

Systems Administration Toolkit: Using SNMP data

Skill Level: Intermediate

[Martin Brown \(mc@mcslp.com\)](mailto:mc@mcslp.com)

Professional writer

Freelance

15 Apr 2008

The Simple Network Management Protocol (SNMP) is built in to many devices, but often the tools and software that can read and parse this information are too large and complicated when you only want to check a quick statistic or track a particular device or issue. This article looks at some simplified methods for getting SNMP information from your devices and how to integrate this information into the rest of your network's data map.

About this series

The typical UNIX® administrator has a key range of utilities, tricks, and systems he or she uses regularly to aid in the process of administration. There are key utilities, command line chains, and scripts that are used to simplify different processes. Some of these tools come with the operating system, but a majority of the tricks come through years of experience and a desire to ease the system administrator's life. The focus of this series is on getting the most from the available tools across a range of different UNIX environments, including methods of simplifying administration in a heterogeneous environment.

SNMP basics

There are many ways you can monitor your UNIX server. See the [Resources](#) for some examples of the type of monitoring available. Monitoring a single server is not a problem, but monitoring the same information across a number of servers can present problems. If one of the servers you are in charge of runs out of disk space, you want to know about it before it starts to affect your users and clients.

Monitoring multiple servers in this way, especially if they use a variety of different operating systems, can be a problem. The differences in command line tools, output formats, values, and other information all complicate what should otherwise be a simple process. What is required is a solution that provides a generic interface to the information that works, irrespective of the UNIX variant you are using.

The Simple Network Management Protocol (SNMP) provides a method for managing information about different systems. An agent runs on each system and reports information using SNMP to different managing systems.

SNMP is often a built-in component for network devices such as routers and switches, and is the only method available for retrieving statistics and status information remotely (without logging in to some sort of interface). On most hosts you will need to explicitly run SNMP software to expose information about the host over the SNMP protocol.

Information can be retrieved from an agent either explicitly, by requesting the information using a GET request, or the agent can broadcast information to management systems using the TRAP or INFORM messages. In addition, managing systems can set information and parameters on the agent, but this is usually only used to change the network configuration.

The types of information that can be shared can be quite varied. It can be everything from network settings, statistics, and metric data for network interfaces, through to monitoring CPU load and disk space.

The SNMP standard does not define what information the agent returns; instead, the available information is defined by Management Information Bases (MIBs). The MIB defines the structure of the information that is returned, and are organized into a hierarchical structure using object identifiers (OID). You access information within an agent by requesting data using a specific location within the MIB structure.

For example, some of the more common IDs are shown in Listing 1.

Listing 1. SNMP object IDs

```
sysDescr.0      1.3.6.1.2.1.1.1.0
sysObjectId.0  1.3.6.1.2.1.1.2.0
sysUpTime.0    1.3.6.1.2.1.1.3.0
sysContact.0   1.3.6.1.2.1.1.4.0
sysName.0      1.3.6.1.2.1.1.5.0
sysLocation.0  1.3.6.1.2.1.1.6.0
sysServices.0  1.3.6.1.2.1.1.7.0
ifNumber.0     1.3.6.1.2.1.2.1.0
```

You can see from this list that the MIBs are numerical and, effectively, in sequence. When obtaining information you can use a GET request to obtain a specific value, or GETNEXT to get the next property from the last one you read. You can also use the

names. The names shown above are all part of the system tree, so you can read the value by getting using the OID 'system.sysUpTime.0'.

The values that you read are also of specific types. You can read integer, floating point, and string values that are all defined as 'scalar' objects. Within these objects are types that are identified with specific significance. For example, time interval values are reported as 'timeticks,' or hundredths of a second. These values need to be converted into a more readable human form before being displayed. There are also MIB objects that return tabular data. This is handled by returning additional OID instances that can be grouped together to make an SNMP table.

