SimpleSoft

APPLICATION NOTE

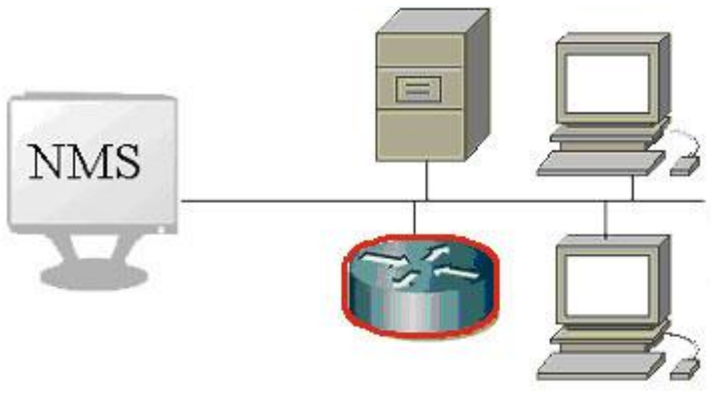
SimpleAgentPro®

Using SimpleAgentPro for Tech Support

Network Management Systems (NMS) often encounter unexpected behavior from some devices when deployed at customer premises. In order to address these technical support issues, NMS vendors need to be able to duplicate these problems in their development/test lab environment. This poses a unique challenge since access to customer networks from their development/support centers is often limited or not available. In addition, creating a lab with every available type of networking equipment is not feasible.

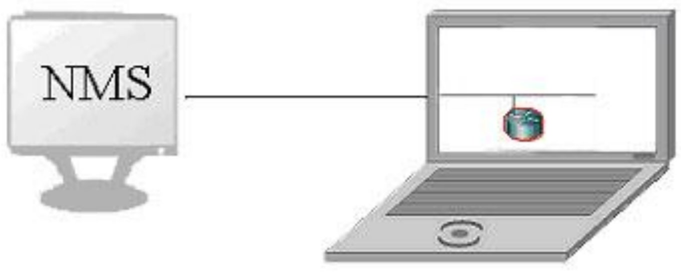
Duplicating the behavior of networking devices at customer premises involves:

- Knowing which variables are supported and which are not
- Providing similar values for queries
- Allowing for packet loss and slow responses.
- Even generating non-standard responses.



Duplicating Customer Environment

SimpleAgentPro is a network simulator that simulates the network management interfaces of thousands of networking devices on one computer. You can use SimpleAgentPro for not just scalability and functional testing but also for duplicating customer problems in your lab. A few of the approaches used in duplicating errant behavior of devices are described below.



Using MIB walks to seed SNMP simulation:

SimpleAgentPro includes small, independent, command line utilities that you can take with you to customer sites to get mib dumps from devices. These utilities are available on Windows, Linux and Solaris.

Getting mib dumps from a device involves sending a series of getnext queries to the device to “walk” the SNMP data supported by the device, and recording the variables and values returned in a file. You can use the sapwalk2 utility that is included with SimpleAgentPro installation to do the MIB walk.

Lets assume that you have a device with IP address of 192.168.10.57, with community string of “mypublic” on your customer premises that is displaying unusual behavior, and you have taken the command line utilities like sapwalk2 at your customer premises.

Step1: Getting MIB walks:

SimpleSoft’s **sapwalk2** command line utility can be used to do MIB walks on the devices and record the information in a file. To run the utility you can do:

- sapwalk2 –i 192.168.10.57 –v v2c –c mypublic –s 1.0 –o 192.168.10.57.walk

sapwalk2 will send a series of getnext requests using SNMPv2c with community string of mypublic to the device with IP address of 192.168.10.57 and start the getnext recording from a variable with an object identifier of 1.0. The values returned by the device will be saved to a file called “192.168.10.57.walk”. Although the above example shows sapwalk2 using SNMPv2c, you can use sapwalk2 with –v v1 and –v v3 to use SNMPv1 and v3 protocols.

Getting multiple values to auto-compute rate of change:

By default, sapwalk2 only gets the values once. It is like taking a snap shot of the values returned by the device. To make it poll variables of type counter and gauge multiple times so that it can compute a rate of change, you can do

- ➢ sapwalk2 –i 192.168.10.57 –v v2c –c mypublic –s 1.0 **–z 3:100** –o 192.168.10.57.walk

When sapwalk2 retrieves a variable of type Counter or Gauge, it will take 3 samples, by sending additional SNMP Get Requests for the variable 100 milliseconds apart, and record, not just one value but 3 values for the variable. Later on when this raw data is converted into a variable file, the slope will be computed by trying to determine a line that best fits between the 3 points retrieved.

