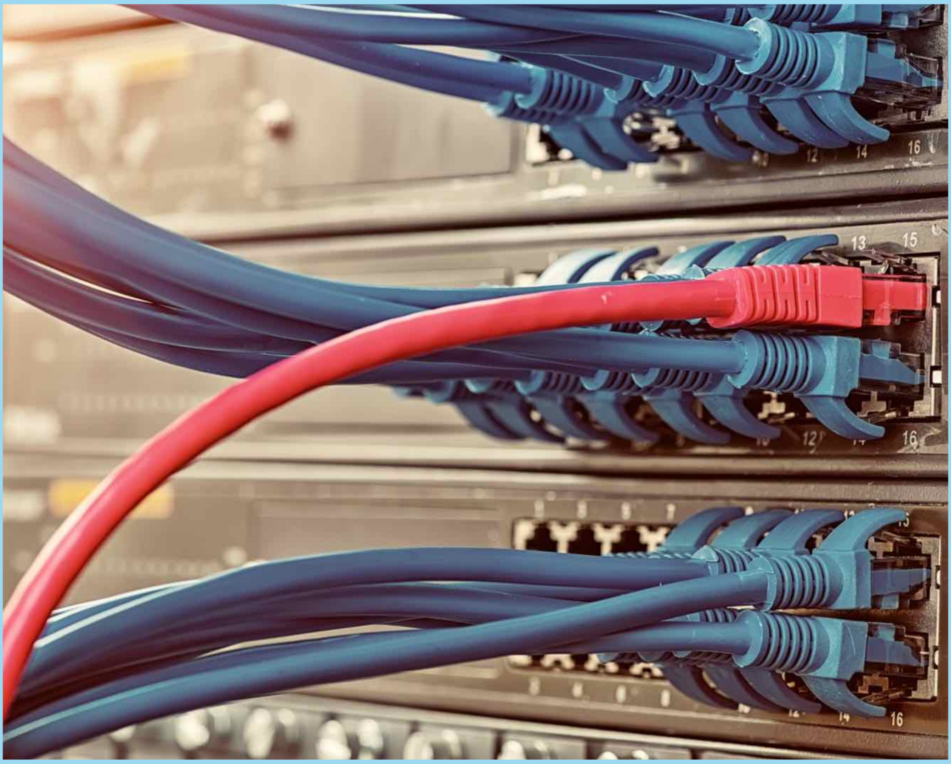


SNMP Fundamentals

Learn all about OIDs, MIBs, traps, polling and much more

2nd Edition- November 2020

SNMP Fundamentals

Your guide to the fundamentals of SNMP including terms, terminology and real-world usage examples

***By* EasySNMP.com**

Revision History

October 19th 2020 - 1st Edition

November 5th 2020 - 2nd Edition

Table of Contents

1. About this Book

SNMP is short for "Simple Network Management Protocol".

The protocol was original developed in late 1980s when it started to become obvious that there was a need for a standard way to monitor and manage network equipment from different vendors.

Although it has the word "Simple" in its name, SNMP has a reputation for being difficult to learn. This is partly due to the fact that the protocol uses specific terms and terminology that are not often found in other information technology fields. Terms like "OID", "MIB" and "symbolic name" are obscure to many readers and don't immediately convey meaning.

This ebook will introduce you to the fundamentals of the SNMP protocol. We'll start with a brief overview of some terminology and then will dive into real-world examples showing you how to use SNMP to get valuable information from your different kinds of networking gear.

By studying each chapter in this ebook and using the examples provided, you'll become familiar with the terms used by SNMP and will gain a working knowledge of how to perform common operations using SNMP-based tools.

2. Introducing SNMP

SNMP is the main protocol used by network professionals to monitor their networking equipment.

One of its major strengths is its flexibility. The original designers of the protocol made sure that it could be expanded and adapted to monitor all kinds of networking gear and IT equipment. As a result you can use SNMP to get data not only from standard network devices like switches and routers but also from firewalls, printers, servers, power equipment, cooling systems and even many of the very latest IoT-based sensors.

