Table of Contents

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

Chapter 2. SNMP

2.1. Overview of SNMP

SNMP stands for Simple Network Management Protocol, and it is exactly what it sounds like: a relatively simple protocol used to manage devices on a network. The managed device is commonly a workstation, a switch, or a router but can be anything capable of speaking IP. For example, many printers and uninterruptible power supplies (UPSs) can be managed with SNMP. What exactly does it mean to be able to "manage" a device? It means an operator, or an SNMP **management agent** acting on behalf of an operator, can query the device for information and may also be able to instruct the device to make a change in configuration. In managing a router, an operator might use SNMP to query the router for the number of packets transmitted on an interface or instruct the router to make a change to its routing table.

The strength of SNMP lies in its simplicity and its standardization. Because the protocol is simple, it can be implemented in even the most basic devices. The fact that it is an Internet standard makes it an attractive tool for a site that has many devices from many different vendors. Each device may operate differently under the hood, but all devices must present the same management interface via SNMP. Then an operator can manage every device on the network in nearly the same way. The result of SNMP's ubiquity is that it is now the underlying technology for other management tools, including Neo and MRTG, which are discussed in later chapters.

There are currently three versions of SNMP: SNMPv1, SNMPv2, and SNMPv3. SNMPv2 is not drastically different from SNMPv1, and SNMPv3, which adds security to the protocol, is not as widely deployed as the first two. This chapter focuses primarily on SNMPv1.

2.1.1. SNMP

When we talk about using SNMP for network management, we're really talking about a small collection of standards. SNMP itself is the protocol used to transmit management packets over the network. Other standards define how pieces of management information are named and organized. SNMP is built on top of the User Datagram Protocol (UDP). Because UDP is an unreliable protocol by design, there is nothing to ensure that SNMP packets will not be lost in transmission. Accordingly, most management agent software implements a timeout and retransmission to ensure that SNMP messages are received.

Three pieces of information are included in each SNMP packet before the actual management information. First is the version number, which for SNMPv1 is the value zero. Next is the SNMP **community name**. The community is a text string that serves as a rudimentary kind of authentication. The device being managed is configured ahead of time to allow certain communities access to certain management capabilities. For example, you may give a community called `my-secret` access only to retrieve management information from a device while you may give the community `really-secret` access to both change and retrieve it. The operator or management agent supplies the appropriate community name in the SNMP request, and upon receipt of the request, the device decides whether the given community has the necessary privileges.

It is important to note that in SNMPv1 and SNMPv2, the community name is not encrypted in the packet in any way. It is sent over the network as a cleartext password. This means that if these packets are in some way intercepted, an eavesdropper will easily be able to pick out the community name. If the managed device has no access restrictions in place other than the community name, the eavesdropper will now be able to use the community to perform management requests against the device. SNMPv3, in contrast, does have a real authentication mechanism so that an eavesdropper will not be able to gain privileged access. Unfortunately, SNMPv3 is not available on many manageable devices. This means the only options may be to:

- Disable some or all SNMP access

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

- Design your network in such a way that management traffic does not travel on networks where a malicious user may eavesdrop
- Use a firewall or other filter to block SNMP traffic from untrusted sources
- Enforce no special access restrictions and hope for the best

Network operators should consider what the security implications would be of unauthorized read- and write-level access to network devices and make management access decisions accordingly. Incidentally, should you wish to make your SNMP management open to everyone, the accepted default value for the community name is `public`.[1]

[1] At MIT we set the read-only community to public so that anyone may query devices for information. The read-write community is, of course, not open to the public.

The third piece of information sent in the SNMP header indicates the type of request being sent and is called the **Protocol Data Unit (PDU) type**. PDU is just a fancy word for a packet. There are five valid choices for the PDU type, listed in Figure 2.1. A `get-request` asks the device to return a piece of information while a `set-request` asks the device to change a setting. The `get-response` is the device's response to one of those requests. The other two PDU types, `get-next-request` and `trap`, are discussed later in this chapter.

Figure 2.1. PDU Types in SNMPv1.

PDU Type	Name	Function
0	`get-request`	Retrieve a variable's value
1	`get-next-request`	Retrieve the next variable and its value
2	`get-response`	Response from the managed device
3	`set-request`	Set a variable's value
4	`trap`	Spontaneous information from the device

2.1.2. Variables and the MIB

While the SNMP standard defines a framework for talking to a device about management, we need a common way to talk about the information we wish to retrieve or change. Each piece of information available via SNMP is called an SNMP **variable**. There is a variable that indicates the operational status of an interface, for example, called `ifOperStatus`. Another variable, called `sysUpTime`, represents the amount of time the given device has been up and running.

Every SNMP variable is specified in the Management Information Base, more commonly called the **MIB**. The MIB is a separate standard from SNMP. The job of the MIB is to:

- Assign every variable a textual name
- Define a data type for the variable
- Describe the function of the variable and how the data type is used to achieve that function
- Define the access possibilities for the variable (e.g., read-only versus read-write)

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

Practically speaking, the MIB is a text file that lists SNMP variables and all the necessary information associated with them. The file is written in Abstract Syntax Notation One (ASN.1), a standard format for describing data, so that it is readable both by humans and by programs that act as SNMP agents. The data types themselves are further standardized by the Structure of Management Information (SMI) standard, which defines data types such as "TimeTicks" or "IpAddress."

The first MIB, the Internet Standard MIB, was published by the Internet Engineering Task Force (IETF) in 1998 as RFC 1066.[2] RFC 1158 and, later, RFC 1213 describe a new Internet MIB called MIB-II, which effectively replaces the original MIB. Any device speaking SNMP today is required to implement MIB-II.

[2] This document was also revised very slightly and published in 1990 as RFC 1156, but these modifications did not reflect any technical changes in the system.

There are now so many SNMP variables in use that it is not feasible to have one file that describes them all. Additionally, it is desirable for individual enterprises, such as private companies, to have control over their own set of variables. These variables might control products they produce, for example. So instead of one large MIB file, there are many separate component files, some published by the IETF but also many published by vendors.

Note that the term MIB is sometimes used to refer to the Internet standard MIB, which now effectively means MIB-II. But other times, it refers to some particular portion of the MIB that is published separately. So while a purist might say there is only one MIB, it is common to hear someone talk about the "Bridge MIB" or an "Asante MIB." It is usually clear from context what is meant.