Saving timing information for subsequent replay:

Sometimes, some devices return most values very quickly, but for some variables, they take a long time to return a response. By default, sapwalk2 does not save any timing information when retrieving values, but if you use the “-xl” and “-xz” arguments, you can have it add the timing information and even specify a certain threshold so that in only includes it for responses that take longer than the specified threshold information. This timing information can then be converted into a modeling file that will inject a delay for the specific variables. To use this feature, you can do

- ➢ sapwalk2 –i 192.168.10.57 –v v2c –c mypublic –s 1.0 **–xl 1 –xz 2** –o 192.168.10.57.walk

When sapwalk2 retrieves a variables, if it takes longer than 2 seconds, then it will also write out the timing information in the walk file. (-xl determines whether to write out timing information or not, and –xz specifies the threshold). Later on when this raw data is converted into a variable file, using the sapw2var utility, you can use –xl to specify a tcl script file, and –xz to specify a threshold value, and the sapw2var utility will auto create a tcl modeling file that will inject a particular delay for a particular variable.

Auto recording vlan information:

To support multiple instances of the Bridge MIB for different VLANs in the same device, some companies like Cisco, use the community string indexing/context name scheme to support it. By using the –xv bridge option, sapwalk2 can setup to make additional walks to record bridge mib information for VLANs also. When doing a walk, if the sapwalk2 utility comes across the vtpVlanTable and finds vlans like 2, 4 within it, after the main walk is complete, it starts additional walks with mypublic@2, and mypublic@4 to record the bridge mib branch only. This information can be later on coverted to %csi_begin/%csi_end sections in the var file.

To use this feature, you can do

- ➢ sapwalk2 –i 192.168.10.57 –v v2c –c mypublic –s 1.0 **–xv bridge** –o 192.168.10.57.walk

In the case of v3, the –xv bridge option will cause it to do subsequent walks with the contextnames of "vlan-2" and "vlan-4". You do not need to do separate walks to record the vlan bridge mib data supported.

Using Other SNMP Walk Utilities:

You can also use SNMP walk utilities from Net-SNMP, HP, CA/Concord, Microsoft and others, instead of sapwalk2 if they are more easily available at customer premises. Their usage instructions will assist you in determining how to run them.

When using Net-SNMP's snmpwalk utility, we recommend specifying the following options to generate the easiest output to parse.

```
 -ObenU
        b tells it output the instance component in raw mode.
        e tells it to output enumerated values as numbers
        n tells it to output oids as numbers
        U tells it to skip the units suffix for values
```

More info on the snmpcmd can be got at:
http://www.net-snmp.org/docs/man/snmpcmd.html

Step2: Converting walk files to var:

Once the walk file is created at customer sites, you can have it transferred to your development and test lab area via ftp or as an email attachment. The walk files are simple ascii text files.

SimpleSoft's **sapw2var** command line utility can then be used to convert walk files to var files. If you have the mibs supported by the device, you can compile them and create a compiled mib file. If they are not available, you can simply use mib2.cmf which comes with the installation when converting the walk file to a var file. The converter simply treats variables not present in the cmf file as ReadOnly variables and still supports them. To convert the walk file generated in Step 1, you can do:

- ➢ sapw2var –w 192.168.10.57.walk –c ..\cmf\mib2.cmf –v mydev.var –t ssoft

sapw2var will convert the walk file "192.168.10.57.walk" with compiled mib information from "../cmf/mib2.cmf" into a new var file called "mydev.var" and tell sapw2var that the input file was created by using ssoft tools.   If the walk file was created by using Net-SNMP snmpwalk, then you would use the "-t net" argument, and "-t hp" for HP walk utilities.

The var file created in Step 2 now contains all the variables supported by the device found on you customer premises. You can now create a map, and a device to the map and in its SNMP properties specify "mib2.cmf" and "mydev.var" as its compiled mib file and variable file. Additional information on defining maps and adding devices to a map is available in Chapters 4 and 7 of the manual.

You can also add a new device to your copy of the device library to use it in the future.

Using a Poller to collect SNMP data to return similar values:

Another option to get multiple values is to use the sappoll utility. If you know the list of variables polled by your application, you can specify the list of variables to retrieve, and the number of samples to take, and the sappoll utility will record the data. SimpleAgentPro can then replay that data returning the same values back when polled.