Another strength is its broad adoption. Before the advent of SNMP, every vendor had its own protocol. As a result interoperability was almost non-existent. While it is still true that some vendors prefer to implement their own custom protocols, almost all of them also support SNMP.

These two strengths, flexibility and broad adoption, have helped make SNMP the universal language for network monitoring and management.

Since it was first designed and released, SNMP has grown to the point where it would be unthinkable for a major network equipment vendor to release a product that did not have at least basic support for it.

Established vendors like Cisco, HP, Dell and HP have not only embraced SNMP, they have contributed to it by participating in the standards body that guides the evolution of the protocol and by proposing their own extensions.

3. SNMP Terminology

Terms and Terminology

Let's start with a review of some of the terms and terminology that you are likely to encounter as you start to work with SNMP. Some of them may be unfamiliar at first but with the help of this chapter you will quickly gain an understanding of them, which will help you in the subsequent chapters.

OIDs

The term OID is a short for "Object Identifier" or "Object ID" and an OID is just a set of numbers separated by periods.

An example of an OID that you'll frequently run into is:

```
1.3.6.1.2.1.1.5.0
```

That string of number can be used to retrieve the name that has been assigned to an SNMP-enabled device. It's also commonly referred to as the system name or sysName OID.

OIDs have a hierarchical structure that is very similar to how files and folders are organized in a file system.

In other words, "1.3.6.1.2.1.1.5.0" is an OID that is nine levels deep where each level is represented by a number and is separated by a period.

Symbolic Names

As you can imagine, it's hard to remember that "1.3.6.1.2.1.1.5.0" is what you use to get the system name value. To make it is easier to know what you are looking at, SNMP supports the concept of symbolic names.

A symbolic name is just a string that's easier to read than a long series of numbers. For example, to get the system name value you can use this symbolic name:

```
SNMPv2-MIB::sysName.0
```

As you can see, the symbolic name immediately conveys more information than the numeric OID.

Under the hood, the SNMP tool that you use to get a value will translate the symbolic name into the associated OID and then retrieve value.

MIBs

MIB is short for "Management Information Block". In the previous section showed you how the symbolic name "SNMPv2-MIB::sysName.0" is mapped to the numeric OID "1.3.6.1.2.1.1.5.0". That mapping is done using a MIB file.

MIBs are text files that describe how that mapping is performed. SNMP-based tools usually have the ability to accept new MIB files and use them to start monitoring values described in them.

For example, if a vendor releases a new kind of networking equipment and they want to make a new set of values available through SNMP, they'll assign each of those values its own numeric OID and then they will create a MIB file that assigns symbolic names to them. They will usually then make that MIB file available through their web site or support portal. If you download the associated MIB file then you can use the symbolic names instead.

It's important to note that you don't need a MIB file to retrieve values from a device. Every device makes all of its values accessible using numeric OIDs. Having a MIB file can make it much easier to understand the results that you get from the device so, in general, if a MIB file is available you will want to download and use it.

Some software tools will describe how they "compile" MIBs. This generally just means that they will take a standard MIB and convert it into a different format for their own internal use. Other tools will only ask that you place new MIB files in a specific directory where they will be automatically picked up and used.

Community Strings

You can think of a community string as a sort of password. It's the secret code that a device expects to see before it will respond to any incoming requests for SNMP data.

When you enable SNMP on a device, the default community string is usually "public". Most devices will let you change this to a different value. It's good practice to change it to something else as soon as possible because it's sometimes possible to get sensitive data from a device using SNMP, including IP addresses and other network details that you may not want to divulge.

Using a custom community string can be thought of as a first-line of defense to protect access to your devices.

4. Getting SNMP Data

Command Line vs Graphical Tools

To get SNMP data from your devices you can use command line tools or graphical tools.

In the category of graphical tools, there are a number of different options. Searching for "SNMP browser" in Google or your favorite search engine will turn up a number of options, some free and some paid.

Most Linux distros have command line SNMP tools built-in by default or they are easily installed using the distro's package manager. For Windows, we recommend downloading the Net-SNMP set of tools which can be found at http://www.net-snmp.org. All recent versions of MacOS come with command line SNMP tools installed by default.

