

Introduction

Purpose

This How-To is one in a series designed to serve as a reference for getting started with OpenNMS. Eventually, these documents will cover everything necessary to get OpenNMS installed and running in your environment.

Copyright

Content is available under a [Creative Commons Attribution-NonCommercial-ShareAlike2.5 License](https://creativecommons.org/licenses/by-nc-sa/2.5/).

Corrections and Omissions

Please submit any corrections and omissions to the author.

Overview

OpenNMS is an enterprise-grade network management platform developed under the open-source model. Unlike traditional network management products which are very focused on network elements such as interfaces on switches and routers, OpenNMS focuses on the services network resources provide: web pages, database access, DNS, DHCP, etc. (although information on network elements is also available).

Since the majority of network services are provided using the TCP/IP protocol, OpenNMS is very IP-centric. The basic monitored "element" is called an "interface", and an interface is uniquely identified by an IP address. Services are mapped to interfaces, and if a number of interfaces are discovered to be on the same device (either via SNMP or SMB) then they may be grouped together as a "node".

Discovery in OpenNMS consists of two parts: discovering an IP address to monitor and then discovering the services supported by that IP address. The first part is much simpler than the second.

[-] Discovery

[-] The Discovery Configuration File

Discovery in OpenNMS is controlled by the `discovery-configuration.xml` file (located in the `/opt/OpenNMS/etc` directory).

Let's look at that file:

```
<discovery-configuration threads="1" packets-per-second="1"
                        initial-sleep-time="300000"
                        restart-sleep-time="86400000"
                        retries="3" timeout="800">

  <include-range retries="2" timeout="3000">
    <begin>192.168.0.1</begin>
    <end>192.168.0.254</end>
  </include-range>

  <include-url>file:/opt/OpenNMS/etc/include</include-url>
</discovery-configuration>
```

Now, all this file controls is a process that will send an ICMP "ping" to a particular set of IP addresses. If there is a response within the timeout, a "new suspect" event is generated. Otherwise, the IP address is ignored.

The global discovery attributes are:

threads

This is the number of threads that will be used for discovery. By default this is set to 1.

packets-per-second

This is the number of ICMP packets that will be generated each second. The default is 1. Note that there is a relationship between the packets-per-second and the number of threads. If your network has an average latency of 500ms, then setting packets-per-second to 2 would double the speed at which NewSuspect messages were created. But if there is only one thread available, setting this number to 3 would have little effect - the single thread would be processing as many packets as it could as fast as it could.

initial-sleep-time

This is the time, in milliseconds, before the discovery process will commence after OpenNMS is started (by default 5 minutes). This delay is put in place to allow the product to fully start before generating new events.

restart-sleep-time

Once the discovery process has completed, this is the time, in milliseconds, before it will start again. By default, the process will repeat 24 hours after the last discovery run has completed.

timeout

this is the amount of time, in milliseconds, that the discovery process will wait for a response from a given IP address before deciding that there is nothing there. This can be overridden later in the file.

retries

this is the number of attempts that will be made to query a given IP address before deciding that there is nothing there. This can be overridden later in the file.

Once the defaults are in place (defaults meaning the global values that will be used if they are not overridden in the tags below), the only thing left to tell the discovery process is which IP addresses to try. This is controlled by four different tags:

specific

Specify a IP address to be discovered. Multiple specific tags can be used.

```
<specific>ip-address</specific>
```

Where *ip-address* is the address you want discovered.

include-range

Specify a range of IP addresses to be discovered. Multiple include-range tags can be used.

```
<include-range>  
  <begin>start-ip-address</begin>  
  <end>end-ip-address</end>  
</include-range>
```

Where *start-ip-address* is the beginning of a range to be scanned and *end-ip-address* is the end of that range.

exclude-range

Specify a range of IP address to be excluded from discovery.

```
<exclude-range>  
  <begin>start-ip-address</begin>  
  <end>end-ip-address</end>  
</exclude-range>
```

Where *start-ip-address* is the beginning of a range to be excluded and *end-ip-address* is the end of that range. Note that the exclude-range tag will only override addresses in an include-range. It will not override specific IP addresses or addresses included in a file. There is no "specific" version of the exclude tag - if you want to exclude a specific IP address use an exclude-range where the beginning and ending IP addresses are the same.

include-url

Specify a file containing IP addresses to be included in discovery.

```
<include-url>file:filename</include-url>
```

Where *filename* is the full path to a text file listing IP addresses, one to a line. Comments can be imbedded in this file. Any line that begins with a "#" character will be ignored, as will the remainder of any line that includes a space followed by "#".