To use the utility, you need to first do the walk as described in Step1 and convert the file to a var file as described in Step 2. After creating a file containing a list of variables polled by your application, (an example file is shown below containing instance (I) and nodes(N)) you can run the following two commands:

```
---------------------start of  oidlist.txt------------------------
# scalar ifNumber with instance
I 1.3.6.1.2.1.2.1.0
# node ifTable
N 1.3.6.1.2.1.2.2
# scalar sysDescr with instance
I 1.3.6.1.2.1.1.1.0
# node tcpConnectionTable
N 1.3.6.1.2.1.6.13
# scalar icmpInMsgs with instance
I 1.3.6.1.2.1.5.1
.---------------------end of oidlist.txt-------------------------
```

- sappoll –i 192.168.10.57 –v v2c –c mypublic –f oidlist.txt –t 5 –n 10 –o polldata

sappoll will poll the device with IP address 192.168.10.57 using SNMPv2 and community string of "mypublic" for the variables specified in oidlist.txt, and it will do so 10 times, every 5 seconds and save the information gathered in a file called polldata.

The polldata file will contain the values for the various variables that looks like this

```
#I=1.3.6.1.2.1.2.2.1.11.1
39236, 39239, 39243, 39246, 39249, 39252, 39255, 39258, 39261, 39265
```

You can then combine the information in the polldata file, with the variable file created in Step 2 by using the **sapvfilter** command line utility.

- sapvfilter   mydev.var   mydev2.var polldata ..\cmf\mib2.cmf 3

and it will look at the original file “mydev.var”, and wherever it sees variables that are present in the “polldata” file, it will modify the valuetypes for those variables to refer to the polldata file and create a new var file called “mydev2.var” . It will also use the DELTA (3) scheme to return values for subsequent polls after the recorded values are retrieved.

Since the polldata file has information about ifInUcastPtks.1 (1.3.6.1.2.1.2.2.1.11.1), as shown earlier, if the original “mydev.var” file had

```
ifInUcastPkts.1             , Counter     , RO , randomup(39295, 100)
```

the new “mydev1.var” will replace it with

```
ifInOctets.1                , Counter     , RO , valuelistinonefile(polldata, DELTA)
```

When you create a simulated device and associate “mydev2.var” with it, and it is polled for ifInUcastPtk.1, it will first return 39236, return 39239 the second time it is polled, return 39243 the third time it is polled and so on. On the 11th poll, since it is using the DELTA style of extrapolation, it will add the difference between the 1st and 2nd values (39239 – 39236 = 3) to the last value of 39265 and return 39268.

Associating SNMP error behavior with the simulated device:

SimpleAgentPro includes a Tcl interpretor inside each of the simulated devices. By evaluating Tcl commands inside the simulated device context, you can dynamically control the variables supported, their values, and even the device behavior. You can associate a Tcl based modeling files with the simulated device, in which you can specify the Tcl commands to be evaluated as part of incoming SNMP request processing or even as part of timer based actions. Some examples of error behavior are given below. Other examples can be found in the FAQ section of the manual.

Simulating slow responses:

Some devices are slow in responding to SNMP queries causing the NMS to exercise its timeout/retransmission logic. You can associate a SNMP delay in milliseconds when defining the device. This will cause the simulated device to wait for the specified delay before responding to a SNMP request. From Version 16.0 and later, support has also been added to randomly delay the response within a specified range making it easier to even simulate changing network traffic/load conditions that delay some responses but not others.

You can specify the value you want in the SNMP delay field in the SNMP properties of the device, or use the following Tcl commands.

```
#delay each response by a fixed 500 milliseconds
SA_snmpdelay 500
```

or

```
#delay each response by a random number between 10 and 1000 milliseconds
SA_snmpdelay 10,1000
```

If you want to simulate a situation where the agent response for a few specific variables is slow, as mentioned earlier, you can consider using the –xl and –xz options when using sapwalk2 and the –xl and –xz options with sapw2var when converting the walk file to a var file along with a tcl modeling file.