From a security perspective, SNMP agents can be associated with a specific community, and managing systems access information by using the community as a method of validating their access to the agent. In Version 1 of the SNMP standard, the community string was the only method of securing or restricting access. With Version 2 of the SNMP standard, the security was improved, but could be complex to handle. With Version 3, considered the current version since 2004, the standard was improved with explicit authentication and access control systems.

Getting SNMP statistics

There are many different ways of obtaining information from SNMP systems, including using professional management tools, programming interfaces, and command line tools.

Of the latter, probably the best known and easiest to use is the `snmpwalk` command, which is part of a larger suite of SNMP tools that allow you to obtain information from SNMP agents directly from the command line. This command will walk the entire subtree of a given management value and return all the information about the system contained within the subtree.

For example, Listing 2 shows the output when querying a local system for all the information within the 'system' tree.

Listing 2. 'Walking' an SNMP tree

```
$ snmpwalk -Os -c MCSLP -v 1 localhost system
sysDescr.0 = STRING: Linux tweedledum 2.6.23-gentoo-r8
                #1 SMP Tue Feb 12 16:32:14 GMT 2008 x86_64
sysObjectID.0 = OID: netSnmpAgentOIDs.10
sysUpTimeInstance = Timeticks: (34145553) 3 days, 22:50:55.53
sysContact.0 = STRING: root@Unknown
sysName.0 = STRING: tweedledum
sysLocation.0 = STRING: serverroom
sysORLastChange.0 = Timeticks: (0) 0:00:00.00
sysORID.1 = OID: snmpFrameworkMIBCompliance
sysORID.2 = OID: snmpMPDCompliance
sysORID.3 = OID: usmMIBCompliance
sysORID.4 = OID: snmpMIB
```

```
sysORID.5 = OID: tcpMIB
sysORID.6 = OID: ip
sysORID.7 = OID: udpMIB
sysORID.8 = OID: vacmBasicGroup
sysORDescr.1 = STRING: The SNMP Management Architecture MIB.
sysORDescr.2 = STRING: The MIB for Message Processing and Dispatching.
sysORDescr.3 = STRING: The management information definitions for
                the SNMP User-based Security Model.
sysORDescr.4 = STRING: The MIB module for SNMPv2 entities
sysORDescr.5 = STRING: The MIB module for managing TCP implementations
sysORDescr.6 = STRING: The MIB module for managing IP and ICMP implementations
sysORDescr.7 = STRING: The MIB module for managing UDP implementations
sysORDescr.8 = STRING: View-based Access Control Model for SNMP.
sysORUpTime.1 = Timeticks: (0) 0:00:00.00
sysORUpTime.2 = Timeticks: (0) 0:00:00.00
sysORUpTime.3 = Timeticks: (0) 0:00:00.00
sysORUpTime.4 = Timeticks: (0) 0:00:00.00
sysORUpTime.5 = Timeticks: (0) 0:00:00.00
sysORUpTime.6 = Timeticks: (0) 0:00:00.00
sysORUpTime.7 = Timeticks: (0) 0:00:00.00
sysORUpTime.8 = Timeticks: (0) 0:00:00.00
```

You can see here a range of information about the host, including the operating system (in `sysDescr.0`), the amount of time that the system has been available (`sysUpTimeInstance`), and the location of the machine. The interval time here is shown in both its original value (timeticks) and the converted, human-readable days, hours:minutes:seconds.

The uptime or availability of a machine is a very common use for SNMP, as it provides probably the most convenient and efficient method for determine whether a machine is up and processing requests. Other solutions that have been described in past parts of the series include ping or using `rwho` and `ruptime`. These latter two solutions are very CPU and network intensive and not very friendly in terms of their resource utilization.

Note, however, the limitation of the uptime described here, which is the information shown in the uptime of the SNMP agent, not the uptime of the entire machine. In most situations the two are same, especially for devices with built-in SNMP monitoring, such as network routers and switches. For computers that expose their status through SNMP, there may be a discrepancy between system and SNMP agent uptime.