All tags are optional and unbounded (you can have as many as you wish).

[] Another Way to Discover Interfaces

Now that the discovery configuration file has been explained, there are two short-comings that need to be pointed out. First, any changes to this file, like most of the configuration files within OpenNMS, requires that OpenNMS be restarted. Second, what if you want to discover a service, such as a web server, on a device you cannot ping?

Remember that all the discover process does is generate a newSuspect event. Included in the `/opt/OpenNMS/bin` directory is a Perl script called `send-event.pl`. You can use this script to generate an internal NewSuspect event - bypassing the discovery process altogether. Combined with a script, you could generate any number of NewSuspect events (just make sure that the IP address really does have some services that can be monitored by OpenNMS. Otherwise, you will have an interface in the system with no services associated with it).

The format of the `send-event.pl` is as follows:

```
/opt/opennms/bin/send-event.pl --interface ip-address
uei.opennms.org/internal/discovery/newSuspect
```

Replace *ip-address* with the address you want discovered.

For csv with FQDN as hosts location

```
cat fqdnlist.csv | awk -F "," '{print "host " $1}' | sh | awk '{print "./send-
event.pl --interface " $4 " uei.opennms.org/internal/discovery/newSuspect"}'
```

For csv file with IP for host location

```
cat ipaddressss.csv | gawk -F "," '{print "host " $1 }' | sh | gawk '{print
"./send-event.pl --interface " $1 "
uei.opennms.org/internal/discovery/newSuspect"}'
```

note: use same format as exported assets.

Depending on what Perl modules you have installed, you may get an error running this script (such as a complaint about `Getopt::Mixed`). To automatically add the necessary modules, try:

```
perl -MCPAN -e 'install mod_name'
```

Replace *mod_name* with the name of the missing module.

[[-](#)] Logs

You can watch the discovery process by examining the `discovery.log` file in the `/opt/opennms/logs` directory.

[[-](#)] Capabilities

Okay, if the discovery process just generates NewSuspect events, what does all the work? This would be the capabilities daemon, `capsd`. `capsd` is responsible for discovering all the services to be monitored, such as `httpd`, `DNS`, etc., as well as if any collectors are present (at the time this is only `SNMP`).

The capsd process is controlled by the `capsd-configuration.xml` file. This file consists of some basic parameters and a collection of "protocols" to be tested. If the protocol is not in the file, then OpenNMS will not discover it.

[] Process Parameters for capsd

The first few lines of the `capsd-configuration.xml` file control how capsd will behave.

```
<capsd-configuration rescan-frequency="86400000"  
    initial-sleep-time="300000"  
    management-policy="managed"  
    max-suspect-thread-pool-size="6"  
    max-rescan-thread-pool-size="3"  
    abort-protocol-scans-if-no-route="false">
```

rescan-frequency

capsd will continue to check each interface to see if new services have been added. The frequency of these rescans is controlled by this parameter. The default value is 24 hours in milliseconds.

initial-sleep-time

like the discovery process, capsd will sleep for a certain amount of time after OpenNMS starts. The default value is 5 minutes in milliseconds.

management-policy

this parameter controls the default behavior of capsd. If it is set to "managed", then all IP addresses in NewSuspect events will be scanned, unless included in an "unmanaged" range defined at the end of this file. If this parameter is set to "unmanaged", then all NewSuspect events will be ignored unless the IP address in the event is expressly included in a "managed" range (also defined at the end of this file).

max-suspect-thread-pool-size

This value determines how many threads will be created to perform capability scans on IP addresses supplied by NewSuspect events. Increasing this value will make the initial discovery move more quickly at the cost of more system resources.