Simulating dropped packets:

Sometimes a device will respond to most packets, but every so often, it will drop some packets. To make the simulated device exhibit this type of behavior, you can associate the following modeling file with the device. In the example shown below, the simulated device will drop every 5th packet.

```
---------------------start of  drop_packet.tcl------------------------
%init_action
   #
   # set these variables if you want to specify the pattern
   # of failed/(total = fail+successful) responses.
   #  eg: 1 in 5, myPatternFail = 1, myPatternTotal = 5
   #  eg: 3 in 7, myPatternFail = 3, myPatternTotal = 7
   #
   set myPatternFlag     1
   set myPatternFail     1
   set myPatternTotal    5
   set myPatternCounter  0


#
# this script segment should create the pattern of
# fail/(total = fail+successful) response
#
%before_snmp_request
   if { $myPatternFlag == 1 } {
        incr myPatternCounter
        if { $myPatternCounter <= $myPatternFail } {
            #
            # setting the error to a special value of
            # 9999 will cause this request to be dropped.
            #
            SA_seterrorstatus 9999
        }
        if { $myPatternCounter >= $myPatternTotal } {
            set myPatternCounter 0
        }
    }.
---------------------end of drop_packet.tcl --------------------------
```

Simulating SNMP Errors in responses:

Sometimes an errant device might return unexpected SNMP errors or incorrect response packet generation.   To make the simulated device exhibit this type of error behavior, you can associate the following modeling file with the device.   Various examples are covered to give you a flavor of the type of things that can be done. You can edit the file to suit the type of errors you want to reproduce.

```
---------------------start of  snmp_err.tcl------------------------
%init_action
    #
    # example where each request will generate multiple
    # responses (this case 3). To reset it back, set it to 1.
    #
    SA_setnumresponses 3


#
# example where there is no response sent
# when a get is made on sysDescr.0
#
%getvalue_action sysDescr.0
    SA_seterrorstatus 9999

#
# example where there is an error sent when a get is
# made on sysObjectID.0.  Some v1 errors are
#       tooBig     (1)
#       noSuchName (2)
#       badValue   (3)
#       readOnly   (4)
#       genError   (5)
#
%getvalue_action sysObjectID.0
    SA_seterrorstatus 5

#
# example where a particular varbind is skipped in
# the response.  This one skips sysUptime.0
#
%getvalue_action sysUpTime.0
   SA_setskipcurvbflag 1

#
# example where the wrong datatype is returned.
# This one changes datatype component
# when a get is made on sysName.0
#
%getvalue_action sysName.0
   SA_setcurtype Integer

#
# example where the wrong oid is returned.
# This one changes oid component
# when a get is made on sysLocation.0
#
%getvalue_action sysLocation.0
   SA_setcuroid 1.3.6.1.2.1.1.2.0

#
# example where the request id in the response is
# made different from the request id
#
%getvalue_action sysServices.0
   SA_setreqid 5432
---------------------end of snmp_err.tcl --------------------------
```

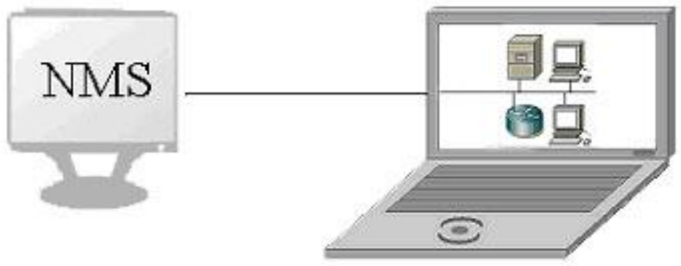
Simulating Two stage SNMPv3 Discovery:

In the case of SNMPv3 discovery, SNMP agents typically respond with a Report PDU that contains the correct engine_id and the boot time and boot counts. Sometimes, due to the ambiguity in the RFC, agents might respond in two steps, by first responding with a Report PDU with just the engineid, and then responding with a boot time and count in a following notInTimeWindow Report PDU. To make the simulated device exhibit this behavior, you can use the following Tcl command and have it executed in the %init_action section of a modeling file.

```
SA_setv3twostepdiscoveryflag 1
```

Simulating Multiple SNMP Devices:

SimpleAgentPro was used in the above examples to simulate a single errant device. You can also use SimpleAgentPro to simulate a group of devices and duplicate the customer network topology in your lab environment. This is particularly useful when duplicating topology related issues.



In the example below we will show the steps to take to duplicate multiple devices, not just one.