You can get a quicker idea of the status of a machine through SNMP using `snmpstatus`. This obtains a number of data points from a specified SNMP agent, including the IP address, description, uptime, and network statistics (packets sent/received, and IP packets sent/received). For example, if we look at a Solaris host, you can see the simplified information, as shown in Listing 3.

Listing 3. Simplified information

```
$ snmpstatus -v1 -c public t1000
[192.168.0.26]=>[SunOS t1000 5.11 snv_81 sun4v] Up: 2:12:10.20
Interfaces: 4, Recv/Trans packets: 643/160 | IP: 456/60
```

```
2 interfaces are down!
```

This machine has recently been rebooted (hence the low uptime and packet statistics). The `snmpstatus` command has also determined that two of the interfaces on the machine (which has four Ethernet ports) are down. This is a good example of the sort of warning information that SNMP can provide to help notify you of an issue that requires further investigating.

For obtaining a specific piece of information, you can use the `snmpget` command, which reads one or more OIDs directly and reports their value. For special types, it will also convert to a human-readable format. For example, to get the system description and uptime, use the following command (in Listing 4).

Listing 4. Getting system description and uptime information

```
$ snmpget -v1 -c public t1000 system.sysUpTime.0 system.sysContact.0
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (867411) 2:24:34.11
SNMPv2-MIB::sysContact.0 = STRING: "System administrator"
```

In isolation, all of these methods are useful, but in reality, you need to be able to monitor and track multiple machines and multiple OIDs to get a full picture of what is going on. We can do this by using one of the many programmable interfaces to SNMP.

Getting SNMP data programmatically

The `Net::SNMP` module for Perl obtains information from one or more agents using SNMP. Other, similar, interfaces are available for other languages, including Python, Ruby, and PHP (see [Resources](#)). The interface works by you creating a session that communicates (and if necessary authenticates) with the SNMP agent on the desired host. Once you have an active and valid session, you can request data from the agent directly for one or more OIDs. The information is returned in the form of a hash of information, tied between the OID and the corresponding value.

Listing 5 shows a very simple script that will obtain the system uptime for each of the hosts supplied on the command line.

Listing 5. Getting a single SNMP agent property with Perl and Net::SNMP

```
#!/usr/local/bin/perl
use strict;
use Net::SNMP;
my $uptimeOID = '1.3.6.1.2.1.1.3.0';
```

```
foreach my $host (@ARGV)
{
    my ($session, $error) = Net::SNMP->session(
        -hostname => $host,
        -community => 'public',
        -port     => 161
    );

    warn ("ERROR for $host: $error\n") unless (defined($session));

    my $result = $session->get_request(
        -varbindlist => [$uptimeOID]
    );

    if (!defined($result))
    {
        warn ("ERROR: " . $session->error . "\n");
    }
    else
    {
        printf("Uptime for %s: %s\n", $host, $result->{$uptimeOID});
    }

    $session->close;
}
```

In the script, we've provided the full numerical OID for the system, sysUpTime property. You have to supply the list of OIDs to obtain when using the `get_request()` method as a reference to an array, and then pull the information back out from the hash that is returned. In Listing 5 we build the array reference dynamically during the call, and then use the OID as the hash key when printing out the result.

Using the script, we can get a list of the uptimes for each host supplied on the command line (see Listing 6).

Listing 6. List of uptimes for each host

```
$ perl uptime.pl tweedledum t1000
Uptime for tweedledum: 4 minutes, 52.52
Uptime for t1000: 6 minutes, 26.12
```

Of course, watching this information manually is hardly efficient.

Tracking SNMP data over time

Viewing a single instance of an SNMP OID property at one time is not always very useful. Often you want to monitor something over time (for example, availability), or you want to monitor for changes in particular values. A good example is disk space. SNMP can be configured to record all sorts of information, and disk space is a common system to want to monitor so that you can identify not only when the disk space reaches a particular level, but also when there is a significant change to the disk space, which might signify a problem.

