iReasoning SNMP Agent Simulator User Guide

Table of Contents

Chapter 1

Installation

Requirements

- Windows, Linux, and UNIX operating systems
- At least 64 MB memory is required

Installation Procedures

1. **Installing simulator**

   On windows, run setup.exe to start simulator installation program. On Linux and UNIX, unzip *simulator.zip* (SUN's Java 5.0 or a later version needs to be installed first).

   The directory structure will look like the following:

Directory Name	Description
bin	Contains windows script files
devices	Device library data.
doc	Contains PDF and HTML formats of user guide
jre	Java Runtime Environment
mib	MIB files
lib	Contains binary jar files
log	Stores log file
config	Stores simulator's configuration file
examples	Cisco router and Windows 2000 SNMP agent simulators

2. Start simulator

On windows, you can double click on the simulator icon on your desktop or start menu, or run bin\runUI.bat scrip to start simulator from command line.

On other platforms, run bin/runUI.sh to start simulator.

After startup, you can create a new project or open an existing project. It is a little slow when loading project for the first time, or after data file is modified. It is because the content of XML format data files needs to be converted and inserted into database. After data has been stored in database, then it will be much faster to open a project.

Chapter 2

Using SNMP Agent Simulator

Overview

SNMP Agent Simulator is a Java based application that can simulate SNMPv1/v2c/v3 agents. Since it is written in Java, it can run on all the platforms that have Java Virtual Machine installed, including UNIX, Linux, Windows, etc. It allows you to develop, test and train SNMP management applications without purchasing and maintaining expensive hardware devices.

Typical applications of SNMP simulator include:

- Developers and testers of management applications can do their jobs without real SNMP agents. SNMP simulators can act like real SNMP agents and are more flexible than them because data file of simulators can be easily modified to simulate different cases.
- Sales and training staff can give live demonstrations in laptop computers without the needs of carrying heavy network devices. Complex networks can be easily simulated in one laptop.

Major Features

- Ease of use
- Record and replay SNMP devices

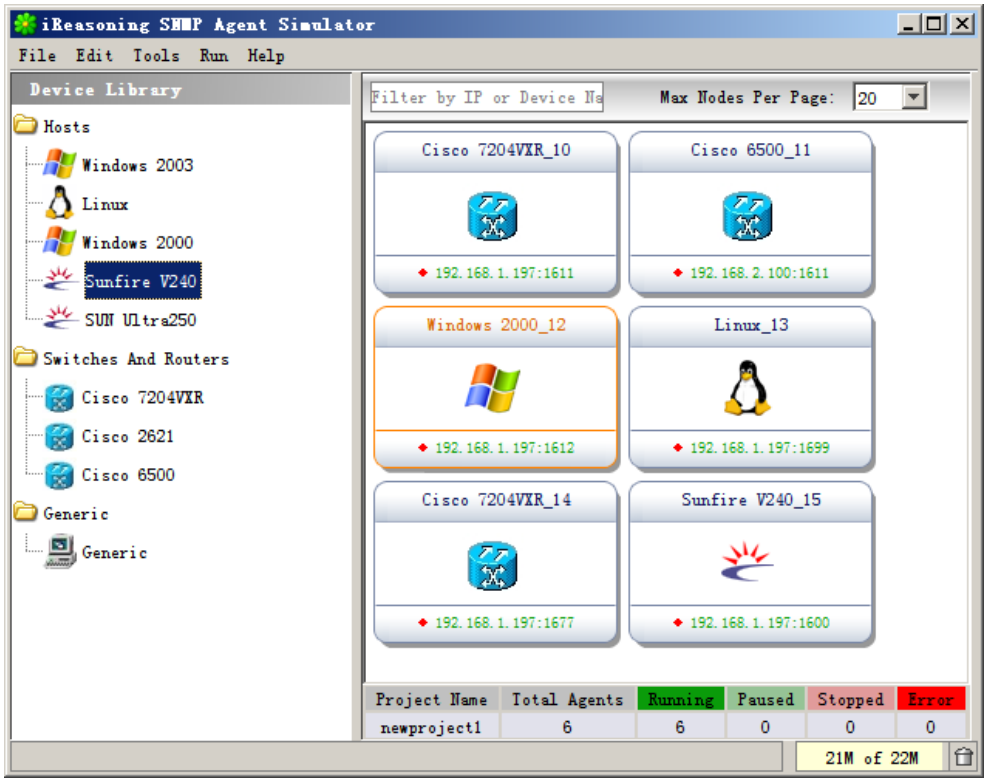
  Values of SNMP agent MIB nodes can be recorded into XML file through GUI or command line interface. Recorded data file can be modified to simulate different configurations.
- Multiple agents in one JVM
- Support for dynamic row creation and deletion
- Support for BeanShell scripting language that is used in data file to model real time SNMP agent behavior
- Trap simulation