In the following sections we look at the MIB in closer detail, but still only as an introduction to the topic. For an in-depth discussion of MIBs, read *Understanding SNMP MIBS* (Prentice Hall PTR, 1996) by David Perkins and Evan McGinnis.

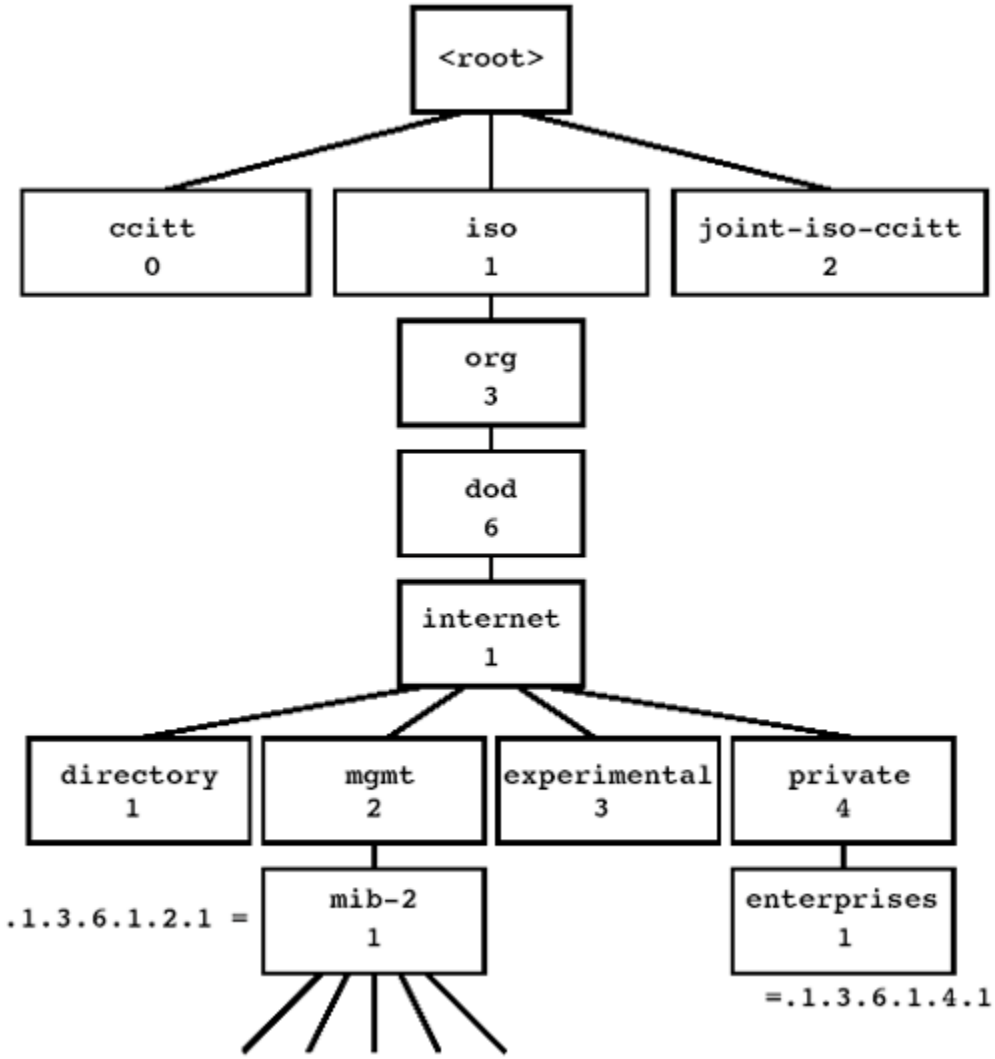
2.1.3. Object Identifiers and Variable Hierarchy

SNMP variable names must be globally unique. To help achieve this goal, they are arranged in a hierarchical fashion. The `sysUpTime` variable mentioned earlier is really named in full `.iso.org.dod.internet.mgmt.mib-2.system.sysUpTime`. The `sysUpTime` variable belongs to a set of variables under the `system` group that all relate to management information about the state of the device as a whole. The `ifOperStatus` variable, which indicates the operational status of an interface, is found under the `interfaces` tree, which contains variables relating to interface-specific information.

The root of the variable hierarchy has no name, but it has three children, depicted in Figure 2.2. One is for use by the International Organization for Standardization (ISO), one for the Comité Consultatif International Téléphonique et Télégraphique (CCITT) (now the International Telecommunications Union [ITU]), and the third is for joint ISO and CCITT use. The scope of this naming scheme is actually much larger than that used by SNMP. All SNMP variables in use on the Internet fall under the `.iso.org.dod.internet` branch of the tree.

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

Figure 2.2. Object Identifier Hierarchy.



In addition to having a textual name, each branch of the tree has an integer associated with it. This number is used in communicating variable names over the wire in SNMP. The textual names are for human readability only.
So `.iso.org.dod.internet.mgmt.mib-2.system.sysUpTime`, when encoded in an SNMP packet, is sent as `.1.3.6.1.2.1.1.3`. This string of integers is called the variable's **object identifier**, often abbreviated to **object ID** or **OID**.

The `mib-2` branch of the variable hierarchy is where the Internet standard MIB begins. The `private.enterprises` portion of the tree is where MIBs assigned to private organizations begins.

2.1.4. Variable Instances for Simple Variables

When making an SNMP request for a simple variable, such as `sysUpTime`, you must append a final `.0` to the variable name, for reasons that will become clear when SNMP tables are discussed in Section 2.1.5. For example, `sysUpTime` must actually be requested as

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

`sysUpTime.0` or at the protocol level, `.1.3.6.1.2.1.1.3.0`. The SNMP tools will require you to supply the final `.0` when you make requests.

2.1.5. Introduction to Tables

Some kinds of management information, such as the name of the system contact, can be easily stored in a single variable. Other kinds of information are better described in a table of values. For example, if a device has 24 ports, it would be convenient to represent traffic data as a table with 24 rows, one for each port.

