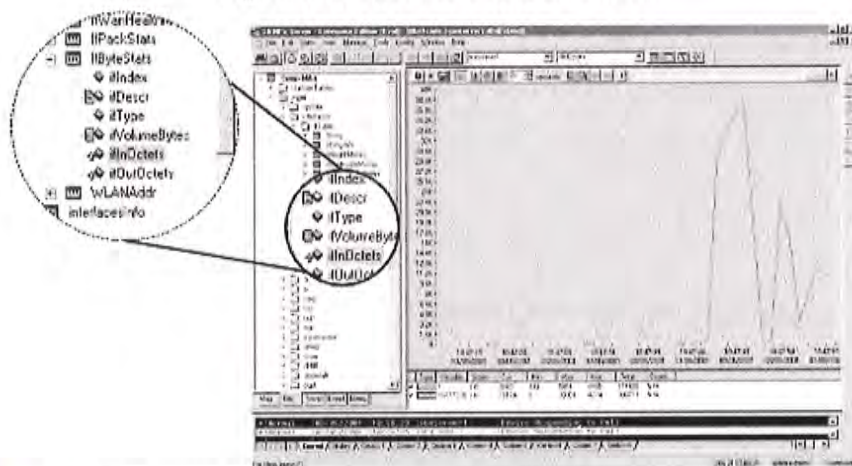
DESCRIPTION

"The total number of SNMP messages which were passed from the SNMP protocol entity to the transport service."

::= { snmp 2 }

We'll use the orahub device for this example. Start by clicking on the MIB Database selection tab shown in [Figure 9-9](#); this is the tab at the bottom of the screen that looks something like a spreadsheet -- it's the second from the left. Click down the tree until you come to iso.org.dod.internet.mgmt.mib-2.snmp. Click on the object you would like to graph (for this example, snmpOutPkts). You can select multiple objects with the Ctrl key.

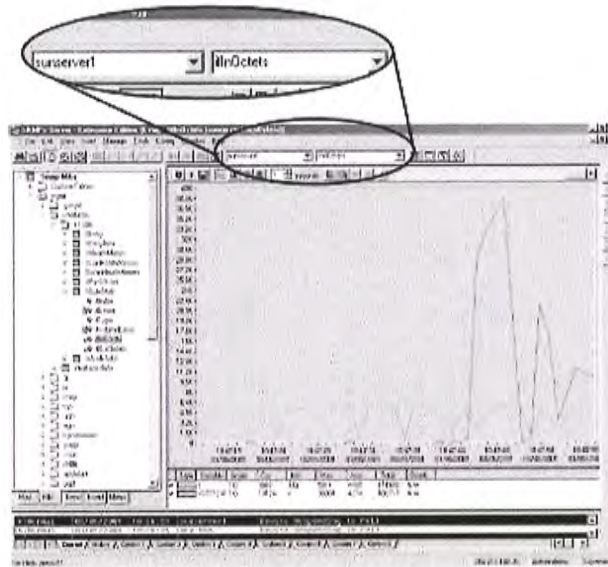
Figure 9-9. SNMPc MIB Database view



SNMPc has a nonstandard way of organizing MIB information. To get to the snmpOutPkts object, you need to click down through the following: "Snmp MIBs →mgmt →snmp →snmpInfo." Though this is quicker than the RFC-based organization used by most products, it does get a little confusing, particularly if you work with several products.

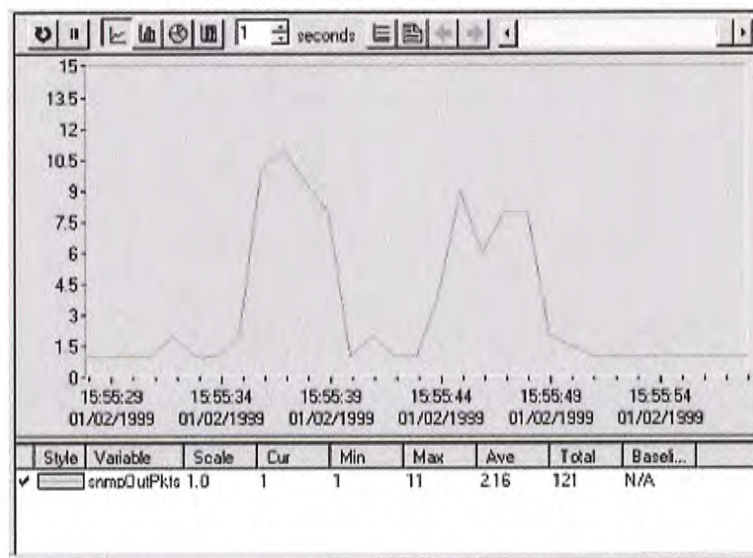
Once you have selected the appropriate MIB object, return to the top level of your map by either selecting the house icon or clicking on the Root Subnet tab (at the far left) to select the device you would like to poll. Instead of finding and clicking on the device, you can enter in the device's name by hand. If you have previously polled the device, you can select it from the drop-down box. [Figure 9-10](#) shows what a completed menu bar should look like.