  Simulator can generate request-based, threshold-based, and timer-based SNMPv1/v2/v3 traps and informs. And it can simulate trap storm as well.
- Error simulation
- Scalability

  The number of agents on one computer is dependent on available system resources.
- Community/Context indexing (Different data for different community or SNMPv3 context)

Usage of Graphical User Interface

- Main window



Main window has two panes. The left pane is the device library that contains pre-defined devices, which can be dragged to the right pane. The right pane shows simulators and their status. Right click on a simulator and choose a context menu item to change the status or settings.

Generate Data from MIBs

SNMP walk data can be generated from MIB files as well.

Click on "Tools/Generate Data from MIBs" menu. Multiple MIBs can be entered and they are separated by comma.

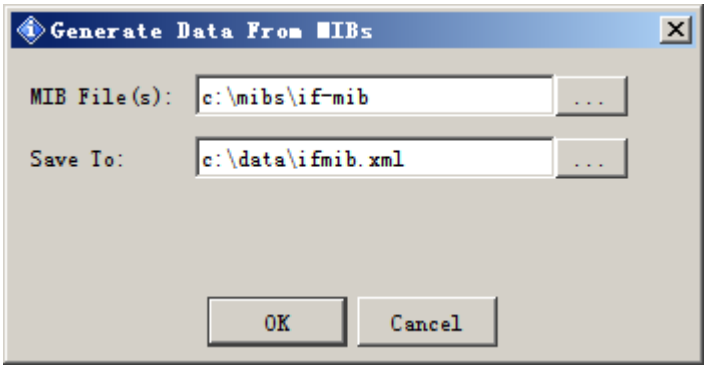


Figure: Generate Data from MIBs

In this example, SNMP walk data is generated at "c:\simulators\data\mibdata" based on MIB-II and host mib.

To add newly generated data to a simulator, right mouse click on a simulator and select "Properties" menu item, and change the data file property.

Recording SNMP Traps

SNMP traps can be recorded and added to a simulator. When simulator starts, recorded traps will be sent out to all trap sinks defined in simulator's configuration file at specific time.

Click on "Tools/Record SNMP Trap" menu.

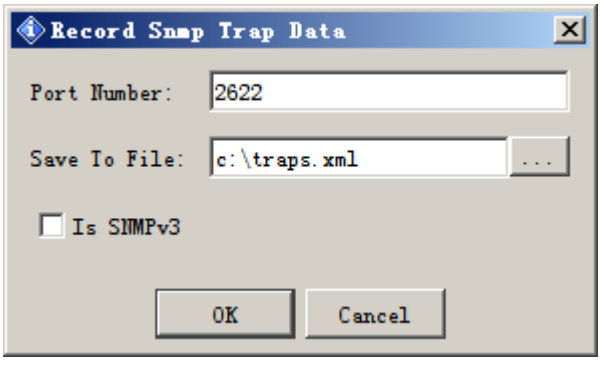


Figure: Record SNMP Trap

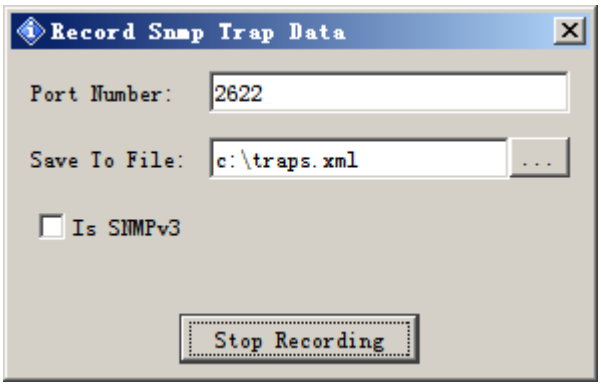


Figure: Recording traps...