For example, Listing 7 shows a callback-based solution to constantly monitor the disk space. In the script, we output a running total, but it could be configured to only output the warning message that is triggered when there is a reduction in the disk space.

Listing 7. Getting a running view of SNMP properties

```
#!/usr/local/bin/perl

use strict;
use warnings;
use Net::SNMP qw(snmp_dispatcher);

my $diskSpaceOID = '1.3.6.1.4.1.2021.9.1.7.1';

foreach my $host (@ARGV)
{
    my ($session, $error) = Net::SNMP->session(
        -hostname => $host,
        -nonblocking => 0x1,
    );

    if (!defined($session))
    {
        warn "ERROR: $host produced $error - not monitoring\n";
    }
    else
    {
        my ($last_poll) = (0);

        $session->get_request(
            -varbindlist => [$diskSpaceOID],
            -callback => [
                \&diskSpace_cb, \&$last_poll
            ]
        );
    }
}

snmp_dispatcher();

exit 0;

sub diskSpace_cb
{
    my ($session, $last_poll) = @_;

    if (!defined($session->var_bind_list))
    {
        printf("%-15s ERROR: %s\n", $session->hostname, $session->error);
    }
    else
    {
        my $space = $session->var_bind_list->{$diskSpaceOID};

        if ($space < ${$last_poll})
        {
            my $diff = (($last_poll-$space)/${$last_poll})*100;
            printf("WARNING: %s has lost %0.2f%% disk space\n",
                $session->hostname,$diff);
        }

        printf("%-15s Ok (%s)\n",
            $session->hostname,
            $space);
    }
}
```

```
        );  
    }  
    ${$last_poll} = $space;  
}  
  
$session->get_request(  
    -delay      => 60,  
    -varbindlist => [$diskSpaceOID]  
);  
}
```

The script is in two parts, and uses some functionality within the `Net::SNMP` module that allows you to call a function when an SNMP value is obtained from a host, coupled with the ability to continually monitor hosts and SNMP objects in a simple, but efficient, loop.

The first part sets up each host to monitor the information. We are only monitoring one piece of information, but we could monitor others as part of the solution. The object is configured as 'non-blocking,' so that the script will not wait if the host cannot be reached, but simply move on to the next host. Finally, in the call to `get_request()`, we submit the callback information. The first argument here is the name of the function to be called when the response is received from the agent. The second is an argument that will be supplied to the function when it is called.

We'll use this argument to be able to record and track the previous value returned by the SNMP call. Within the callback function, we compare the newly returned value and the previous value. If there's a reduction, we calculate the percentage reduction and then report a warning.

The final part of the callback is to specify that another retrieval should occur, here specifying that the next retrieval should be delayed by 60 seconds. The existing callback information is retained. In effect, the script obtains the value from the SNMP agent, calls the callback function, which then queues up another retrieval in the future. Because the same callback is already defined, the process repeats in an endless loop.

Incidentally, the script uses the `dskAvail` OID value, and calculates the percentage difference based on the last and new values. The `dskTable` tree that this property is part of actually has a disk percentage property that we could have queried, instead of calculating it manually. However, the value returned is probably not finely grained enough to be useful.

You can see this property and current values by using `snmpwalk` to output the `dskTable` tree, which itself is part of the UCD MIB (Listing 8).