Figure 9-10. SNMPc menu bar graph section



To begin graphing, click the button with the small jagged graph (the third from the right). Another window will appear displaying the graph (Figure 9-11). The controls at the top change the type of graph (line, bar, pie, distribution, etc.) and the polling interval and allow you to view historical data (the horizontal slider bar). Review the documentation on how each of these work or, better yet, play around to learn these menus even faster.

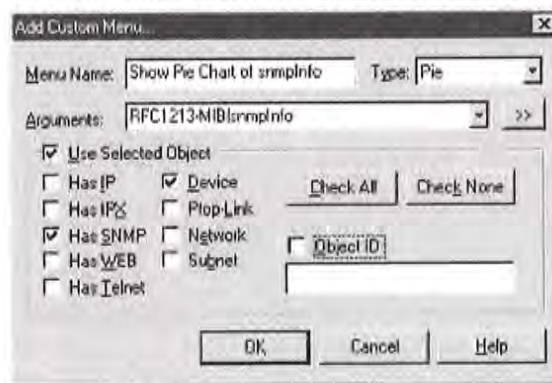
Figure 9-11. SNMPc snmpOutPkts graph section



Once you have a collection of frequently used graphs, you can insert them into the custom menus. Let's insert a menu item in the Tools menu that displays all the information in the snmpInfo table as a pie chart. Click on the Custom Menus tab (the last one), right-click on the Tools folder, and then left-click on "Insert Menu". This gets you to the "Add Custom Menu" window (Figure 9-12). Enter a menu name and select "Pie" for

the display type. Use the browse button (>>) to click down the tree of MIB objects until you reach the snmpInfo table; then click "OK." Back at "Add Custom Menu," use the checkboxes in the "Use Selected Object" section to specify the types of nodes that will be able to respond to this custom menu item. For example, to chart snmpInfo a device obviously needs to support SNMP, so we've checked the "Has SNMP" box. This information is used when you (or some other user) try to generate this chart for a given device. If the device doesn't support the necessary protocols, the menu entry for the pie chart will be disabled.

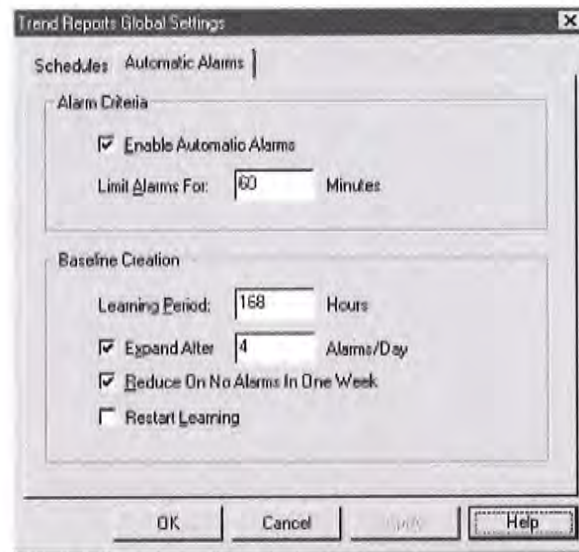
Figure 9-12. SNMPc Add Custom Menu window



Click "OK" and proceed to your map to find a device to test. Any SNMP-compatible device should suffice. Once you have selected a device, click on "Tools" and then "Show Pie Chart of snmpInfo." You should see a pie chart displaying the data collected from the MIB objects you have configured. (If the device doesn't support SNMP, this option will be disabled.) Alternately, you could have double-clicked your new menu item in the Custom Menu tab.

SNMPc has a threshold system called Automatic Alarms that can track the value of an object over time to determine its highs and lows (peaks and troughs) and get a baseline. After it obtains the baseline, it alerts you if something strays out of bounds. In the main menu, clicking on "Config → Trend Reports" brings up the menu shown in [Figure 9-13](#).