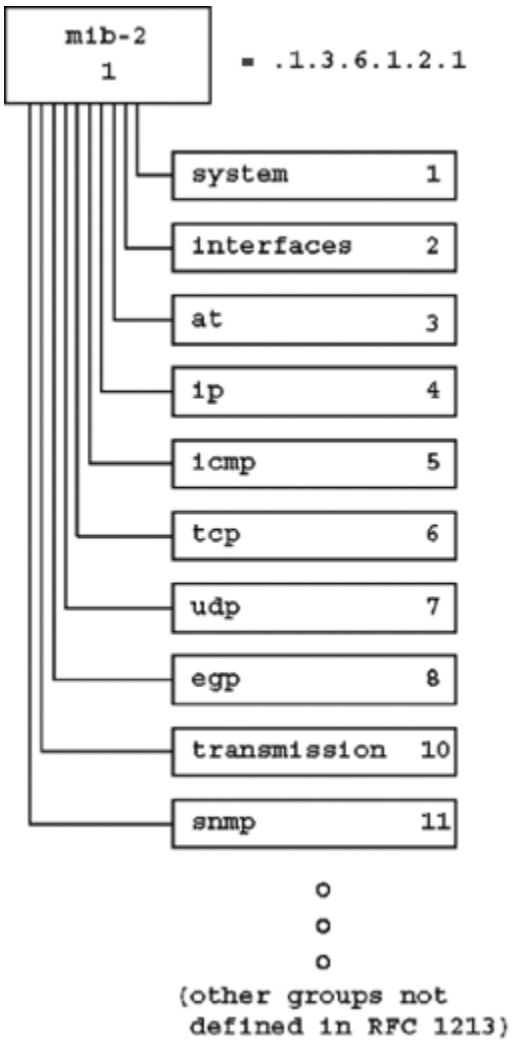
The MIB has a facility for defining a table. Instead of going into great detail on how tables are designed, it's easier to look at a real example:

```
interfaces.ifTable.ifEntry.ifOperStatus.1 = up(1)
interfaces.ifTable.ifEntry.ifOperStatus.2 = up(1)
interfaces.ifTable.ifEntry.ifOperStatus.3 = up(1)
interfaces.ifTable.ifEntry.ifOperStatus.4 = down(2)
interfaces.ifTable.ifEntry.ifOperStatus.5 = down(2)
interfaces.ifTable.ifEntry.ifOperStatus.6 = up(1)
interfaces.ifTable.ifEntry.ifOperStatus.7 = up(1)
interfaces.ifTable.ifEntry.ifOperStatus.8 = up(1)
interfaces.ifTable.ifEntry.ifOperStatus.9 = up(1)
interfaces.ifTable.ifEntry.ifOperStatus.10 = down(2)
interfaces.ifTable.ifEntry.ifOperStatus.11 = up(1)
interfaces.ifTable.ifEntry.ifOperStatus.12 = up(1)
interfaces.ifTable.ifEntry.ifOperStatus.13 = up(1)
interfaces.ifTable.ifEntry.ifOperStatus.14 = up(1)
interfaces.ifTable.ifEntry.ifOperStatus.15 = up(1)
interfaces.ifTable.ifEntry.ifOperStatus.16 = up(1)
interfaces.ifTable.ifEntry.ifOperStatus.17 = down(2)
interfaces.ifTable.ifEntry.ifOperStatus.18 = down(2)
```

This is a portion of the `interfaces` group, which is a direct child of `mib-2` (see Figure 2.3. The `interfaces` group is made up of one simple variable (`ifNumber`, the number of interfaces on the system) and one table called `ifTable`. The preceding output is only a small portion of the `ifTable`, in particular the portion showing the operational status of an interface. Instead of a `.0` on the end of each variable name, a number is appended that acts as an index into a row of the table. For the interface table, this index typically corresponds to the interface port number. In the example, we can determine that port one is operational while port ten is not. On Ethernet switches, this often means that port one has link while port ten does not.

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

Figure 2.3. Children of mib-2 Defined in RFC 1213.



The interface table has many columns other than the `ifOperStatus` column that contain additional data about the interface. The `ifSpeed` column, for example, indicates the bandwidth of the interface, and the `ifInOctets` column contains the number of bytes of data received by the interface.

2.1.6. Lexicographic Ordering and Get-Next-Request

Because each SNMP variable name is an ordered sequence of integers, it is possible to list variables in a lexicographic order. That is, every variable comes numerically before or after another variable, and so a complete ordered listing of all the variables a device uses can be made. This ordering gives purpose to the `get-next-request` PDU type. Using the `get-request` PDU type, we supply a variable name and the managed device returns the variable's value, along with the name of the variable we requested. When we make a request with the `get-next-request` PDU type, we supply a variable name and the managed device returns the value of the *next* SNMP variable it knows about, along with the name of that variable.

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

This is a convenient way to walk through a table. We continue asking for the next variable in the table until we either find a variable that is no longer of interest or run out of variables. If we start at the root, `get-next-request` will allow us to step through the entire tree of variables that a device supports. An example is given in Section 2.4.3, when the `snmpwalk` tool is discussed.

2.1.7. Traps

The `trap` PDU type is a special kind of SNMP message. Instead of generating an SNMP message in response to a query, a device can spontaneously send a trap to a listening SNMP agent. One useful example of a trap is the `linkDown` trap. When the link state on an interface changes from up to down, the managed device, if configured to use traps, sends an SNMP packet indicating the change. The listening SNMP agent can then use this information to inform an operator.

Because a trap is sent over UDP, just like any other SNMP packet, there is no guarantee that it will make it to its destination. But the problem is somewhat worse for traps than for other types of SNMP requests. SNMP agents that make a query with `get-request`, `get-next-request`, or `set-request` can wait for a response, and if one does not arrive, it can try the query again. A device sending a trap, however, has no mechanism to determine if the trap was received.

Also remember that under certain circumstances, a device may not be able to send a trap. If it unexpectedly loses power, for example, it will not have any opportunity to send a message.

2.2. What SNMP Can Help You Do

Protocol design aside, the practical power of SNMP lies in device information available to be retrieved and changed. Fortunately there is a great deal of device configuration available via SNMP. This section looks at some of the standard variables available on networked devices.