Most graphical SNMP browsers use the standard command line tools under the hood so the examples in this ebook will use the command line tools as well.

By understanding how the command line versions work you will get a better understanding of how the SNMP protocol works and what options are available for it.

Using "snmpget"

The first command line tool you want to become familiar with is called "snmpget". This tool is used to retrieve values from a device and display them on the command line.

Let's use snmpget to retrieve the system name for a device by opening a command line window and running the following command:

```
snmpget -v 1 -c public 10.0.1.1 1.3.6.1.2.1.1.0
```

In the above example, the -v parameter tells snmpget to use SNMPv1 and the -c parameter tells it to use "public" for the community string.

Following the community string is the IP address of the device to query and lastly we have the OID whose value we want to retrieve.

Running the above command should result in something like the following output:

```
SNMPv2-MIB::sysName.0 = STRING: My Router
```

As you can see, the snmpget tool found the value we asked for. It not only found it but also detected that the value was a string type and it gave us the string value which is "My Router".

In our example we used the numeric OID but we could have used the symbolic name instead. Here is how the command looks using the symbolic name for the same value:

```
snmpget -v 1 -c public 10.0.1.1 SNMPv2-MIB::sysName.0
```

Since the symbolic name and the numeric OID refer to the same value, the output from this command is exactly the same:

```
SNMPv2-MIB::sysName.0 = STRING: My Router
```

You can easily get other values. Below is an example of how to get the system's description, which is a longer value that tells us a bit more about the device. This example, and the ones that follow in this ebook will show both the command and the resulting output:

```
command:
snmpget -v 1 -c public 10.0.1.1 SNMPv2-MIB::sysDescr.0

result:
SNMPv2-MIB::sysDescr.0 = STRING: Demo Router in Lab 4
```

As with the system name, the snmpget command reported that it found a string value, this case "Demo Router in Lab 4".

Getting Multiple Values

In the above examples we showed you how to use snmpget to retrieve a single value. You can also use snmpget to retrieve multiple values by including more OIDs or symbolic values after the first one.

For example, this will retrieve the system and description in one command:

```
command:
snmpget -v 1 -c public 10.0.1.1 SNMPv2-MIB::sysName.0
SNMPv2-MIB::sysDescr.0

result:
SNMPv2-MIB::sysName.0 = STRING: My Router
SNMPv2-MIB::sysDescr.0 = STRING: Demo Router in Lab 4
```

Lists of Values

In the previous section we showed you how to get multiple values using snmpget, but the assumption was that you already knew which individual values you were looking for.

Of course, there are some cases where you don't know in advance which values are available in the first place.

For example, although a switch or a router might have a fixed set of physical interfaces where you can connect network cables, almost all switches also have support for what are called virtual interfaces.

Virtual interfaces have many different uses and applications in the world of networking but, for the purposes of this ebook, the most important thing to know is that the total number of interfaces on a device might not correspond to the number of interfaces that are physically present on the hardware.

To give some context to this, look at the following snmpget command which retrieves the description for one interface:

```
command:
snmpget -v 1 -c public 10.0.1.1 IF-MIB::ifDescr.1

result:
IF-MIB::ifDescr.1 = STRING: eth0
```

In the above example, we asked for the description for the first interface (as defined by the .1 at the end of the symbolic name) and snmpget returned "eth0".

To get the name of the next interface, we can use this snmpget command which uses .2 in the symbolic name:

```
command:
snmpget -v 1 -c public 10.0.1.1 IF-MIB::ifDescr.2

result:
IF-MIB::ifDescr.3 = STRING: itf0
```

The problem is that we don't know how many interfaces the device has. We can guess based on the number of ports on the device but the number of virtual interfaces is unknown.

Of course we could just keep trying "ifDescr.3", "ifDescr.4", etc., until we stop getting results, but that doesn't sound like a very efficient approach.

Instead, we can use a different SNMP tool called "snmpwalk".