Listing 8. Getting a dump of available MIB data

```
$ snmpwalk -v 1 localhost -c public UCD-SNMP-MIB::dskTable  
UCD-SNMP-MIB::dskIndex.1 = INTEGER: 1
```

```
UCD-SNMP-MIB::dskPath.1 = STRING: /
UCD-SNMP-MIB::dskDevice.1 = STRING: /dev/sda3
UCD-SNMP-MIB::dskMinimum.1 = INTEGER: 100000
UCD-SNMP-MIB::dskMinPercent.1 = INTEGER: -1
UCD-SNMP-MIB::dskTotal.1 = INTEGER: 72793272
UCD-SNMP-MIB::dskAvail.1 = INTEGER: 62024000
UCD-SNMP-MIB::dskUsed.1 = INTEGER: 7071512
UCD-SNMP-MIB::dskPercent.1 = INTEGER: 10
UCD-SNMP-MIB::dskPercentNode.1 = INTEGER: 3
UCD-SNMP-MIB::dskErrorFlag.1 = INTEGER: noError(0)
UCD-SNMP-MIB::dskErrorMsg.1 = STRING:
```

To find the property in the first place, you can dump all the known properties by using `snmptranslate`. By filtering this with `grep` we can see the information we want:

```
$ snmptranslate -Ts |grep dsk.
```

To get a numerical value, use `snmptranslate` and provide the name with the `-On` option (see Listing 9).

Listing 9. Using `snmptranslate`

```
$ snmptranslate -On UCD-SNMP-MIB::dskAvail
.1.3.6.1.4.1.2021.9.1.7
```

Running the script, we get a running commentary (and warnings) for the disk space usage on the specified host. See Listing 10.

Listing 10. Monitoring disk space automatically

```
$ perl disk-space-auto.pl tweedledum
tweedledum      Ok (50319024)
WARNING: tweedledum has lost 2.67% disk-space)
tweedledum      Ok (48976392)
WARNING: tweedledum has lost 1.65% disk-space)
tweedledum      Ok (48166292)
tweedledum      Ok (48166292)
tweedledum      Ok (48166292)
tweedledum      Ok (48166292)
```

You can see from this output that we have lost some significant space out of the space available on this disk on the specified host. To monitor more hosts, just add more hostnames on the command line.

Publishing information through an SNMP agent

The SNMP package includes a daemon, `snmpd`, which can be configured to expose a variety of information using the SNMP protocol. The configuration for the information to be exposed is controlled using the `/etc/snmpd.conf` file.

For example, Listing 11 shows the `snmpd.conf` file on the host used in the earlier

examples in this article.

Listing 11. Sample snmpd.conf file

```
syslocation serverroom
proc imapd 20 10
disk / 100000
load 5 10 10
```

Each of these lines populates different information. In the example, we set the location of the machine, and then configure some specific items to monitor.

The proc section monitors a specific process, shown here as a monitor for the IMAP daemons for a mail service. The numbers following the option specify the maximum number of processes allowed to be running, and the minimum number that should be running. You can use this to make sure that a particular service is running, and that you haven't exceeded capacity that might indicate a fault. When the process count goes above the MAX value, an SNMP trap is generated.

For the disk, you specify the path to the directory to be monitored and the minimum size (in kilobytes) that the disk should have free. Again, an SNMP trap is triggered if the disk space dips below this value.

Finally, the load information shows the maximum CPU load for 1, 5, and 15 minutes that should be reported. This is equivalent to the output of the uptime command that shows the process loading for these intervals. Like the other configured limits, a trap is raised when these limits are exceeded.

Manually setting this information is not difficult, but also not ideal. A simple menu-based solution, snmpconf, is available if you want a more straightforward method of setting the configuration.

Summary

Monitoring your servers and devices is a process that can be very complex, especially as the number of devices in your network increases. SNMP is an efficient, and extensible, method for exposing and reporting this information. Because the interface is consistent across all the devices, you can get uptime, network statistics, disk space, and even process monitoring using the same methods across multiple hosts.

In this article we've looked both at the basics of SNMP and also how to read specific values from different hosts. Using the Net::SNMP perl module we have also examined methods for reading information, using both one-hit and continual monitoring-based solutions. Finally, we examined the methods for configuration additional information to be exposed on a system so that you can customize and

monitor the systems you need for your network when using the snmpd daemon.