Click on "Stop Recording" button when it is time to finish recording. The trap data is stored in "c:\traps.xml" in this example.

To add trap data to a simulator, right click on a simulator and select "Properties", and change the trap file property.

Community/Context Indexing

An SNMP agent has two data files, and each of them has different community or context (for SNMPv3 agent). The community/context is used to determine which data file should be returned for a SNMP request.

For instance, in the bundled context sample agent (in examples/context directory), two data files are specified in the project file:

<dataFile name="data/linux1.xml"/>
<dataFile name="data/linux2.xml"/>

And in the beginning of linux1.xml:

*<Instances readCommunity="**public1**" >*

In the beginning of linux2.xml:

*<Instances readCommunity="**public2**" >*

The community/context of the first data file is *public1*, and *public2* for the second data file.

If the agent is SNMPv1/v2c agent, community names are used. If it is an SNMPv3 agent, the value of readCommunity becomes the context name.

SNMPv3 parameters are

Field	Value
User Name	newUser
Auth Algorithm	MD5
Auth Password	abc12345
Priv Algorithm	DES
Priv Password	abc1345
Context	public1 or public2

Note that the context settings in the config file (SnmpSimulator.xml):

contextPrefix="pub"
match="prefix"

The default settings were match="exact". Because both of context names start with pub, so it is changed to prefix match.

If you use SNMP managers to check agent's values, you will notice that different data is returned for different communities/context names.

Usage of Command Line Interface

Enter ireasoning/simulator/bin directory.

- runSimulator.bat/runSimulator.sh

  Usage: runSimulator projectFile

  This script executes a project file, which is also used in GUI. A project files defines properties of simulators, including data file, initial states, etc. If the initial state of a simulator is "up", it will be started when the script runs.

- runSnmpTrapRecorder.bat/runSnmpTrapRecorder.sh

  Usage: runSnmpTrapRecorder [options...]

  Where possible options include:
  -? print this Usage
  -h print this Usage
  -f output file name
  -p <p> port number, 162 by default.
  -m <m> mib files to load. Use ',' or ';' to separate multiple files

  Example:
  runSnmpTrapRecorder -f output.xml
  runSnmpTrapRecorder -f output.xml -p 1620

  This script records SNMP traps and save them to an XML file. It can be terminated by hitting CTRL+C.

- runSnmpWalkRecorder.bat/runSnmpWalkRecorder.sh

Usage: runSnmpWalkRecorderCmd [options...] <hostname> [<startingOID> <endingOID>]

<startingOID> starting object identifiers, .1.3 by default. Or MIB node name if MIB loaded
<endingOID> ending object identifiers, .1.3 by default. Or MIB node name if MIB loaded

where possible options include:
-? print this Usage
-h print this Usage
-f output file name
-i interval of walks, in seconds
-t times of walks
-c <c> community name
-v <1|2|3> specifies SNMP version to use, default is 1
-p <p> port number, 161 by default.
-m <m> mib files to load. Use ',' or ';' to separate multiple files

SnmpV3 options:
-u <u> user name
-a <a> authentication algorithm, one of md5 or sha. md5 by default.
-A <A> authentication password
-x <x> privacy algorithm, either "DES" or "AES". DES by default.
-X <X> privacy password

Example:
runSnmpWalkRecorderCmd -f output.xml localhost .1.3
runSnmpWalkRecorderCmd localhost -f output.xml -v 3 -u newUser -A abc12345 -X abc12345 .1.3
runSnmpWalkRecorderCmd -f output.xml localhost .1.3.6.1.2.1 .1.3.6.1.5.1
runSnmpWalkRecorderCmd -i 30 -t 2 -f output.xml localhost .1.3.6.1.2.1 .1.3.6.1.5.1

This script records SNMP walk data and save it to an XML file. It can be terminated by hitting CTRL+C.