max-rescan-thread-pool-size

This value determines how many threads will be created to perform capability scans on interfaces that have already been discovered. Rescans are either automatically scheduled (see `rescan-frequency`) or generated ad hoc through the Web UI.

abort-protocol-scans-if-no-route

This is an extremely important parameter for modifying the behavior of capsd. When attempting to connect to a specific port to test for a service, it is possible to receive a "no route to host" exception. In theory, this is because the host is not reachable, but in practice any number of things, such as firewalls, can cause this error. If this parameter is set to "false", these "no route to host" messages are ignored. But if it is set to "true", then capsd will stop checking for additional services. This can greatly improve the speed of discovery if the capsd file has been "tuned" (discussed below).

[] Protocols

OpenNMS tests the existence of a particular network service through the use of "protocols". At the most basic, this could be a connection to a TCP port to test for a particular banner, but there are also special classes for a variety of other protocols. The current protocols supported out of the box are:

- Citrix
- DHCP
- DNS
- Domino IIOP
- FTP
- General Purpose (script based)
- HTTP
- HTTPS
- ICMP
- IMAP
- JBOSS
- JDBC
- JDBC Stored Procedure
- JSR160
- K5
- LDAP
- Microsoft Exchange
- MX4J
- Notes HTTP
- NSClient (Nagios Agent)
- NRPE (Nagios Remote Plugin Executor)
- NTP
- POP3
- Radius
- SMB
- SMTP
- SNMP
- SSH
- TCP

When a newSuspect event is received by capsd and the management policy for the IP address in that event is "managed," the capsd process will work its way through this file testing one protocol after another, in the order they are listed in this file. The first protocol to be tested is ICMP:

```
<protocol-plugin protocol="ICMP"
    class-name="org.opennms.netmgt.capsd.IcmpPlugin"
    scan="on" user-defined="false">
  <property key="timeout" value="2000"/>
  <property key="retry" value="2"/>
</protocol-plugin>
```

Each protocol starts with a `protocol-plugin` tag. This tag has four attributes:

`protocol`

This is the name of the protocol.

`class-name`

This defines the protocol class that will be used to test for the service.

`scan`

Capsd scans can be turned "on" or "off" per protocol with this attribute.

`user-defined`

The Web UI allows for the dynamic creation of new services. This attribute tracks whether or not the service was added by the user.

In addition, each `protocol-plugin` can have a number of properties defined by a `key` and a `value`. The possible properties for each protocol will be discussed in the next section, although almost all include a timeout value and the number of times to try to make a connection.

There is a little-known feature available in capsd. This is the ability to configure each protocol based on IP addresses. This is through the `protocol-configuration` tag. The best way to describe this is through an example. Let's take the ICMP configuration from above and modify it:

```
<protocol-plugin protocol="ICMP"
  class-name="org.opennms.netmgt.capsd.IcmpPlugin"
  scan="on" user-defined="false">
  <protocol-configuration scan="on" user-defined="false">
    <range begin="192.168.10.0" end="192.168.10.254"/>
    <property key="timeout" value="4000"/>
    <property key="retry" value="3"/>
  </protocol-configuration>

  <protocol-configuration scan="off" user-defined="false">
    <range begin="192.168.20.0" end="192.168.20.254"/>
  </protocol-configuration>

  <protocol-configuration scan="enable" user-defined="false">
    <specific>192.168.30.1</specific>
  </protocol-configuration>

  <property key="timeout" value="2000"/>
  <property key="retry" value="2"/>
</protocol-plugin>
```

There are three `protocol-configuration` tags that have been added. Suppose you have one subnet that is over a slow link and it may take a little longer for an ICMP request to be returned. In the first example, the 192.168.10.0 subnet is allowed a 4 second response instead of the default of 2, and three retries.

Suppose you have another segment that you just don't want to scan for ICMP. In the second example, scan is set to "off", and that range will not be tested for ICMP.

Finally, the third example demonstrates setting scan to "enable", which forces the protocol to be associated with the device without testing for it. This is useful if you know the protocol is going to exist on a device, but for some reason it has not been added yet or it is down.