Using "snmpwalk"

The snmpwalk tool solves the problem of finding values when you don't know exactly how many are present. It can even be used to scan an unknown device and find the complete list of values it supports. In this section we'll show you how to achieve both of these goals, but first we'll look at the syntax and options used with snmpwalk.

The syntax of the snmpget and snmpwalk commands can be confusingly similar but with experience you will gain familiarity with both.

Compare the following snmpwalk example with the ones provided earlier for snmpget:

```
command:
snmpwalk -v 1 -c public 10.0.1.1 1.3.6.1.2.1.1

result:
SNMPv2-MIB::sysDescr.0 = STRING: Demo Router in Lab 4
SNMPv2-MIB::sysContact.0 = STRING: Network Admin
SNMPv2-MIB::sysName.0 = STRING: My Router
SNMPv2-MIB::sysLocation.0 = STRING: Main Office
SNMPv2-MIB::sysServices.0 = INTEGER: 14
```

The command is almost the same but the meaning of the last parameter is different. With snmpget, the last value you specify is the exact OID or symbolic name of the value you want to retrieve. With snmpwalk the last parameter is instead a starting point where the utility should begin looking for values.

As you can see in the result, snmpwalk returned a list of items showing both the symbolic name and the associated value for each.

Most SNMP-enabled devices make hundreds and sometimes thousands of values available for monitoring. In the above example we saw that just 5 values were returned.

That's because we used a fairly specific starting point. The OID 1.3.6.1.2.1.1 specifies the base of the "system" section so that's why we see only those values.

If we use a more general starting point, we get more values:

```
command:
snmpwalk -v 1 -c public 10.0.1.1 1.3.6.1.2.1

result:
SNMPv2-MIB::sysDescr.0 = STRING: Demo Router in Lab 4
SNMPv2-MIB::sysContact.0 = STRING: Network Admin
SNMPv2-MIB::sysName.0 = STRING: My Router
SNMPv2-MIB::sysLocation.0 = STRING: Main Office
SNMPv2-MIB::sysServices.0 = INTEGER: 14
IF-MIB::ifPhysAddress.1 = STRING:
IF-MIB::ifPhysAddress.2 = STRING: 44:d9:e7:7:40:68
IF-MIB::ifPhysAddress.3 = STRING:
IF-MIB::ifPhysAddress.4 = STRING: 44:d9:e7:7:40:62
IF-MIB::ifAdminStatus.1 = INTEGER: up(1)
IF-MIB::ifAdminStatus.2 = INTEGER: up(1)
IF-MIB::ifAdminStatus.3 = INTEGER: up(1)
IF-MIB::ifAdminStatus.4 = INTEGER: up(1)
...
```

We've truncated the output to just 14 entries but on our test device a total of about 2000 items were returned.

If you want to get all values that the device supports then you can leave out the last parameter as shown in this example:

```
command:
snmpwalk -v 1 -c public 10.0.1.1
```

As you can see, snmpwalk provides a great mechanism to see what values a device supports. The length of the starting point you specify allows you to select how specific you want to be.

When running an snmpwalk command, it's often useful to redirect the output to a text file. The following example shows how to accomplish this:

```
command:
snmpwalk -v 1 -c public 10.0.1.1 > output.txt
```

After the above command has finished running, the complete results will be stored in the output.txt file which you can view using your preferred text editor.

Most of the time, snmpwalk will be able to use its default set of MIB files to translate the number OIDs it finds into more readable symbolic names.

When it can't you will instead see full or partial number OIDS listed.

Here is an example of a part OID:

```
SNMPv2-SMI::mib-2.3.1.1.1.11.1.10.77.251.4 = INTEGER: 11
```

Based on the first portion of the OID, snmpwalk was able to determine that it is part of the SNMPv2-SMI::mib-2 set of OIDs but could not be more specific. This is because the MIB file(s) required for further translation were not present.

Sometimes full numeric OIDs might be returned as shown in this example:

```
.1.3.6.1.2.1.4.31.3.1.30.2.11 = Counter32: 90659
```

If the required MIB files were present then snmpwalk would be able to return this instead:

```
IP-MIB::ipIfStatsOutTransmits.ipv6.11 = Counter32: 90659
```

Without the MIBs, all we have to work with is a string of numbers. With the MIBs we get enough information in the symbolic name to better understand what we are looking at.