- netsnmp.bat

Convert Net-SNMP SNMP walk output to data file.

Helper Functions

Helper functions can be used in the scripts in the SNMP walk data file.

- **long getSysUpTime()**
  This function returns the simulator up time, in 1/100 second.

  Usage example:
  time = getSysUpTime();

- **exec(String command)**
  This function executes an external command.

  Usage example:
  exec("command1.exe");

- **int getSubtreeInstanceCount(String subTreeOID)**
  This function returns the number of MIB instances falling under the subtree.

  Parameters:
  subTreeOID: the OID of a MIB tree node

  Usage example:
  //gets the total number of MIB instances under subtree
  //".1.3.6.1.2.1.2.1"
  count = getSubtreeInstanceCount(".1.3.6.1.2.1.2.1");

- **void sleep(long milliseconds)**
  This function causes current simulator to sleep (temporarily cease execution) for the specified number of milliseconds.

  Parameters:
  milliseconds: time to sleep, in milliseconds

  Usage example:
  sleep(1000);//sleep for 1 second

- **VarBind createVarBind(String oid, String type, String value)**
  This function creates a SNMP variable binding object.

  Parameters:

oid:	object identifier of the variable binding
value:	value of this variable binding, in string format
type:	data type of the value.
	The possible types are (case insensitive) :
OctetString:	octet string
Null:	null
OID:	object identifier
BITS:	SNMP BITS
UnsignedInteger:	unsigned integer
Counter32:	32-bit counter
Gauge:	32-bit gauge
TimeTicks:	time ticks
IpAddress:	ip address
Opaque:	opaque
Integer:	integer
Counter64:	64-bit counter

  Usage example:

```
// Creates a variable binding whose OID is ".1.3.6.1.2.1.1.1.5"
// and its values is an integer (5)
VarBind varbind1 = createVarBind(".1.3.6.1.2.1.1.1.5", "integer", "5");
```

- **void sendV1Trap(int generic, int specific, VarBind varbinds)**
  This function sends out SNMPv1 trap.

  Parameters:

generic:	value of generic field
specific:	value of specific field
varbinds:	variable bindings

  Usage example:

```
sendV1Trap(6, 110, null); //null means no variable bindings for this trap

//A trap with two variable bindings
sendV1Trap(1, 0, createVarBind(".1.3.6.1.2.1.1.1.5", "integer",
   "5").createVarBind(".1.3.6.1.2.1.1.1.5", "timeticks", "51211"));
```

- **void sendV2Trap(String snmpTrapOid, VarBind varbinds)**
  This function sends out SNMPv2 trap

  Parameters:
  snmpTrapOid: OID value of snmpTrapOid object.
  varbinds: variable bindings

  Usage example:
  ```
  //create a trap with only one variable binding
  sendV2Trap(".1.3.6.1.4.1155.1", createVarBind(".1.3.6.1.2.1.1.1.5",
                  "integer", "5"));

  //create a trap with two variable bindings
  sendV2Trap(".1.3.6.1.4.1155.1", createVarBind(".1.3.6.1.2.1.1.1.5",
    "integer", "5").createVarBind(".1.3.6.1.2.1.1.1.5", "timeticks",
    "51211"));
  ```

- **void sendTrap(String trapName)**
  This function sends out SNMPv1/v2 trap to all trap destinations defined in simulator's config file.

  Parameters:
  trapName: name of a trap node in data xml file

  Usage example:
  ```
  sendTrap("trap1");
  ```

- **int atoi(String stringValue)**
  This function converts a string into integer.

  Parameters:
  stringValue: a string to be converted to integer.

  Usage example:
  ```
  int i = atoi("511");
  ```

- **String itoa(int i)**
  This function converts an integer into string.

  Parameters:
  i: an integer to be converted to string.

  Usage example:
  ```
  String s = itoa(511);
  ```

- **int random(int uppperbound)**

  This function returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the uppperbound (exclusive), drawn from this random number generator's sequence.