[[-\]](#) Plugin Properties

The following table shows all of the `property` tags that are available for each protocol plugin. The default values are the ones hard-coded into the plugin itself, not the defaults in the configuration file.

[[-\]](#) Citrix

port

The port to connect to. Default is "1494".

timeout

The time in milliseconds to wait for a response. The default is "5000".

retries

The number of attempts made to detect the service. The default is "0".

[[-\]](#) DHCP

port

The port to connect to. Default is "67".

timeout

The time in milliseconds to wait for a response. Default is "3000".

retries

The number of attempts made to detect the service. Default is "3".

[[-\]](#) DNS

port

The port to connect to. Default is "53".

timeout

The time in milliseconds to wait for a response. Default is "3000".

retries

The number of attempts made to detect the service. Default is "3".

lookup

The default host name to attempt to resolve. Default is "localhost".

[[-\]](#) Domino IIOP

ports

The port to connect to. Default is "63148".

timeout

The time in milliseconds to wait for a response. Default is "5000".

retries

The number of attempts made to detect the service. Default is "0".

IOR port

Port to look for the IOR via HTTP. Default is "80".

[+] FTP

port

The port to connect to. Default is "21".

timeout

The time in milliseconds to wait for a response. Default is "5000".

retries

The number of attempts made to detect the service. Default is "0".

[+] HTTP

ports

The port to connect to (can be more than one, separated by a comma). Default is "80,8080,8000".

timeout

The time in milliseconds to wait for a response. Default is "5000".

retries

The number of attempts made to detect the service. Default is "0".

max-ret-code (1.3.2 and later)

The highest numerical HTTP response code that will be taken to indicate success. Default is 399 if a URL is specified, 600 if not.

check-return-code (1.3.2 and later)

Boolean indicating whether or not to check the HTTP response code for success/failure. Default is "true". Note that illegal return codes ($99 \leq \text{code} \leq 600$, per [RFC1945](#)) still indicate failure.

[+] HTTPS

ports

The port to connect to (can be more than one, separated by a comma). Default is "443".

timeout

The time in milliseconds to wait for a response. Default is "30000".

retries

The number of attempts made to detect the service. Default is "1".

max-ret-code (1.3.2 and later)

The highest numerical HTTP response code that will be taken to indicate success. Default is 399 if a URL is specified, 600 if not.

check-return-code (1.3.2 and later)

Boolean indicating whether or not to check the HTTP response code for success/failure. Default is "true". Note that illegal return codes ($99 \leq \text{code} \leq 600$, per [RFC1945](#)) still indicate failure.

[[-\]](#) ICMP

timeout

The time in milliseconds to wait for a response. Default is "800".

retries

The number of attempts made to detect the service. Default is "2".

[[-\]](#) IMAP

port

The port to connect to. Default is "143".

timeout

The time in milliseconds to wait for a response. Default is "5000".

retries

The number of attempts made to detect the service. Default is "0".

[[-\]](#) JBOSS

factory

The method of connecting to JMX. Default is "RMI". The other acceptable value is "HTTP".

timeout

The time in milliseconds to wait for a response. Default is "3000".

version

The version of JBOSS being detected. Default is "4".

port

The TCP port to use for the connection. Default is "1099".

[[-\]](#) JDBC

Unlike nearly all the other plugins, the JDBC plugin is *highly* unlikely to work with the default configuration values. You will have to configure user, password, url and driver to match your database before this will work.

timeout

The time in milliseconds to wait for a response. Default is "5000".

retries

The number of attempts made to detect the service. Default is "0".

user

The username with which to authenticate to the database. Default is "sa"

password

The password corresponding to the username. Default is blank

url

The url of the database (JDBC url format). Default is

"jdbc:sybase:Tds:OPENNMS_JDBC_HOSTNAME/tempdb"

driver

The JDBC driver class to create the connection from. Default is "com.sybase.jdbc2.jdbc.SybDriver"

host

The host the database lives on. Default is "OPENNMS_JDBC_HOSTNAME"

[+] JDBC Stored Procedure

Configuration is as for the JDBC plugin, except there is an additional parameter to define the stored procedure to run. Caveats regarding configuration of the JDBC plugin apply here also. The additional parameter:

stored-procedure

The name of the stored procedure to run after connecting to the database. Default is "isRunning". The stored procedure must have a single output parameter of type java.sql.Types.BIT. The actual return value is discarded

[+] JSR160

timeout

The time in milliseconds to wait for a response. Default is "5000".