The "ipIfStats" portion suggests that the value is related to TCP/IP interface statistics. The portion that reads "OutTransmits" suggests that this is a value counting the number of outgoing transmissions. The last portion "ipv6.11" tells us its related to IPv6 networking and the counter is for the interface with index 11.

Armed with the MIB files we can get even more detail. Recall that MIB files are just text files so you can view them with any text editor.

If you search for and download the IP-MIB file and search for "ipIfStatsOutTransmits" you will find the following description of the value:

```
ipIfStatsOutTransmits OBJECT-TYPE
   SYNTAX     Counter32
   MAX-ACCESS read-only
   STATUS     current
DESCRIPTION
"The total number of IP datagrams that this entity
supplied to the lower layers for transmission.  This
includes datagrams generated locally and those forwarded
by this entity.
Discontinuities in the value of this counter can occur at
re-initialization of the management system, and at other
times as indicated by the value of
ipIfStatsDiscontinuityTime."
```

Using the MIB file we now have a complete understanding of the counter's meaning and even information about another value that is related to its use.

As you have seen, MIB files can help substantially to interpret the results that you get from snmpwalk.

Whenever you are dealing with a new piece of hardware, visit the support section of the manufacturer's web site. There you will find the MIB files for the device.

Most versions of snmpwalk will look for MIB files in a folder called "MIBS" which is either located under the folder where the snmpwalk is located or at another location defined by a command line environment variable.

If you don't see a MIBs folder, check the documentation for the toolset that you are using and follow the instructions provided there.

5. SNMP Versions and History

SNMP Protocol Versions

So far, the examples we have shown in this ebook have all used the first version of the SNMP protocol which, in short form, is usually referred to as SNMPv1. In addition to SNMPv1 there are two other versions that you will commonly encounter when working with SNMP. In this chapter we will review each version and explain their histories and purposes.

SNMPv1

Version 1 is the version that was defined in the original specification of the SNMP protocol when it was published in 1987 and it is still widely in use today.

This is quite remarkable when you consider that it predates not only the internet but pretty much the entire computing industry as we know it. To put this in context, 1987 was the year IBM introduced the PS/2 with an innovative 3.5 inch floppy disk drive, the year MSDOS 3.3 was released and the year that "Star Trek: The Next Generation" premiered on network television.

As we've seen in previous chapters, the flexibility of the SNMP protocol was one of its original strengths and this has also contributed to its longevity.

When you first start working with a new kind of network device, we recommend that you first try scanning it using snmpwalk with version 1 and the community string "public" as seen in the following example:

```
command:
snmpwalk -v 1 -c public 10.0.1.1
```

Many devices are configured out-of-the-box to respond to a request like this. For those that are not, they usually only require that you go to their settings to enable SNMP.

Although the designers of the original protocol ensured that it was flexible enough to work under many conditions and for many purposes, they could not think of everything in advance.

As technology evolved, two different classes of requirements emerged and subsequent versions of the SNMP protocol were released to address them. The first of of these new requirements was for larger data types and the second was for security.

SNMPv2c

Version 2c of the SNMP protocol was released in 1993 and it was designed to address a limitation that became apparent as networks became faster and could support increasingly higher bandwidth rates.

In the original version of the protocol, all counters and numeric values were 32 bits in size.

Probably the most important use of counters in SNMP is to keep track of bandwidth rates on each interface. This allows networking professionals to detect which connections are under the most stress and guides not only their day-to-day operations but their long term planning as well.

Bandwidth is usually expressed in multiples of bits per second, such as kilobits per second, megabits per second, gigabits per second, etc. Under the hood though, networking equipment uses "octets" to record bandwidth. An octet is the networking equivalent of a byte in that it is composed of eight bits.

Whenever data is transmitted on a network interface, the device updates a counter adding the number of bytes transmitted to the previous value of the counter.