Step1: Making a list of devices to learn:

To learn from multiple devices, you need to generate a file that contains the list of IP addresses of the devices to learn from along with their SNMP profiles. If the list is small you can manually enter the list of devices along with their SNMP profiles, or use the ssdiscover utility. A sample file is shown below.

```
---------------------start of devlist.txt------------------------
192.168.10.57  v2c mypublic
192.168.10.5  v1 public
192.168.10.17 v2c private
192.168.10.88 v1 public
---------------------end of devlist.txt--------------------------
```

Step2: Do SNMP Walks on the devices and get MIB dumps:

Now that you have a list of devices to learn from, you need to do snmp walks on them and get their MIB dumps. You can use the sapnwalk utility which calls the underlying sapwalk2 utility to learn from a list of devices in parallel.

Using the devlist.txt file created in Step 1, and assuming that you want the walk files to be created in /tmp/mywalks directory, you can run the following command

- ➢ sapnwalk –f devlist.txt –w /tmp/mywalks

sapnwalk will start up to 10 threads in parallel and invoke the sapwalk2 utility to do the walks on the devices specified in the devlist.txt file. After sapnwalk is completed, the /tmp/mywalks directory will have the following files:

```
192.168.10.57.walk
192.168.10.5.walk
192.168.10.17.walk
192.168.10.88.walk
```

The files contain the output of the MIB walks done on the devices.   If these utilities are run at the customer site, then simply bring these files along with the devlist.txt file over to your development/lab area to run the remainder of the steps.

Step3: Convert Walk files to Var files and create a map.

Now that you have your walk files, you need to convert them into var files that are used by SimpleAgentPro for device simulation. You also need to create a map which is a collection of devices with their properties that can be started in the simulator.

The sapnw2var utility does both. It first converts the walk files to var files and then also creates map files for each of the devices in the device list.

The same devlist.txt that we had created in Step 1, can be used here also. If we use a project name of "test", we can say

- sapnw2var –f devlist.txt –w /tmp/mywalks –p test –b tmp

where test is the name of the project, tmp is the base map name, and "192.168.10…" are the values for the three tokens used in the myrules.rul.

After sapnw2var has completed, if you go to the <install_dir>/sapro/projects directory, you should see a "test" directory. Inside this test directory, would be two subdirectories: map and var.

The var subdirectory will contain the converted var files:

```
192.168.10.57.var
192.168.10.5.var
192.168.10.17.var
192.168.10.88.var
```

The map subdirectory will contain one map file:

```
tmp_1.map
```

When you start the map, it will create 4 devices with similar IP addresses as the original IPs at the customer premises. So please make sure you are running in an isolated network.

Using WireShark packet captures to seed SNMP simulation:

SimpleAgentPro also includes a utility that will create a SNMP variable file from a pcap file that contains captured traffic. So if you are able to get the customer to run wireshark or tcpdump and capture the traffic between the NMS and the errant device on their network, you can have the pcap file sent to your development and support departments, and they can create a simulated device that will have SNMP data which is based on the responses in the captured traffic.

This is particularly useful when the number of variables supported by a device are very large and it might take many hours, if not days to run a SNMP walk to get a MIB dump from that device. In such cases, you can use wire-shark with a filter to capture the traffic between an NMS and a particular device and use it for recreating problems. When the problem is intermittent, you can even run wireshark for a few days with the filter and capture the packet exchanges between the NMS and the errant device.

The captured traffic will typically have three cases:
- Request for a particular variable sent only once and its value returned
- Request for a particular variable sent multiple times and a same value returned
- Request for a particular variable sent multiple times and different value returned.

Based on the settings in the rules file, these additional values can also be used in the simulation.

Lets assume that you have a device with IP address of 192.168.10.57, with community string of “mypublic” on your customer premises that is displaying unusual behavior, and you have captured the traffic between the NMS and that device in a pcap file called “traffic.pcap”

To simply use the first value seen for a particular variable, you can use the sapp2var utility as follows.

➢ sapp2var -i traffic.pcap –o mydev.var –c ../cmf/mib2.cmf

and sapp2var will look for SNMP packets in “traffic.pcap”, and for the first SNMP device responding to port 161, it will create a new var file “mydev.var” using mib information for “mib2.cmf”.

You can explicitly specify the IP of the agent (if the pcap file contains SNMP packet interaction from multiple devices) by using the –a option.

➢ sapp2var -i traffic.pcap –o mydev.var –c ../cmf/mib2.cmf –a 192.168.10.57

and sapp2var will look for SNMP packets in “traffic.pcap” from 192.168.10.57, to create a new var file “mydev.var” using mib information for “mib2.cmf”.