Resources

Learn

- [System Administration Toolkit: Monitoring User Usage](#) (Martin Brown, developerWorks, October 2007) looks at ways of examining the utmp and wtmp files for information about users and their activities
- [System Administration Toolkit: Monitoring Mail Usage](#) (Martin Brown, developerWorks, December 2007) gives examples on monitoring the system log for mail deliveries and information
- [System Administration Toolkit: Monitoring Disk Usage](#) (Martin Brown, developerWorks, June 2006) covers many techniques for investigating the disk space used on your machine.
- Read [System Administration Toolkit: Standardizing your UNIX command-line tools](#) (Martin Brown, developerWorks, May 2006) to learn how to use the same command across multiple machines.
- The Python SNMP Framework provides access to SNMP agents for Python programmers (<http://pysnmp.sourceforge.net/>).
- Ruby SNMP is an interface to SNMP through Ruby (<http://snmplib.rubyforge.org/>).
- PHP provides built-in support for SNMP, providing you have already installed the Net-SNMP package. The [documentation](#) provides details on how to use the SNMP functionality.
- [System Administration Toolkit: Time and event management](#) (Martin Brown, developerWorks, May 2006) covers the creation and organization of time scripts using cron and at.
- Read [Scheduling recurring tasks in Java](#) (Tom White, developerWorks, November 2003) to learn how to build a simple, general scheduling framework for task execution conforming to an arbitrarily complex schedule.
- For an article series that will teach you how to program in bash, see [Bash by example, Part 1: Fundamental programming in the Bourne again shell \(bash\)](#) (Daniel Robbins, developerWorks, March 2000), [Bash by example, Part 2: More bash programming fundamentals](#) (Daniel Robbins, developerWorks, April 2000), and [Bash by example, Part 3: Exploring the ebuild system](#) (Daniel Robbins, developerWorks, May 2000).
- [System Administration Toolkit](#): Check out other parts in this series.
- [Making Unix and Linux work together](#) (Martin Brown, developerWorks, April 2006) is a guide to getting traditional Unix distributions and Linux working together.

- Different systems use different tools, and the IBM Redbook [Solaris to Linux Migration: A Guide for System Administrators](#) will help you identify some key tools.
- [Exploring the Linux memory model](#) (Vikram Shukla, developerWorks, January 2006) helps you understand how Linux uses memory, swap space and exchanges pages and processes between the two.
- [New to AIX and UNIX](#): Visit the New to AIX and UNIX page to learn more about AIX and UNIX.
- The [developerWorks AIX and UNIX zone](#) hosts hundreds of informative articles and introductory, intermediate, and advanced tutorials.
- [AIX 5L Wiki](#): A collaborative environment for technical information related to AIX.
- Stay current with [developerWorks technical events and webcasts](#).
- [Technology bookstore](#) Browse this site for books and other technical topics.

Get products and technologies

- The [Net-SNMP package](#) provides the tools for querying snmp agents, and the snmpd daemon for setting up your own agent on your system for disk, process and other monitoring types
- The [Net::SNMP Perl module](#) provides a Perl interface to SNMP agent information.

About the author

Martin Brown

Martin Brown has been a professional writer for over eight years. He is the author of numerous books and articles across a range of topics. His expertise spans myriad development languages and platforms -- Perl, Python, Java, JavaScript, Basic, Pascal, Modula-2, C, C++, Rebol, Gawk, Shellscript, Windows, Solaris, Linux, BeOS, Mac OS/X and more -- as well as Web programming, systems management and integration. Martin is a regular contributor to ServerWatch.com, LinuxToday.com and IBM developerWorks, and a regular blogger at Computerworld, The Apple Blog and other sites, as well as a Subject Matter Expert (SME) for Microsoft. He can be contacted through his Web site at <http://www.mcslp.com>.