2.2.1. The System Group

The `system` group is a direct child of `mib-2` that contains variables related to general information about the device. RFC 1213 defines seven simple variables in this group, listed in Figure 2.4. The # column in this table designates the part of the object identifier appended to `mib-2` for the given variable. The RW column indicates whether the variable can be set with an SNMP `set-request`.

Figure 2.4. Variables in the `System` Group.

Variable	#	RW	Function
`sysDescr`	1		Text description of the device, including OS
`sysObjectID`	2		An SNMP object identifier for this device
`sysUpTime`	3		Device up time, in hundredths of a second

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

Variable	#	RW	Function
sysContact	4	Yes	Contact information for the device admin
sysName	5	Yes	Admin assigned name, typically the hostname
sysLocation	6	Yes	Physical location information for the device
sysServices	7		Integer value representing available services

The function of most variables is clear. The `sysObjectID` variable returns an SNMP object ID that is unique to that particular kind of device. This object ID is designated by the vendor and therefore always starts with `.1.3.6.1.4.1`, which is the prefix for private enterprises. The `sysServices` variable designates which layers of network service are provided by the device. Its use requires some explanation. From RFC 1213:[3]

[3] RFC 1213. Management Information Base for Network Management of TCP/IP-based internets: MIB-II. K. McCloghrie, M.T. Rose. Mar-01-1991.

```
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.
```

Example values of the system group from a real device look like this:

```
system.sysDescr.0 = "Asante IntraSwitch 5324MT"
system.sysObjectID.0 = OID: enterprises.298.2.2.15
system.sysUpTime.0 = Timeticks: (186442210) 21 days, 13:53:42.10
system.sysContact.0 = "admins@example.com"
system.sysName.0 = "switch.example.com"
system.sysLocation.0 = "5-125T"
system.sysServices.0 = 10
```

The variables `sysContact`, `sysName`, and `sysLocation` are set by an operator using either SNMP or a separate management interface to the device. Notice that the `sysObjectID` variable points to an object ID under `enterprises.298`, which is assigned to Asante. The `sysServices` value is 10, indicating that layer 2 and layer 4 services are available. On this device, an Ethernet switch, the layer 2 service is packet switching and the layer 4 service is the IP functionality the device uses for management.

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

2.2.2. The Interfaces Group

The `interfaces` group is another direct child of `mib-2`. It contains variables related to information about individual interfaces on a piece of network hardware. A workstation or other simple network device with one network connection may have only one or two interfaces. A switch or router may have many.

The `interfaces` group has only one simple variable, `ifNumber`, which is the number of interfaces available on the system. An example from a router:

```
interfaces.ifNumber.0 = 91
```

and from a workstation:

```
interfaces.ifNumber.0 = 2
```

In this case, the workstation has two interfaces, one physical network interface, and one loopback interface.

The only other variable defined for the `interfaces` group is the table `ifTable`. The columns of this table are listed in Figure 2.5. Most of these are self-explanatory. One thing worth noting is that it is not always easy to relate the interface index number to the port number on a device. Sometimes the index number will be the same as the port number, such that when `ifIndex` is three, it will refer to port three on the device. Sometimes this is not the case, particularly on devices that have multiple boards and number each port as a board/port pair. There is no standard mechanism for making the conversion. Cisco devices, for example, use a simple formula to convert the board/port to an `ifIndex` number and vice versa. Extreme products use a similar formula but with different values so that port one on board one is `ifIndex` 1001, port two on board one is 1002, and port one on board two is 2001. Some devices provide the ability to use a separate SNMP lookup in the private enterprises portion of the MIB to perform the conversion.

Figure 2.5. Variables in `IfTable`.

Variable	#	RW	Function
`ifIndex`	1		Interface index number
`ifDescr`	2		Text string describing the interface
`ifType`	3		Integer representing the interface protocole
`ifMtu`	4		The MTU of the interface
`ifSpeed`	5		Nominal bandwidth of the interface
`ifPhysAddress`	6		Layer 2 physical address of the interface
`ifAdminStatus`	7	Yes	Desired administrative state of the interface
`ifOperStatus`	8		Current operational state of the interface

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

Variable	#	RW	Function
ifLastChange	9		sysUpTime when OperStatus last changed
ifInOctets	10		Total number of octets received by interface
ifInUcastPkts	11		Number of unicast packets received
ifInNUcastPkts	12		Number of non-unicast packets received
ifInDiscards	13		Number of inbound packets discarded
ifInErrors	14		Number of inbound packets with errors
ifInUnknownProtos	15		Number of inbound packets of unknown protocol
ifOutOctets	16		Total number of octets send by the interface
ifOutUcastPkts	17		Number of unicast packets sent
ifOutNUcastPkts	18		Number of non-unicast packets sent
ifOutDiscards	19		Number of outbound packets discarded
ifOutErrors	20		Number of outbound packets with errors
ifOutQLen	21		Number of packets in the outgoing queue
ifSpecific	22		A media dependent object identifier

The `interfaces` group does not report current interface bandwidth usage. Originally the RFC called for `ifSpeed` to do just that, but in practice, it is always used to report the maximum speed possible on the link. On switches capable of supporting multiple link speeds, the value is usually set to the speed of the current link, in bits per second, or zero if no link is present.

In order to gather statistics on current interface bandwidth use, it is necessary to retrieve a byte count, such as `ifInOctets`, wait a designated period of time, retrieve a second byte count, and then subtract the difference and divide by the elapsed time. For example, we retrieve:

```
interfaces.ifTable.ifEntry.ifInOctets.6 = 4109645530
```

and then exactly 10 seconds later retrieve:

```
interfaces.ifTable.ifEntry.ifInOctets.6 = 4109669635
```


Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

The average bandwidth use at this time is $\frac{4109669635-4109645530}{10} = 2410$ Bytes/s.

The `ifType` variable uses an integer to represent the protocol spoken by the interface. The value for Ethernet is 6, while PPP's is 23. The full table of supported values is listed in RFC 1213.