Figure 9-13. SNMPc Trend Reports Global Settings menu



Check the "Enable Automatic Alarms" box to enable this feature. The "Limit Alarms For" box lets you specify how much time must pass before you can receive another alarm of the same nature. This prevents you from being flooded by the same message over and over again. The next section, "Baseline Creation," lets you configure how the baseline will be learned. The learning period is how long SNMPc should take to figure out what the baseline really is. The "Expand After" option, if checked, states how many alarms you can get in one day before SNMPc increases the baseline parameters. In [Figure 9-13](#), if we were to get four alarms in one day, SNMPc would increase the threshold to prevent these messages from being generated so frequently. Checking the "Reduce On No Alarms In One Week" box tells SNMPc to reduce the baseline if we don't receive any alarms in one week. This option prevents the baseline from being set so high that we never receive any alarms. If you check the last option and click "OK," SNMPc will restart the learning process. This gives you a way to wipe the slate clean and start over.

9.2.5 Open Source Tools for Data Collection and Graphing

One of the most powerful tools for data collection and graphing is MRTG, familiar to many in the open source community. It collects statistics and generates graphical reports in the form of web pages. In many respects, it's a different kind of animal than the tools discussed in this chapter. We cover MRTG in [Chapter 13](#).

Chapter 10. Traps

Traps provide a way for an agent to send a monitoring station asynchronous notification about conditions that the monitor should know about. The traps that an agent can generate are defined by the MIBs it supports; the number of traps can range from zero to hundreds. To see what traps are defined in any MIB

file, search for the term "TRAP-TYPE" (SMIv1) or "NOTIFICATION-TYPE" (SMIv2) in the MIB file. This search will quickly get you a list of possible traps.

Of course, just having asynchronous traps arrive at your NMS isn't terribly useful. You can configure the NMS's response to different traps; the response can be anything from discarding the trap to running a script that sends a message to your pager (or even takes some drastic action, such as shutting down your power supplies). In this chapter, we'll show you how to handle incoming traps using OpenView and other tools such as Perl. Then we'll discuss how to read and configure different aspects of trap events. Finally, we'll show you how to define your own traps that report special conditions of particular interest to your network.

10.1 Understanding Traps

Before discussing the tools for receiving and generating traps, it's worth reviewing what a trap is. Traps were introduced in [Chapter 2](#). A trap is basically an asynchronous notification sent from an SNMP agent to a network-management station. Like everything else in SNMP, traps are sent using UDP (port 162) and are therefore unreliable. This means that the sender cannot assume that the trap actually arrives, nor can the destination assume that it's getting all the traps being sent its way. Of course, on a healthy network most traps should reach their destinations. But if networks were always healthy, we wouldn't need SNMP.

In somewhat more detail, a trap is a bundle of data that's defined by a MIB. Traps fall into two categories, generic and enterprise-specific. There are seven generic trap numbers (0-6), defined in [Table 2-8](#), for conditions ranging from system reboots (coldStart) and interface state changes (linkUp and linkDown) to generic trap 6 (enterpriseSpecific). Enterprise-specific traps are the loophole that makes the trap mechanism so powerful. Anyone with an enterprise number can define enterprise-specific traps for whatever conditions they consider worth monitoring. An enterprise-specific trap is identified by two pieces of information: the enterprise ID of the organization that defined the trap and a specific trap number assigned by that organization. The notion of an enterprise-specific trap is extremely flexible, because organizations are allowed to subdivide their enterprises as much as they like. For example, if your enterprise number is 2789, your enterprise ID is .1.3.6.1.4.1.2789. But you can further subdivide this, defining traps with enterprise IDs such as .1.3.6.1.4.1.2789.5000, .1.3.6.1.4.1.2789.5001, and so on.

The fact that you've received a trap and therefore know its generic trap number, enterprise ID, and specific trap number is

often all you need to diagnose a problem. But traps also carry additional information. In the case of generic traps 0-5, the specific information is predefined and hardwired into the NMS. When you receive a generic trap, the NMS knows how to interpret the information it contains and will be able to display it appropriately, whether it's the time of the reboot or the identity of the interface that just changed state. In contrast, the information carried by an enterprise-specific trap is entirely up to the person who defined the trap. An enterprise-specific trap can contain any number of variable bindings, or MIB object-value pairs. When you define your own traps, you can decide what information is appropriate for them to carry. The objects contained in a trap can be standard MIB objects, vendor-specific objects, or objects of your own devising. It's common to define objects purely for the purpose of including them within a trap.

10.1.1 SNMPv2 Traps

SNMPv2 defines traps in a slightly different way. In a MIB, Version 1 traps are defined as `TRAP-TYPE`, while Version 2 traps are defined as `NOTIFICATION-TYPE`. SNMPv2 also does away with the notion of generic traps -- instead, it defines many specific traps (properly speaking, notifications) in public MIBs. SNMPv3 traps, which are discussed briefly in [Appendix F](#), are simply SNMPv2 traps with added authentication and privacy capabilities. Most SNMP implementations support only Version 1.