  Parameters:

  uppperbound: the upper bound on the random number to be returned. Must be positive.

  Usage example:

  ```
  i = 1000 + random(500);
  ```

- **int random(int lowerbound, int uppperbound)**

  This function returns a pseudorandom, uniformly distributed int value between lowerbound (inclusive) and the uppperbound (exclusive), drawn from this random number generator's sequence.

  Parameters:

  lowerbound: the lower bound on the random number to be returned. Must be positive.

  uppperbound: the upper bound on the random number to be returned. Must be positive.

  Usage example:

  ```
  i = 1000 + random(100, 500);
  ```

- **previousValue(initialValue)**

  This function returns the previous value of a MIB instance, that is, the value returned for the previous SNMP request.

  Parameters:

  initialValue : the initial value of this MIB instance

  Usage example:

  ```
  Object getValue(String oid, SnmpPdu pdu)
  {
    int ret = previousValue(50) + 10;
    //for this MIB instance, its first value is 50,
    //the second value is 50 + 10 = 60,
    //the third value is 60 + 10 = 70. ...
     return "" + ret;
  }
  ```

SNMP Walk Data File

This file stores data of one or multiple SNMP walks against one SNMP agent.

One simulator can have one or multiple SNMP walk data files. However, multiple walk data files must have different community names (community-string-indexing), otherwise only the first data file will be used.

▪ Root Node

<SnmpSimulatorData
sysUpTimeOffset="11111111"
delayRange="50-200"
>

sysUpTimeOffset:	The offset of sysUpTime, in milliseconds. The returned value of getSysUpTime() function will be shifted by this offset.
delayRange:	The delay time range for every SNMP response, in milliseconds. delayRange="50-200" means delay time will be an random number between 50 and 200. If this attribute is not present, no intentional delay time.

▪ Instances Section

Instances section stores MIB walk instances. Each instance node in this section represents a MIB node.

Sample:
<Instances
interval="10"
readCommunity="public"
writeCommunity="public"
>

interval:	The interval between SNMP walks, in seconds. If interval passes, simulator will advance to next set of SNMP walk data. If it is not present, only the first set of SNMP walk data will be used.
readCommunity:	SNMP community name for READ operations, such as GET, GET_NEXT and GET_BULK.
writeCommunity:	SNMP community name for SET operation.

Instance Node

Instance node in this section represents a MIB instance node.

Sample:

```
<Instance
  oid=".1.3.6.1.2.1.1.6.0"
  valueType="OctetString"
  trapOnSet="trap1"
  trapOnSetCount="2"
  name="sysLocation"
>
   <Value> <![CDATA[sysLocationValue1]]> </Value>
   <Value na="yes"/>
   <Value> <![CDATA[sysLocationValue3/4]]> </Value>
   <Value same="yes"/>
</Instance>
```

Field	Description
oid	Object identifier of this node.
valueType	The data type of this node, possible values are {OctetString, Null , OID , BITS , UnsignedInteger , Counter32 , Gauge , TimeTicks , IpAddress , Opaque , Integer , Counter64 }
trapOnSet	Optional. If it's not empty, a trap is generated for each SET request for this node. Its value is the name of trap.
trapOnSetCount	Number of traps to be sent.
name	Optional. The name of this node, to be used by other nodes for referencing it.

ScriptOnSet

Optional. A script will be executed on SET request.

Example:

```
<Instance
        oid=".1.3.6.1.2.1.1.6.0"
        valueType="OctetString"
        trapOnSet="trap1"
        trapOnSetCount="2"
        name="sysLocation"
    >
        <Value> <![CDATA[sysLocationValue1]]> </Value>
        <ScriptOnSet>
                    <![CDATA[
                      print("pdu is :" + pdu);
                      exec("an external command");
                    ]]>
        </ScriptOnSet>
    </Instance>
```

Value Node

Value nodes store values of a MIB node. There can be one or multiple value nodes. Multiple value nodes are used to represent multiple SNMP walks, each corresponding to one SNMP walk. The interval of nodes is specified in the *Instances* node. For example, the interval is 1 minute and each Instance node has three Value nodes. When simulator starts, the first Value node of each Instance node is used to compute the response for SNMP request. At minute 2, the second Value node is used. At minute 3, the third Value node is used. Since no more Value nodes after minute 3, the third Value node continues to be used until simulator stops.