2.2.3. ip.ipNetToMediaTable

The `ip.ipNetToMediaTable` is used to translate IP addresses to hardware addresses. In the context of Ethernet, this means looking up the Ethernet address associated with a particular IP address in the device's Address Resolution Protocol (ARP) cache. An older method of doing this involved using the `at` group, which while usually still supported, is not as generic as the newer system. The `ip.ipNetToMediaTable` is shown in Figure 2.6. To show how this table would be used, sample data from a router follows. In order to save space, the variable names have been shortened; each variable name would begin with the prefix `ip.ipNetToMediaTable.ipNetToMediaEntry`.

```
ipNetToMediaPhysAddress.4.10.152.0.44 = 0:0:1d:dc:85:2d
ipNetToMediaPhysAddress.4.10.152.0.45 = 0:e0:63:2b:c9:c0
ipNetToMediaPhysAddress.4.10.152.0.46 = 0:e0:63:2b:ca:f8
ipNetToMediaPhysAddress.4.10.152.0.53 = 8:0:20:9e:f1:80
ipNetToMediaPhysAddress.4.10.152.0.54 = 8:0:69:2:e6:ec
ipNetToMediaPhysAddress.4.10.152.0.60 = 8:0:20:93:da:3f
ipNetToMediaPhysAddress.4.10.152.0.61 = 0:5:2:f6:c7:b
ipNetToMediaPhysAddress.4.10.152.0.62 = 0:1:e6:4f:9a:6a
```

Figure 2.6. The `ip.ipNetToMediaTable`.

Variable	#	RW	Function
`ipNetToMediaIfIndex`	1	Yes	Interface number for the address
`ipNetToMediaPhysAddress`	2	Yes	Physical address
`ipNetToMediaNetAddress`	3	Yes	IP address
`ifNetToMediaType`	4	Yes	Integer representing media type

In reading the first line of output, we learn that the IP address `10.152.0.44` has a cached Ethernet address of `0:0:1d:dc:85:2d`.

There are a couple of important things to note. First is that before the IP address, there is an interface index number. In this example, every address shown is on interface four. When making a query to determine the hardware address of a particular IP address, we'll need to know the interface it's on so that we can supply that part of the variable name. In practice, there are two ways to do this. One is to determine first which is the correct interface via some other SNMP query. On a router we might look up which interface we would route the packet to and assume that is the correct interface to use in looking up the hardware address. An easier, though brute-force, solution is to simply perform the lookup for every interface number on the device.

Also note that lookups can be performed only in the IP address to hardware address direction. If you wish to map a hardware address to an IP address, you must either retrieve the entire table until you find the hardware address you are looking for or hope that the vendor supplies a private MIB that allows reverse lookups, which most do not.

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

2.2.4. The Bridge MIB

The variable group starting with `dot1dBridge` is better known as the bridge MIB and is defined by RFC 1493. It is a direct child of `mib-2`, though it is not defined in RFC 1213. As expected, the bridge MIB contains information relevant to switching and bridging.[4] One table from the bridge MIB that is worthy of special mention is the forwarding database table, depicted in Figure 2.7. This table is indexed by physical address. For a particular address, `dot1dTpFdbPort` will return the port number to which the bridge will forward packets for that address. If that port and address mapping is learned dynamically by the bridge, `dot1dTpFdbStatus` is set to 3. Other values for `dot1dTpFdbStatus` indicate the port mapping was made differently. For example, a value of 4 indicates the address belongs to the bridge itself and is therefore statically configured.

[4] For all practical purposes, switching and bridging can be considered the same.

Figure 2.7. The `dot1dTpFdbTable`.

Variable	#	RW	Function
`dot1dTpFdbAddress`	1		Physical address
`dot1dTpFdbPort`	2		Associated port
`dot1dTpFdbStatus`	3		How the mapping was made

Sample output of `dot1dTpFdbTable` follows. The variable names have again been shortened; each variable name would begin with `dot1dBridge.dot1dTp.dot1dTpFdbTable.`

```
dot1dTpFdbEntry.dot1dTpFdbPort.8.0.9.50.98.244 = 25
dot1dTpFdbEntry.dot1dTpFdbPort.8.0.9.231.93.192 = 25
dot1dTpFdbEntry.dot1dTpFdbPort.8.0.32.5.9.147 = 7
dot1dTpFdbEntry.dot1dTpFdbPort.8.0.32.125.167.158 = 25
dot1dTpFdbEntry.dot1dTpFdbPort.8.0.32.138.134.127 = 25
dot1dTpFdbEntry.dot1dTpFdbPort.8.0.32.143.66.1 = 25
dot1dTpFdbEntry.dot1dTpFdbPort.8.0.32.147.187.7 = 13
dot1dTpFdbEntry.dot1dTpFdbPort.8.0.32.154.38.212 = 25
dot1dTpFdbEntry.dot1dTpFdbPort.8.0.32.160.9.193 = 1
dot1dTpFdbEntry.dot1dTpFdbPort.8.0.32.164.117.198 = 25
dot1dTpFdbEntry.dot1dTpFdbPort.8.0.32.178.66.134 = 25
dot1dTpFdbEntry.dot1dTpFdbPort.8.0.32.231.127.52 = 12
dot1dTpFdbEntry.dot1dTpFdbPort.8.0.43.28.153.246 = 25
dot1dTpFdbEntry.dot1dTpFdbPort.8.0.105.11.122.33 = 25
```

Be aware that the hardware addresses are shown here in decimal notation. The first line indicates that the hardware address `08:00:09:32:62:f4` (the hexadecimal version of `8.0.9.50.98.244`) is found behind port 25.

This table is immensely useful because it allows an operator to ask a switch which port a particular host is on. This is a vital tool for diagnosing and solving network problems. Neo, a network administration tool introduced in Chapter 4, provides a simple interface to perform these lookups.

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

2.3. Installing SNMP Tools

In order to make use of the management capabilities available from SNMP, you will need a set of tools that can make SNMP queries. At a minimum these tools will need to allow you to:

- Query a variable and view the response
- Set a variable and determine if it was successful
- Query entire tables with `get-next-request`
- Receive traps

It would also be beneficial if the tools could perform more advanced tasks, such as retrieving data from the `interfaces` group and presenting it in a clear format.

A widely used set of very good SNMP tools is the **net-snmp** package from the University of California at Davis. The project Web page is at http://www.net-snmp.org/. Until recently, these tools were called the **ucd-snmp** tools, and earlier versions still bear this name. Versions 4 and earlier are ucd-snmp; versions 5 and later are net-snmp.