10.2 Receiving Traps

Let's start by discussing how to deal with incoming traps. Handling incoming traps is the responsibility of the NMS. Some NMSs do as little as display the incoming traps to standard output (stdout). However, an NMS server typically has the ability to react to SNMP traps it receives. For example, when an NMS receives a linkDown trap from a router, it might respond to the event by paging the contact person, displaying a pop-up message on a management console, or forwarding the event to another NMS. This procedure is streamlined in commercial packages but still can be achieved with freely available open source programs.

10.2.1 HP OpenView

OpenView uses three pieces of software to receive and interpret traps:

- `ovtrapd (1M)`
- `xnmtrap`
- `xnmevents`

OpenView's main trap-handling daemon is called `ovtrapd`. This program listens for traps generated by devices on the network and hands them off to the Postmaster daemon (`pmd`). In turn, `pmd` triggers what OpenView calls an event. Events can be configured to perform actions ranging from sending a pop-up window to NNM users, forwarding the event to other NMSs, or doing nothing at all. The configuration process uses `xnmtrap`, the Event Configurations GUI. The `xnmevents` program displays the events that have arrived, sorting them into user-configurable categories.

OpenView keeps a history of all the traps it has received; to retrieve that history, use the command `$OV_BIN/ovdumpevents`. Older versions of OpenView kept an event logging file in `$OV_LOG/trapd.log`. By default, this file rolls over after it grows to 4 MB. It is then renamed `trapd.log.old` and a new `trapd.log` file is started. If you are having problems with traps, either because you don't know whether they are reaching the NMS or because your NMS is being bombarded by too many events, you can use `tail -f` to watch `trapd.log` so you can see the traps as they arrive. (You can also use `ovdumpevents` to create a new file.) To learn more about the format of this file, refer to OpenView's manual pages for `trapd.conf` (4) and `ovdumpevents` (1M).

It might be helpful to define what exactly an OpenView event is. Think of it as a small record, similar to a database record. This record defines which trap OpenView should watch out for. It further defines what sort of action (send an email, page someone, etc.), if any, should be performed.

10.2.2 Using NNM's Event Configurations

OpenView uses an internal definition file to determine how to react to particular situations. This definition file is maintained by the `xnmtrap` program. We can start `xnmtrap` by using the menu item "Options → Event Configurations" (on the NNM GUI) or by giving the command `$OV_BIN/xnmtrap`. In the Enterprise Identification window, scroll down and click on the enterprise name "OpenView .1.3.6.1.4.1.11.2.17.1." This displays a list in the Event Identification window. Scroll down in this list until you reach "OV_Node_Down." Double-click on this event to bring up the Event Configurator ([Figure 10-1](#)).