LDAP

version

The LDAP version to test for. Default is "3".

timeout

The time in milliseconds to wait for a response. Default is "5000".

retries

The number of attempts made to detect the service. Default is "0".

Microsoft Exchange

timeout

The port to connect to. Default is "5000".

retries

The number of attempts made to detect the service. Default is "0"

pop3 port

The port to look for the POP3 protocol. Default is "110".

imap port

The port to look for the IMAP protocol. Default is "143".

mapi port

The port to look for the MAPI protocol. This port/service is used by Exchange for doing RPC over HTTP. Default is "593".

MX4J

timeout

The time in milliseconds to wait for a response. Default is "5000".

Notes HTTP

ports

The port to connect to (can be more than one, separated by a comma). Looks for the string "Notes" in the banner. Default is "80,8080,8000".

timeout

The time in milliseconds to wait for a response. Default is "5000".

retries

The number of attempts made to detect the service. Default is "0".

NRPE

command

The command to send to the NRPE agent. Default is the NRPE Hello command ("_NRPE_CHECK")

port

The port the NRPE agent is listening on. Default is "5666"

padding

The padding to use in the packet. Default is 2

timeout

The time in milliseconds to wait for a response. Default is "5000".

retry

The number of attempts made to detect the service. Default is "0".

NSClient

command

The command to send to the NSClient agent. Default is the client version check ("1").

port

The port on which the agent is listening. Default is "1248"

parameter

A parameter to send along with the command. Default is null

criticalPercent

If the command sent returns a value which can be compared, this value is the comparison value for a critical level. Default is "0"

warningPercent

If the command sent returns a value which can be compared, this value is the comparison value for a warning level. Default is "0"

password

The password needed to connect to the agent. Default is "None"

timeout

The time in milliseconds to wait for a response. Default is "5000".

retry

The number of attempts made to detect the service. Default is "0".

NTP

port

The port to connect to. Default is "123".

timeout

The time in milliseconds to wait for a response. Default is "3000".

retries

The number of attempts made to detect the service. Default is "3".

POP3

port

The port to connect to. Default is "110".

timeout

The time in milliseconds to wait for a response. Default is "5000".

retries

The number of attempts made to detect the service. Default is "0".

Radius

authport

The port the radius daemon uses for authentication. Default is 1812

acctport

The port the radius daemon uses for accounting. Default is 1813

authtype

The type of authentication the radius daemon requires. Default is "pap"

user

A username that can be used to test authentication. Default is "OpenNMS"

password

A corresponding password that can be used to test authentication. Default is "OpenNMS"

secret

The shared secret with the radius daemon. Default is "secret"

timeout

The time in milliseconds to wait for a response. Default is "5000".

retries

The number of attempts made to detect the service. Default is "0".

SMB

No properties for SMB plugin.

SMTP

port

The port to connect to. Default is "25".

timeout

The time in milliseconds to wait for a response. Default is "5000".

retries

The number of attempts made to detect the service. Default is "0".

SNMP

port

The port to connect to. Default is "161".

timeout

The time in milliseconds to wait for a response. Default is null.

retries

The number of attempts made to detect the service. Default is null.

force version

The protocol version (SNMPv1 or SNMPv2) to use to check for the service. Default is null.

vbname

The OID to query. Default is ".1.3.6.1.2.1.1.2" (this is SNMPv2-MIB::sysObjectID.0).

vbvalue

The (optional) value to check for if the OID returns one. Default is null.