The Web site for these tools has source distributions available, as well as binary distributions for some platforms. In particular, there are binary distributions available of version 5 for Linux and binary distributions of version 4 available for Linux, Solaris, HPUX, FreeBSD, Irix, and Windows. The most recent version is always available as a source distribution.

2.3.1. Building from Source

After the source distribution is downloaded, it must be uncompressed and extracted from the tar archive. This can be done as:

```
Solaris% gunzip -c net-snmp-5.0.8.tar.gz | tar xvf -
```

or on a system with gnu tar it can be done as:

```
Linux% gtar zxvf net-snmp-5.0.8.tar.gz
```

Of course, the actual filename will depend on the version you have chosen to download. In this case, a directory will be created called `net-snmp-5.0.8`. Change to that directory and type:

```
Solaris% ./configure
```

to configure net-snmp in preparation for building. After running for a while, it will ask you for input on several questions.

- **Default SNMP Version.** First, it will want to know the default version of SNMP to use, for which you can answer 1, 2, or 3. The version can always be overridden on the command line, so your answer will not prevent you from using any functionality later on. The default answer is version 3. In the examples in this chapter, we always set the version explicitly on the command line.
- **System Contact Information.** This will be the system contact returned for `system.sysContact.0` if you decide to run the SNMP daemon from the package. It is often set to the email address of the administrator.
- **System Location.** This will be the location returned for `system.sysLocation.0` if you decide to run the SNMP daemon from the package. It should be set to the physical location of the device.
- **Logfile Location.** This specifies the name of the file to which net-snmp will send logging information and error messages. The default answer is usually acceptable.

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

- **Snmpd Persistent Storage Location.** This is the name of the directory where net-snmp will keep statefull configuration files. The default answer is again acceptable.

2.3.2. Build and Install

After configuration is complete, you can type:

```
Solaris% make
```

to build the entire package. If you want to install it on your system, login to a root account and type:

```
Solaris# make install
```

By default, this will install into the following directories in `/usr/local/`:

- `bin/`: net-snmp applications
- `sbin/`: net-snmp daemons (`snmpd`, `snmptrapd`)
- `share/snmp/`: net-snmp configuration data
- `share/snmp/mibs/`: MIB files
- `lib/`: net-snmp programming libraries
- `include/`: net-snmp programming includes
- `man/`: net-snmp man pages

If you wish to place these files somewhere other than in `/usr/local`, you must run the configure script with the `--prefix` option, as in:

```
Solaris% ./configure --prefix=/usr/local/mydirectory
```

and then run the `make install`.

2.4. Using SNMP Tools

The following sections present examples of using the net-snmp tools. In each case, the program name is executed without a full pathname, assuming that `/usr/local/bin/` and `/usr/local/sbin/` are in your path. If they are not, you will need to type the full path to the program, as in:

```
Solaris% /usr/local/bin/snmpget -h
```

2.4.1. Snmpget

The `snmpget` program built by net-snmp can be used to retrieve the value of an SNMP variable. A simple example is:

```
Solaris% snmpget -v 1 -c public switch.example.com \
 system.sysUpTime.0
```


Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

```
SNMPv2-MIB::sysUpTime.0 = Timeticks: (405064255) 46 days, \
 21:10:42.55
```

The backslash is inserted only to break up the command line for clarity. The first two arguments specify that we want to use SNMP version one. The `-c` argument tells `snmpget` to use the following string as the SNMP community; in this case, the community is `public`. This is followed by the name of the device we wish to query and, finally, the name of the SNMP variable to look up. Remember that a final zero must be appended to the variable name when a simple variable is referenced.

Though it is easier to remember the textual variable name, you may sometimes want to use the numeric object ID instead. `snmpget` will allow you to do this:

```
Solaris% snmpget -v 1 -c public switch.example.com \
  .1.3.6.1.2.1.1.3.0
SNMPv2-MIB::sysUpTime.0 = Timeticks: (405108049) 46 days, \
  21:18:00.49
```

Here `snmpget` automatically looks up the variable name in the appropriate MIB and displays it for you. If it can't find the variable name in a MIB, it translates as much as possible and leaves the rest in numeric form:

```
Solaris% snmpget -v 1 -c public switch.example.com \
  .1.3.6.1.2.1.17.1.2.0
SNMPv2-SMI::mib-2.17.1.2.0 = INTEGER: 33
```

When this happens, it means `snmpget` cannot find the MIB that contains this particular variable. You can, however, obtain the MIB and direct net-snmp to use it. For more information on how to do this, see Section 2.4.6 later in this chapter.

The `snmpget` command has a number of other command line options available, listed in Figure 2.8. These options are also available for the `snmpset` and `snmpwalk` commands described in the next sections. Additionally, the man pages for `snmpget` and `snmpcmd`, installed with net-snmp, describe the options in more detail.

Figure 2.8. Options Common to `snmpget`, `snmpset`, and `snmpwalk`.

Argument	Function
`-h`, `--help`	Print the help message.
`-H`	Print configuration file options relevant to this program.
`-V`, `--version`	Print net-snmp software version number.
`-v` *version*	Specify SNMP version. Must be `1`, `2c`, or `3`.
`-c` *community*	Specify the SNMP community.
`-r` *retries*	Specify the number of times to retry requests.
`-t` *timeout*	Specify the number of seconds to wait for a response.
`-d`	Dump SNMP packets in hexadecimal.

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

Argument	Function
`-D` *module*[,...]	Debugging for the specified modules. Try `ALL`.
`-m` *mib*[:*mib*...]	Load named MIBs. Use `ALL` for everything.
`-M` *dir*[:*dir*...]	Include named directories when looking for MIBs.
`-P` *suboptions*	Set options for MIB parsing.
`-O` *suboptions*	Set output options.
`-I` *suboptions*	Set input options.
`-C` *suboptions*	Set application-specific options.

The `-O` option is particularly useful because it controls the output format that these tools use. It is used in conjunction with the suboptions listed in Figure 2.9, which is from the net-snmp documentation. For example, to instruct `snmpget` to print object IDs numerically:

Figure 2.9. Output Options for `snmpget`, `snmpset`, and `snmpwalk`.