To use the multiple values for the same variable, you can use the –v option.

- ➢ sapp2var -i traffic.pcap –o mydev.var –c ../cmf/mib2.cmf –a 192.168.10.57 –v poll.txt

and sapp2var will look for SNMP packets in “traffic.pcap” from 192.168.10.57, to create a new var file “mydev.var” using mib information for “mib2.cmf”. If it finds variables that have been polled multiple times, it will save the values in a file called “poll.txt” and refer to it in the var file (by using valuelistinfile() valuetype)

You can also use a rules file to specify additional rules for handing multiple values. An example rules file is shown below.

```
---------------------start of  myrules.txt------------------------
ifSpeed                                    fixed(%s)
1.3.6.1.4.1.9.9.166                        fixed(%s)
%ONEVALINPCAP_IPADDRESS    fixed(%s)
%ONEVALINPCAP_INTEGER      random(%s, 111111)
.---------------------end of myrules.txt--------------------------
```

You can run the sapp2var utility as follows.

- ➢ sapp2var -i traffic.pcap –o mydev.var –c ../cmf/mib2.cmf –r myrules.txt -v poll.txt –a 192.168.10.57 –d 1

sapp2var will look for SNMP packets from the agent with IP of 192.168.10.57 in the traffic.pcap file, and create a new var file “mydev.var” using information in the “myrules.txt” file. If it finds variables that have been polled multiple times, it will save the values in a file called “poll.txt” and refer to it in the var file.

As before, you can now create a map, and add a device to the map and the devices’s SNMP properties, specify “mib2.cmf” and “mydev.var” as its compiled mib file and variable file, to simulate the device. In case the “poll.txt” file is also created, you should move it to the same directory as the “mydev.var” file.

Usage information for the sapp2var utility is given below:

```
#sapp2var: ver 2.4
#Copyright (c) 1994-2015 SIMPLESOFT Inc.
Usage: sapp2var
        -i   input pcap file with captured packets
        -o  output var file to create
        -c  compiled mib file for mib info
        -r  (optional) rules file for conversion(default=none)
        -v  (optional) csv value file for poll data(default=first val)
        -a  (optional) Agent IP(default=firstIP on snmpport)
```

```
-m  (optional) Manager IP(default=any)
-p  (optional) Agent SNMP Port(default=161)
-d  (optional) debug info in log file(2/1/0)(default=0)
-l  (optional) log file name(default=sapp2var.log)
-b  (optional) base community for vlan support(default=none)
-x  (optional) output list of oids instead (0/1)(default = 0)
-tf (optional) tcl script file name(default=none)
-tt (optional) timestamp threshold (default=1sec)
```

Learning Telnet Responses from a Device to seed Telnet Simulation:

Many a times, the NMS gets some information from the device using SNMP, while it gets other types of information from the device using Telnet/SSH that use the device's command line interface (CLI). SimpleAgentPro device simulation includes support for Telnet/SSH in addition to SNMP.

Just like you learnt SNMP data from the device using sapwalk2 utility, SimpleAgentPro includes a telnet learner utility "tellearn". Tellearn acts as a telnet client, sends a sequence of commands based on the contents of a command file to the device, receives responses from the device, and records them in a telnet modeling file that can be subsequently used to seed the simulation. Unlike the SNMP walk utility, which due to the GetNext feature in SNMP gets to learn the entire MIB tree supported by the device, the telnet learner utility needs to be given an explicit list of commands that are to be learnt.

Lets assume that you have a device with IP address of 192.168.10.57, with username "admin", password "abc", and enable password "pqr" that supports telnet and you want to learn responses for some commands from it.

You can create a command list file called "clist.txt" containing

```
---------------------start of clist.txt------------------------
show version
enable
terminal length 0
show running-config
---------------------end of clist.txt--------------------------
```

You can run the tellearn utility as follows.

- tellearn -i 192.168.10.57 –u admin –p abc –e pqr –c clist.txt –o mydev.tel

and the tellearn utility will set up a telnet connection with 192.168.10.57 and when prompted for username, will specify “admin”, when prompted for password, will specify “abc” and then upon getting the command prompt, will start sending commands, one after the other, from the clist.txt file. In case of enable, it provide the password “pqr”. The responses will be converted and recorded in the telnet modeling file created called “mydev.tel”. It will contain “%cmd_action” sections for each of the commands in the clist.txt along with their corresponding responses.