SSH

timeout

The time in milliseconds to wait for a response. Default is "5000".

retries

The number of attempts made to detect the service. Default is "0".

port

The port the ssh daemon is listening on. Default is "22".

match

A regular expression to check for in the response from the SSH server. Default is null

banner

If match is not defined, another regular expression to check for in the response. Default is null

TCP

port

The port to connect to. Default is null.

timeout

The time in milliseconds to wait for a response. Default is "5000".

retries

The number of attempts made to detect the service. Default is "0".

banner

Check the "banner" string against the string that is returned if the connection is successful. Default is null.

Mapping Protocol Plugins to Services

Note that the protocol plugins represent the code that is used to determine if a particular service exists. It is not the service itself. The `capsd-configuration.xml` file is where the services are actually defined.

For example, look at the HTTP service:

```
<protocol-plugin protocol="HTTP"
    class-name="org.opennms.netmgt.capsd.HttpPlugin"
    scan="on" user-defined="false">
  <property key="ports" value="80"/>
  <property key="timeout" value="3000"/>
  <property key="retry" value="2"/>
</protocol-plugin>
```

This service will use the HTTP plugin to check for a service on port 80 and create a service called "HTTP". With a simple change of port number, you can create a new service:

```
<protocol-plugin protocol="HTTP-8080"
    class-name="org.opennms.netmgt.capsd.HttpPlugin"
    scan="on" user-defined="false">
  <property key="ports" value="8080"/>
  <property key="timeout" value="3000"/>
  <property key="retry" value="2"/>
</protocol-plugin>
```

Same protocol plugin, but a completely different service as far as OpenNMS is concerned. In some cases, mainly with HTTP, you can check multiple ports. If you do this, then the service will be considered to exist if a valid response is received on *any or all* of the ports tested.

One of the more powerful plugins is the TCP plugin. Here it is used to test for the existence of secure shell:

```
<protocol-plugin protocol="SSH"
    class-name="org.opennms.netmgt.capsd.TcpPlugin"
    scan="on" user-defined="false">
  <property key="banner" value="SSH"/>
  <property key="port" value="22"/>
  <property key="timeout" value="3000"/>
  <property key="retry" value="3"/>
</protocol-plugin>
```

This will connect to port 22 and look for the string "SSH" to be returned. By using this banner check you could create different services for different version of software, such as

Oracle7 versus Oracle8, as long as the information was included in the banner (to check the banner, you can use `telnet ip-address port`). Currently, the match is strictly a substring search. In future versions regular expression may be allowed.

Server Message Block (SMB)

SMB is used by Windows servers to share files, similar to NFS. OpenNMS does not poll either SMB or NFS, but it can use some of the information provided by SMB to name nodes and group interfaces into nodes. If SMB is discovered on a device, it will be noted on the node page. You can allow OpenNMS to "log in" to an SMB share using the following tag:

```
<smb-config>
  <smb-auth user="guest" password="guest" type="domain">WORKGROUP</smb-
auth>
</smb-config>
```

Here you can enter in a valid username, password and domain for OpenNMS to use when trying to connect to an interface.

Management Policies

As mentioned in the beginning of this section, the default management policy is "managed", which means that capsd will attempt a services scan on all interfaces in newSuspect events. This can be overridden with the `ip-management` tag. From the default configuration file:

```
<ip-management policy="managed">
  <range begin="192.168.0.0" end="192.168.0.255"/>
  <include-url>file:/opt/OpenNMS/etc/include</include-url>
</ip-management>

<ip-management policy="unmanaged">
  <specific>0.0.0.0</specific>
  <range begin="127.0.0.0" end="127.255.255.255"/>
</ip-management>
```

This tag has a `policy` attribute which can be either `managed` or `unmanaged`. Then you can define ranges, specific IP addresses and files as needed. Note that the "managed" example is used specifically as an example: since the default policy is "managed" it is not needed.