Figure 10-1. OpenView Event Configurator -- OV_Node_Down

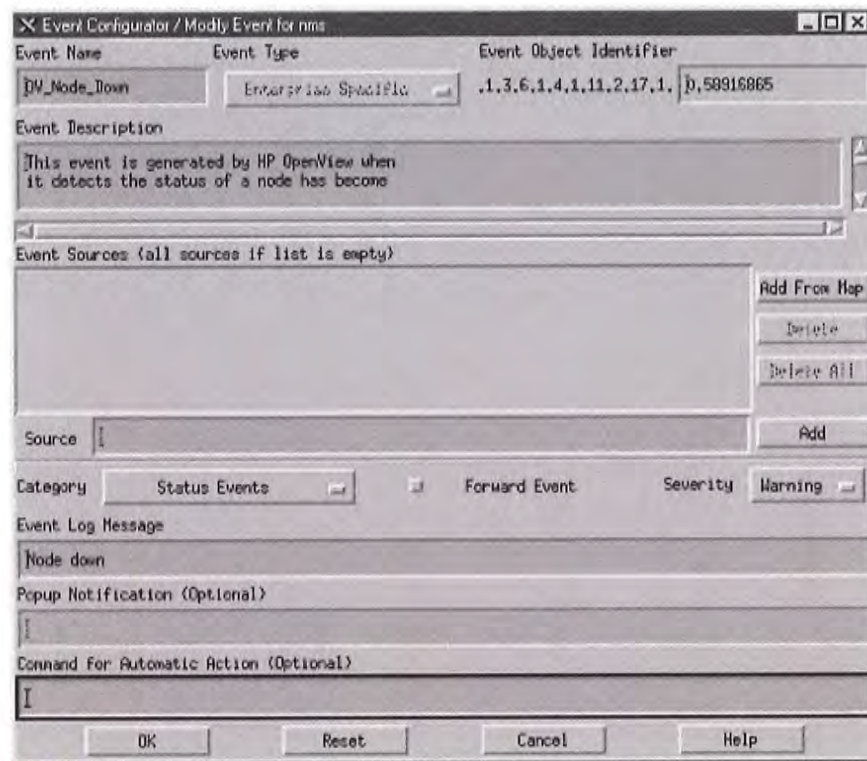


Figure 10-1 shows the OV_Node_Down event in the Event Configurator. When this event gets triggered, it inserts an entry containing the message "Node down," with a severity level of "Warning," into the Status Events category. OpenView likes to have a leading 0 (zero) in the Event Object Identifier, which indicates that this is an event or trap -- there is no way to change this value yourself. The number before the 0 is the enterprise OID; the number after the 0 is the specific trap number, in this case 58916865.^[1] Later we will use these numbers as parameters when generating our own traps.

^[1] This is the default number that OpenView uses for this OV_Node_Down trap.

10.2.2.1 Selecting event sources

The Source option is useful when you want to receive traps from certain nodes and ignore traps from other nodes. For example, if you have a development router that people are taking up and down all day, you probably would rather not receive all the events generated by the router's activity. In this case, you could use the Source field to list all the nodes from which you would like to receive traps, and leave out the development router. To do this, you can either type each of the hostnames by hand and click "Add" after each one, or select each node (using the Ctrl and mouse-click sequence) on your OpenView Network Node Map and click "Add From Map." Unfortunately, the resulting list isn't easy to manage. Even if you take the time to add all the current routers to the Event Sources, you'll

eventually add a new router (or some other hardware you want to manage). You then have to go back to all your events and add your new devices as sources. Newer versions of OpenView allow you to use pattern matching and source files, making it easier to tailor and maintain the source list.

10.2.2.2 Setting event categories

When NNM receives an event, it sorts the event into an event category. The Categories drop-down box lets you assign the event you're configuring to a category. The list of available categories will probably include the following predefined categories (you can customize this list by adding categories specific to your network and deleting categories, as we'll see later in this section):

- Error events
- Threshold events
- Status events
- Configuration events
- Application alert events
- Don't log or display
- Log only

The last two categories really aren't event categories in the true sense of the word. If you select "Don't log or display," OpenView will not save the event in its database and will not display the Event Log Message in any Event Categories. OpenView will display the Popup Notification in a pop-up window and run the Command for Automatic Action. The "Log only" option tells OpenView not to display the event but to keep a log of the event in its database.^[2]

^[2] Again, in earlier releases of OpenView this log was located in *\$OV_LOG/trapd.log*. New versions use the OpenView Event Database. This is backward-compatible using the *ovdumpevents* command to produce a *trapd.log* file.



"Log only" is useful if you have some events that are primarily informational; you don't want to see them when they arrive, but you would like to record them for future reference. The Cisco event `frDLCIStatusChange - .1.3.6.1.2.1.10.32.0.1` is a good example of such an event. It tells us when a Virtual Circuit has changed its operational state. If displayed, we will see notifications whenever a node goes down and whenever a circuit changes its operational

state to down. This information is redundant because we have already gotten a status event of "node down" and a DLCI change.^[3] With this event set to "Log only" we can go to the log file only when we think things are fishy.

^[3] Newer versions of OpenView have a feature called Event Correlation that groups certain events together to avoid flooding the user with redundant information. You can customize these settings with a developer's kit.

10.2.2.3 Forwarding events and event severities

The "Forward Event" radio button, once checked, allows you to forward an event to other NMSs. This feature is useful if you have multiple NMSs or a distributed network-management architecture. Say that you are based in Atlanta, but your network has a management station in New York in addition to the one on your desk. You don't want to receive all of New York's events, but you would like the node_down information forwarded to you. On New York's NMS, you could click "Forward Event" and insert the IP address of your NMS in Atlanta. When New York receives a node_down event, it will forward the event to Atlanta.

The Severity drop-down list assigns a severity level to the event. OpenView supports six severity levels: Unknown, Normal, Warning, Minor, Major, and Critical. The severity levels are color-coded to make identification easier; [Table 10-1](#) shows the color associated with each severity level. The levels are listed in order of increasing severity. For example, an event with a severity level of Minor has a higher precedence than an event with a severity of Warning.

Table 10-1. OpenView Severity Levels	
Severity	Color
Unknown	Blue
Normal	Green
Warning	Cyan
Minor	Yellow
Major	Orange
Critical	Red