With counters that are 32-bits in length, this allows the device to keep track of up to 4 gigabytes of traffic. In 1987, a time when 2400 baud modems were the standard, this was an astronomical amount, but on today's fiberoptic networks that amount can be transmitted in a fraction of a second.

The 32-bit limit made it impossible to accurately measure bandwidth on newer pieces of network equipment because as soon as 4 gigabytes of traffic had passed through an interface, the counter would reset to zero.

SNMPv2c was developed to address several different issues but the counter size limit in SNMPv1 is probably the most important one. In SNMPv2c, a new counter type adds support for 64-bit counters which also have a limit but since that limit is 18,446,744,073,709,551,615 (two to the power of 64 minus 1), it safely handles the bandwidth rates of all current networks and almost anything we can currently imagine for the future.

Telling snmpget or snmpwalk to use SNMPv2c is easy. Simply use "2c" after the -v parameter instead of "1".

```
command:
snmpwalk -v 2c -c public 10.0.1.1
```

If your device supports SNMPv2, you'll see results that look a lot like what you get with version 1 but with the addition of new counters and values.

You might be wondering what the "c" in SNMPv2c stands for and why it is there. Around the time that the need for a new version of SNMP became clear, there were multiple competing groups each with their own take on how to address the problems emerging with the origin protocol.

During this development phase, each variant adopted a different letter so as to distinguish it from the rest, resulting in SNMPv2u, SNMPv2p and others. In the end SNMPv2c became the one to be widely adopted and it is the one we use now.

SNMPv3

A different set of industry changes drove the development of SNMPv3.

As you learned in Chapter 3, access to SNMP data on a device is protected by a community string. Community strings are the only security option available in SNMPv1 and SNMPv2c.

If you don't know the correct community string for a device then you can't access the data that it holds. In fact, with SNMP if you specify the wrong community string, most devices will not even reply to your request leaving you to wonder if the device is online at all.

This sounds acceptable until we dig a bit deeper and consider some real-world issues.

The first issue is that the data provided by a device using SNMP can often be sensitive in nature. It almost certainly includes IP addresses for the device and other devices on the network. It might also include locations, contact names and firmware versions numbers which could be used to identify device vulnerabilities that could be exploited.

Many devices are configured out-of-the-box to respond to requests using the community string "public", making it easy to readily get this information but, even if you use a non-default community string, there are still risks involved.

First of all, when you make a request for SNMP using v1 or v2c, the data is sent in the clear. That means that an attacker with a protocol sniffer on your network could see all of the data that you request. Even worse, the community string itself is sent in the clear. Armed with your community string, the attacker can make their own requests to get whatever data they want.

The SNMPv3 protocol was designed with security in mind and addresses these issues.

Let's start by looking at a sample snmpwalk request that uses SNMPv3:

```
command:
snmpwalk -v 3 -l authPriv -u username -a SHA -A
"authphrase" -x AES -X "privphrase" 10.10.60.50
```

First, you will notice that there are new command line parameters and you'll also notice that there's no -c parameter for a community string. Community strings are not used in SNMPv3. Instead, it requires a user name and a pair of passphrases which are used for authentication and to protect the request and its results.

Setting up a device to use SNMPv3 is a bit more complicated than in other versions where all you have to do is specify a community string.

First you need to specify a user name. Along with it you select an authentication passphrase, an authentication protocol, a privacy passphrase and a privacy protocol.

The user name is just what it sounds like, a string identifying the user making the request.

The authentication protocol passphrase is a password that is used to validate the connection request. The authentication protocol tells snmpwalk which security technique to use to encode the passphrase in transit.

The privacy protocol tells snmpwalk how to encrypt the data that it sends and how to decrypt the response that it receives. The privacy passphrase is the password used for encryption purposes which can be, and usually is, different from the authentication passphrase.

All of these combine together to ensure that your SNMP data can be requested and received using secure techniques.

When to Use Each Version

If you have just received a new SNMP device, and you want to scan it to learn a bit about it, start with SNMPv1. It will allow you to get all the fundamental device data that you need to understand what you are dealing with.