Arg	Function
`a`	Print all strings in ASCII format
`b`	Do not break OID indexes down
`e`	Print enums numerically
`E`	Escape quotes in string indices
`f`	Print full OIDs on output
`n`	Print OIDs numerically
`q`	Quick print for easier parsing
`Q`	Quick print with equal-signs

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

Arg	Function
s	Print only last symbolic element of OID
S	Print MIB module-id plus last element
t	Print timeticks unparsed as numeric integers
T	Print human-readable text along with hex strings
u	Print OIDs using UCD-style prefix suppression
U	Don't print units
v	Print values only (not OID = value)
x	Print all strings in hex format
X	Extended index format

```
Solaris% snmpget -O n -v 1 -c public switch.example.com \
   system.sysUpTime.0
.1.3.6.1.2.1.1.3.0 = Timeticks: (2367889214) 274 days, 1:28:12.14
```

Or, to instruct it to print a full variable name:

```
Solaris% snmpget -O f -v 1 -c public switch.example.com \
   system.sysUpTime.0
.iso.org.dod.internet.mgmt.mib-2.system.sysUpTime.0 = Timeticks...
```

And of course, running `snmpget` or any of the other tools with the `-h` or `--help` flags will print a list of every option available.

2.4.2. Snmpset

The `snmpset` command can be used to set the value of a writable SNMP variable. For example, if we wish to set the system contact on a device, we can use `snmpset` as:

```
Solaris% snmpset -v 1 -c really-secret switch.example.com \
  system.sysContact.0 s admin@example.com
SNMPv2-MIB::sysContact.0 = STRING: admin@example.com
```

Here the `-v` and `-c` options are used just as with the `snmpget` command. Note that the community used is different from that in the previous example and is decidedly not `public`. On this device, the `really-secret` community has access to set SNMP variables, while `public` has access only to read SNMP variables.

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

As with `snmpget`, the next two arguments designate the device and variable we wish to query. For `snmpset`, though, we have two additional arguments at the end indicating what value we wish to set. The first argument specifies what kind of value the next argument will be. The `s` means the value will be a text string. Figure 2.10 lists all the possible values for this argument, which you can also view by running `snmpset -h`. Finally the last argument is the value we wish to set. In this case, we have set the system contact to be `admin@example.com`.

Figure 2.10. Value Type Arguments for `snmpset`.

Arg	Variable Type
i	Integer
u	Unsigned integer
t	Timeticks
a	IP address
o	Object ID
s	String
x	Hexadecimal string
d	Decimal string
b	Bits
U	Unsigned 64 bit integer
I	Signed 64 bit integer
F	Float
D	Double

2.4.3. Snmpwalk

The `snmpwalk` command provides a useful way to retrieve a contiguous segment of variables from a device. It uses the `get-next-request` PDU type to continue requesting the next variable until the entire segment is retrieved. For example, the entire system group can be obtained with:

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

```
Solaris% snmpwalk -v 1 -c public switch.example.com system
SNMPv2-MIB::sysDescr.0 = STRING: Cabletron Systems, Inc. ...
SNMPv2-MIB::sysObjectID.0 = OID: SNMPv2-SMI::enterprises ...
SNMPv2-MIB::sysUpTime.0 = Timeticks: (690848548) 79 days, ...
SNMPv2-MIB::sysContact.0 = STRING: admin@example.com
SNMPv2-MIB::sysName.0 = STRING: switch.example.com
SNMPv2-MIB::sysLocation.0 = STRING: 5-125T
SNMPv2-MIB::sysServices.0 = INTEGER: 71
```

If we wanted to retrieve the entire ARP cache from a router, we could do it with:

```
Solaris% snmpwalk -v 1 -c public router.example.com \
   ip.ipNetToMediaTable.ipNetToMediaEntry.ipNetToMediaPhysAddress
```

Additionally, we can retrieve every MIB-II variable on the system with the `snmpwalk` command if we leave off the final argument completely:

```
Solaris% snmpwalk -v 1 -c public router.example.com
```

Also useful is retrieving every private enterprise variable on a device:

```
Solaris% snmpwalk -v 1 -c public router.example.com enterprises
```

This is sometimes a useful way to find out what SNMP support a device might have if you do not have the MIB available. Recently, some vendors have been occasionally hiding variables from SNMP walks in an effort to obscure certain variables that are otherwise accessible with a direct `snmpget`. There is no trick to finding these "hidden" variables; you must obtain them from a MIB or some other published source of information.

2.4.4. Snmptrapd

The `snmptrapd` program is a daemon that listens for SNMP traps and either logs the messages to syslog or stores them in a file. If run with no arguments, it will send the messages to syslog by default. Make sure to run the program as root so that it can listen on the privileged port it requires:

```
Solaris# snmptrapd
```

Your prompt will return immediately as the program turns itself into a daemon. If you wish to run the program so that it stores trap messages in a file instead of sending them to syslog, use the `-o` option:

```
Solaris# snmptrapd -o /var/tmp/trapd.log
```

In order for `snmptrapd` to receive any data, you must have a device configured to send traps to the listening machine. This is done differently on different devices. Typically, there will be a place in the configuration where you can specify the IP address of one or more trap recipients as well as the community name that should be used. Any community name is acceptable; the community name will simply show up in the logfile. You can choose a secret community name in order to provide a small additional amount of security.

When logging to a file, `snmptrapd` will store a message like the following upon receiving a trap:

```
2003-05-14 23:36:39 W92-165T-SW-13.MIT.EDU [10.10.0.31] \
 (via 10.10.0.31) TRAP, SNMP v1, community public
 SNMPv2-SMI::mib-2.17 Enterprise Specific Trap (1) \
   Uptime: 274 days, 7:25:01.00
```

If `snmptrapd` is logging to syslog instead, it will store a message like this:

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

```
10.10.0.31: Enterprise Specific Trap (1) Uptime: 274 days ...
```

This trap is an enterprise-specific trap that reports the system up time.

The message format that `snmptrapd` uses can be changed with additional arguments to the program. See the man page for full details on how to do this.