The command list file (clist.txt) is a simple text file that contains a list of commands whose responses are to be learnt. Each command is assumed to be present on a separate line. It also supports some keywords to handle command completion logic as well as command response parsing.

For example, to learn all show options, you can add

```
show %CMDCOMP%
```

and it will send “show ?”, parse the response and extracts the first column which is not <cr>, and then send “show version”, “show terminal”, “show history”, “show cdp” and learn their responses.

To recursively learn show options, you can add

```
show %RCMDCOMP%
```

and it will send “show ?”, parse the response and then send “show version ?” and check if it has any additional options and if it does not then, it will simply learn “show version” response. In the case of “show cdp ?” it will learn “show cdp”, “show cdp neighbors”, “show udp neighbors detail” also..

To save the options returned in a tcl variable called cmdcomplist, you can do

```
show %TCMDCOMP%
%tclscript  set commandlist [lindex $commandlist [expr $curcmdindex + 1] [lindex $cmdcomlist 0]
```

and the tcl variable “cmdcomplist” will be set to the output of “show ?”. You can then parse that variable and build a new string, which will insert the first option returned in the list of commands to learn after the current command.

To learn the output of a command you can say

```
show version %TCMDRESP%
```

and the entire response will be set in a tcl variable called “cmdresplist”

The tcl variables used internally are set to the following values. You can overwrite these values by creating a tcl file and specifiying it with the “-t” option.

```
set telnetport  23
set cmdprompt1char ">"
set cmdprompt2char "#"
set timeout 10
set getonlynewline 1
set skipblankline 1
set morerespstr " "
set morestringlist "--More--"
```

Usage information for the tellearn utility is given below:

```
#SAPro Telnet Learner: Ver 2.0
#Copyright 1994-2015 SimpleSoft Inc.
#Usage: tellearn <options>
#      -i ipaddress           Device IP Address
#      -o output file         Output Telnet Modeling file
#      -u useranme            Username
#      -p password            Password
#      -e enable password     Enable Password
#      -c cmdlist             Input Telnet Command list
#      -t tclscript           Optional tcl script to override defaults
#      -d debugfile           Optional debug output file
```

Using WireShark packet captures to seed Telnet simulation:

Just like you learnt SNMP variable data from a pcap file using “sapp2tel” utility, there is ia “sapt2tel” utility to learn telnet command/responses from a captured pcap file to create the telnet modeling file that can be used to seed the simulation.

Lets assume that you have a device with IP address of 192.168.10.57, and the telnet communication between the NMS and the device has been captured in a pcap file called “traffic.pcap” as before.

You can run the sapt2tel utility as follows.

- sapt2tel -i traffic.pcap –o mydev.tel -t pcap

and the sapt2tel utility will look at each of the packets in the pcap file and the first time it finds packets that are being sent to port 23 (telnet port), it will transcribing the communication into the “mydev.tel” telnet modeling file.

If there are multiple device communication captured in “traffic.pcap”, you can specify a device IP to use for learning as follows:

- sapt2tel -i traffic.pcap –o mydev.tel -t pcap –d 192.168.10.57

and the telnet utility will only look at telnet communications with 192.168.10.57 to transcribe.

Usage information for the sapt2tel utility is given below:

```
#SAPro Telnet Text-to-ModelingFile Converter: Ver 1.3
#Copyright 1994-2015 SimpleSoft Inc.
#Usage: sapt2tel <options>
#      -i input               Input telnet log file
#      -o output              Output telnet modelling file
#      -l login prompt        Login prompt term chars (:)
#      -c command prompt      Command prompt term chars(>,#)
#      -a addlogin prompt     AddLogin prompt term chars (:)
#      -m more string         More indicator string (---More---)
#      -u username            UserName to use
#      -p password            Password to use (abcd)
#      -e addlogin command    Addlogin command string (enable
#      -x addlogin passwd     Additional Login password (bigfish)
#      -t input_type          text(log)/wire(text)/pcap/pdbg(proxydbg)(default = pcap)
#      -s telclientip         Telnet client IP address
#      -d telserverip         Telnet Server IP address
#      -n telserverport       Telnet Server Port address (default=23)
#      -g debugflag           Debug output level(default=0)
#      -q start_state         0/1/2/3(default=0)
#      -f justlf              0/1(default=1)
#      -z dumpmsg             0/1(default=0)
#      -y directory           0/1(default=none)
```