SNMP

The SNMP protocol is a special case. While most of the other services will eventually be polled, the SNMP service is used to *collect* data. Let's look at its definition in the configuration file:

```
<protocol-plugin protocol="SNMP"
```

```

        class-name="org.opennms.netmgt.capsd.SnmpPlugin"
        scan="on" user-defined="false">
    <property key="force version" value="SNMPv1"/>
    <property key="timeout" value="2000"/>
    <property key="retry" value="3"/>
</protocol-plugin>

```

Note the `force version` property. Since SNMP version 2 agents will respond to SNMP version 1 requests, this test will find both agents. **This property has nothing to do with how the data will be collected.** The SNMP collector automatically checks for SNMPv2 and will use GET-BULK commands to retrieve the data (unless overridden in the `snmp-config.xml` file). But if you wanted to manage a service called "SNMPv2" you could create one with:

```

<protocol-plugin protocol="SNMPv2"
    class-name="org.opennms.netmgt.capsd.SnmpPlugin"
    scan="on" user-defined="false">
    <property key="force version" value="SNMPv2"/>
    <property key="timeout" value="2000"/>
    <property key="retry" value="3"/>
</protocol-plugin>

```

Note that the "SNMPv2" that existed in early 0.9 is no longer checked by default.

The `snmp-config.xml` File

The parameters used to connect with SNMP agents are defined in the `snmp-config.xml` file. Here is an example:

```

<snmp-config retry="3" timeout="800"
    read-community="public" write-community="private">
    <definition version="v2c">
        <specific>192.168.0.5</specific>
    </definition>

    <definition retry="4" timeout="2000">
        <range begin="192.168.1.1" end="192.168.1.254"/>
        <range begin="192.168.3.1" end="192.168.3.254"/>
    </definition>

    <definition read-community="bubba" write-community="zeke">
        <range begin="192.168.2.1" end="192.168.2.254"/>
    </definition>

    <definition port="1161">
        <specific>192.168.5.50</specific>
    </definition>
</snmp-config>

```

The attributes for the `snmp-config` tag are as follows:

`retry`

The number of attempts that will be made to connect to the SNMP agent.

timeout

The amount of time, in milliseconds, that OpenNMS will wait for a response from the agent.

read-community

The default "read" community string for SNMP queries.

write-community

The default "write" community string for SNMP queries. Note that this is for future development - OpenNMS does not perform SNMP "sets" at the moment.

All of the global parameters can be overridden with `definition` tags. These new SNMP definitions can apply to ranges or specific IP addresses. In addition, there are two other attributes available:

port

This overrides the default port of 161.

version

Here you can force either SNMP version 1 "v1" or version 2c "v2c".

capsd and SNMP

When testing SNMP, capsd makes an attempt to receive the sysObjectID for the device using the community string and port defined in `snmp-config.xml`. If this succeeds, the SNMP protocol is marked as "true" for this IP address. Note that it takes the first valid match in `snmp-config.xml` for that IP address, something to look for if the address is included in multiple ranges.

Once all of the protocols have been tested, if SNMP is true for this IP address, more tests are performed by capsd.

First, three threads are generated to collect the data from the system tree, the `ipAddrTable` and `ifTable`.

If, for some reason, the `ipAddrTable` or `ifTable` are unavailable, the process stops (but the SNMP system data may show up on the node page - this happens a lot with UC-Davis SNMP agents where only the system tree is available to a query using the "public" community string).

Second, all of the sub-target IP addresses in the `ipAddrTable` are run through the capsd capabilities scan. Note that this is regardless of how management is configured in the configuration file. This only happens on the initial scan and on forced rescans. On normal rescans (by default, every 24 hours), IP addresses that are "unmanaged" in capsd are not polled.

Third, every IP address in the ipAddrTable that supports SNMP is tested to see if it maps to a valid ifIndex in the ifTable. If this is true, the IP address is marked as a secondary SNMP interface and is a contender for becoming the primary SNMP interface.

Finally, all secondary SNMP interfaces are tested to see if they match a valid package in the collectd-configuration file. If more than one valid IP address meets all three criteria (supports SNMP, has a valid ifIndex and is included in a collection package), then the lowest IP address is marked as primary. All SNMP data collection is performed via the primary SNMP interface.

(Note: in the future we will have the ability to change to a secondary SNMP interface should the primary become unavailable).

When the capsd testing process is complete, events are generated, including NodeGainedService events.

SNMP data collection is covered in another How-To ([\[1\]](#)).