2.4.5. Other Tools

In addition to `snmpget`, `snmpset`, and `snmpwalk`, the net-snmp package comes with a number of other useful tools. Included are:

- `snmpgetnext`: Perform a single get-next-request query
- `snmpbulkget`: Perform a bulk-get request (not for SNMPv1)
- `snmpbulkwalk`: Perform a bulk-get walk (not for SNMPv1)
- `snmpd`: An SNMP listening daemon
- `snmpdelta`: Real-time monitor for integer values
- `snmpdf`: Retrieve disk information from remote workstations
- `snmptrap`, `snmpinform`: Send an SNMP trap
- `snmpnetstat`: Produce network information, as in netstat
- `snmpstatus`: Print general device status information
- `snmptable`: Produce a nicely formatted SNMP table
- `snmptest`: For SNMP debugging
- `snmptranslate`: Print detailed MIB information

For example, `snmpnetstat` can be used to print the routing table on a device:

```
Solaris% snmpnetstat -r -v 1 -c public router.example.com
Routing tables
Destination   Gateway            Flags  Interface
default       ROUTER-2.EXAMPLE.C UG     GigabitEthernet1/2
10.7.10/24    ROUTER-3.EXAMPLE.C UG     GigabitEthernet1/1
10.7.14/24    ROUTER-3.EXAMPLE.C UG     GigabitEthernet1/1
10.7.15/24    ROUTER-3.EXAMPLE.C UG     GigabitEthernet1/1
10.7.16/24    ROUTER-3.EXAMPLE.C UG     GigabitEthernet1/1
10.7.17/24    ROUTER-3.EXAMPLE.C UG     GigabitEthernet1/1
10.7.21/24    ROUTER-3.EXAMPLE.C UG     GigabitEthernet1/1
10.9/23       ROUTER-4.EXAMPLE.C UG     FastEthernet3/7
10.11/23      ROUTER-4.EXAMPLE.C UG     FastEthernet3/7
```

Or `snmpdelta` can be used to monitor the number of packets coming into an interface:

```
snmpdelta -c public -v 1 switch.example.com ifInUcastPkts.6
IF-MIB::ifInUcastPkts.6 /1 sec: 1
IF-MIB::ifInUcastPkts.6 /1 sec: 1
IF-MIB::ifInUcastPkts.6 /1 sec: 1
IF-MIB::ifInUcastPkts.6 /1 sec: 1
IF-MIB::ifInUcastPkts.6 /1 sec: 26
IF-MIB::ifInUcastPkts.6 /1 sec: 20
IF-MIB::ifInUcastPkts.6 /1 sec: 1
IF-MIB::ifInUcastPkts.6 /1 sec: 1
```

And `snmptable` can be used to print an entire table nicely:

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

```
Solaris% snmptable -v 1 -c public switch.example.com ipNetTo...
SNMP table: IP-MIB::ipNetToMediaTable

ipNetToMediaIfIndex ipNetToMediaPhysAddress ipNetToMediaNet...
   27          0:5:dc:95:d0:a             10.7.21.106         ...
   27          0:5:dc:95:d0:a             10.7.21.108         ...
```

2.4.6. Dealing with MIBs

By default, the net-snmp tools store MIBs in `/usr/local/share/snmp/mibs`. The package comes with about 50 MIBs, but inevitably, you will find there are variables you want to use from other MIBs. First note that you do not need a MIB to access a variable as along as you access it by its numeric object ID. However, if you wish to use the textual name, it is necessary to have the appropriate MIB.

Where can you find a particular MIB? It depends on what you're looking for. All of the MIBs defined as IETF standards can be found at http://www.ietf.org/. Vendor MIBs are available from the vendor, sometimes by FTP or on the Web. One excellent source for MIBs is http://www.simpleweb.org/. It has all of the IETF MIBs well organized, as well as pointers to several vendor MIBs. Additionally, it has a nice tool for stepping through MIB variables and definitions.

Once you've downloaded a MIB, you can place it in the same directory as your other MIBs. You still must tell the net-snmp tools to look for it, though. If, for example, you download and install the bridge MIB in `/usr/local/share/snmp/mibs/` as `BRIDGE-MIB.txt`, you can tell `snmpget` to use it with:

```
Solaris% snmpget -m BRIDGE-MIB -v 1 -c public \
  switch.example.com dot1dBaseNumPorts.0
BRIDGE-MIB::dot1dBaseNumPorts.0 = INTEGER: 33
```

If we had left out the `-m BRIDGE-MIB` option, this query would not have worked.

2.4.7. Scripting with SNMP Tools

The net-snmp tools are ideal for use in network management scripts. For example, you may wish to write a script that monitors a set of UPSs. You can use `snmpget` to query the appropriate variables and warn an operator if a UPS goes on battery power.

When using SNMP tools in your scripts, bear in mind that on some smaller devices (and even on some larger ones), it is possible to overburden the processor with SNMP requests. As a result, you must take care not to run repeated `snmpwalk`'s or `snmpget`'s without a break, or at least ensure that it won't cause an operational problem for your device if you do.

Also note that if your MIB files are not located on the same machine as your script (as would be the case if they were on a networked file system, for example), they may not be available just when you want your program to notice a problem. Either store the MIBs locally on the machine or make sure you use the tools with numeric object IDs only (for both input and output) so that the MIBs are not needed for variable lookups.

2.5. Maintaining SNMP Tools

The net-snmp package requires little maintenance. Occasionally, you may wish to add a MIB or update the software, but other than that, there is no routine maintenance necessary.

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

2.6. References and Further Study

There are a number of books available that discuss SNMP in greater depth, including *Essential SNMP* (O'Reilly and Associates, 2001) by Douglas R. Mauro and Kevin J. Schmidt. Both *TCP/IP Illustrated, Volume 1* (Addison-Wesley, 1994) by W. Richard Stevens and *Internetworking with TCP/IP* (Prentice Hall, 2000) by Douglas Comer have sections with details about the SNMP protocol and associated standards.

RFCs 1155, 1156, and 1157, available from http://www.ietf.org/, are the original standards for SNMP. RFC 1157 defines SNMP itself, and RFC 1156 is the standard for MIB-I, now replaced by RFC 1213 for MIB-II. RFC 1155 defines the SMI and should now be read in conjunction with RFC 1212, which describes a concise format for use in a MIB.

Additionally, http://www.simpleweb.org/ has information on many MIBs, including a tool for browsing through MIB variables. It also has SNMP tutorials and references. Detailed information on using and writing MIBs can be found in the book *Understanding SNMP MIBS* (Prentice Hall PTR, 1996) by David Perkins and Evan McGinnis.

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003