SNMPv1 is also great for retrieving non-numeric data or counters that are expected to always have small values. Remember that the largest value a counter can have in

SNMPv1 is around 4 billion. So if all you need to retrieve is the number of interfaces on a device, SNMPv1 is going to be just fine.

In some cases you might not have a choice. Some basic devices, for example a UPS or PDU, might only support SNMPv1 because they have no requirement for the features offered by newer versions.

If you need bandwidth data then you're going to need SNMPv2c. Its support for 64-bit counters will give you the full bandwidth insight that you'll need for any modern device. Plus, with its community-based authentication, it has the benefit of being just as easy to use as SNMPv1.

If security is paramount in your organization, then there's no alternative to SNMPv3. It's more difficult to set up, it's often only available in enterprise-level networking gear, and it's more complicated to use, but it has all of the features and benefits of SNMPv2 plus top notch security layered on top.

6. Summary

SNMP is an extremely powerful and useful protocol for all kinds of network monitoring and management.

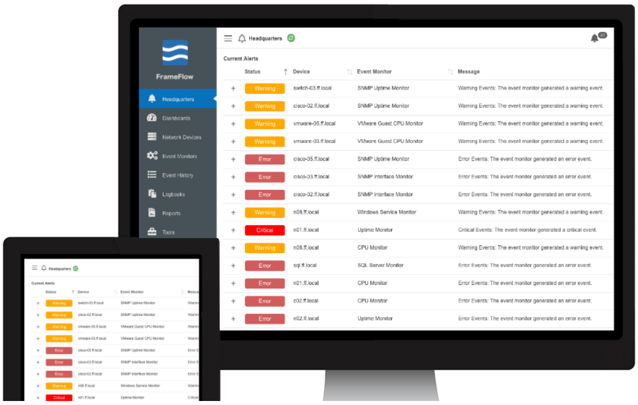
The examples shown in the previous sections of this ebook showed you how to connect to SNMP-enabled devices and retrieve the values that they support.

We hope you've enjoyed this ebook.

This book is sponsored by FrameFlow.

For more SNMP-related resources FrameFlow at https://www.frameflow.com/. There, you can also find out more about their enterprise monitoring solution with deep support for all versions of SNMP.

About Our Sponsor



Headquarters

Current Alerts

	Status	Device	Event Monitor	Message
+	Warning	switch-03.ff.local	SNMP Uptime Monitor	Warning Events: The event monitor generated a warning event.
+	Warning	cisco-02.ff.local	SNMP Uptime Monitor	Warning Events: The event monitor generated a warning event.
+	Warning	vmware-05.ff.local	VMware Guest CPU Monitor	Warning Events: The event monitor generated a warning event.
+	Warning	vmware-03.ff.local	VMware Guest CPU Monitor	Warning Events: The event monitor generated a warning event.
+	Error	cisco-05.ff.local	SNMP Uptime Monitor	Error Events: The event monitor generated an error event.
+	Error	cisco-03.ff.local	SNMP Interface Monitor	Error Events: The event monitor generated an error event.
+	Error	cisco-02.ff.local	SNMP Interface Monitor	Error Events: The event monitor generated an error event.
+	Warning	n08.ff.local	Windows Service Monitor	Warning Events: The event monitor generated a warning event.
+	Critical	n01.ff.local	Uptime Monitor	Critical Events: The event monitor generated a critical event.
+	Warning	n08.ff.local	CPU Monitor	Warning Events: The event monitor generated a warning event.
+	Error	sql.ff.local	SQL Server Monitor	Error Events: The event monitor generated an error event.
+	Error	n01.ff.local	CPU Monitor	Error Events: The event monitor generated an error event.
+	Error	c02.ff.local	CPU Monitor	Error Events: The event monitor generated an error event.
+	Error	n02.ff.local	Uptime Monitor	Error Events: The event monitor generated an error event.

FrameFlow is IT monitoring software that helps you to make sure your critical systems are up and running. With deep support for SNMP, FrameFlow is the preferred solution for monitoring network equipment but also can monitor servers, printers, power equipment, cooling systems and much more.

To learn more visit us at https://www.frameflow.com