If an Instance node has less Value nodes than other Instance nodes, the last Value node is used after SNMP walks exceed all the Value nodes it has.

1. Static value

<Value> <![CDATA[sysLocationValue1]]> </Value>

CDATA node stores static value that is not going to change.

2. Same value

<Value same="yes"/>

It cannot be the first value node. It means that this value is the same as the preceding value node.

3. Disabled value

<Value na="yes"/>

It means that this MIB node is skipped and no value is returned for SNMP requests.

4. Error simulation

```
<Value errorOnSet="badValue" errorOnGet="noSuchName">
        <![CDATA[90]]>
</Value>
```

Field	Description
errorOnSet	If it's not empty, the error status field in the SNMP response PDU indicates the specified error when receiving a SNMP SET request.
errorOnGet	If it's not empty, the error status field in the SNMP response PDU indicates the specified error when receiving a SNMP GET/GET_NEXT/GET_BULK request.
CDATA	If error condition is not satisfied, no error and CDATA's value is returned.

5. Simple script

```
<Value isSimpleScript="y" depends="sysName">
      <![CDATA[sysName + " was used as sysDescr"]]>
</Value>
```

isSimpleScript: If its value is "y" or "yes", the CDATA value is a simple script, which can only have one line, and value of this expression is returned.

6. Full script

<Value script="script2"/>

script: Its value indicates that returned value is from the value of the specified script. In the above example, the return value of a script named *"script2"* will be used as the value of this MIB node.

Scripts Section (optional)

This section contains one or more Script nodes. Script node essentially is a definition of a function.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<SnmpSimulatorData>
    <Instances readCommunity="public">
        <Instance oid=".1.3.6.1.2.1.1.4.0" valueType="OctetString" >
            <Value><![CDATA[somebody@testing.com]]></Value>
        </Instance>
  </Instances>

  <Scripts>
    <ScriptInit>
        <![CDATA[
                int global_count = 0;
                open_db_connection();
        ]]>
    </ScriptInit>
    <Script name="script1"><![CDATA[
      Object getValue(String oid, SnmpPdu pdu)
      {
        exec("execute a command.");
        return “value 1”;
      }
      ]]>
      </Script>
  </Scripts>
</SnmpSimulatorData>
```

ScriptInit Node

Sample:

```
<ScriptInit>
  <![CDATA[
          int global_count = 0;
          open_db_connection();
  ]]>
</ScriptInit>
```

The scripts in this node will be executed when simulator starts.

ScriptDestroy Node

Sample:

```
<ScriptDestroy>
  <![CDATA[
          close_db_connection();
  ]]>
</ScriptDestroy>
```

The scripts in this node will be executed when simulator stops.

Script Node

Sample:

```
<?xml version="1.0" encoding="UTF-8"?>
<SnmpSimulatorData>
    <Instances readCommunity="public">
        <Instance oid=".1.3.6.1.2.1.1.4.0" valueType="OctetString">
            <Value script="script2"/>
        </Instance>
    </Instances>

    <Scripts>
        <Script name="script2" depends="sysName, sysLocation">
            <![CDATA[
            Object getValue(String oid, SnmpPdu pdu)
            {
              return "testvalue";
            }
            ]]>
        </Script>
    </Scripts>
</SnmpSimulatorData>
```

Field	Description
name	name of this script. It's used by others to reference this script.
depends (optional)	Node names used in this script, separated by comma.

getValue function:

Object getValue(String oid, SnmpPdu pdu)

Parameters:

oid: Object identifier of the SNMP request being processed
pdu: Received SNMP PDU object being processed

Returns:

Return value has to be either string or an object implementing SnmpDataType interface, such as SnmpOctetString, SnmpOID, etc.

▪ Traps Section (optional)

This section contains one or more trap nodes, which define SNMP traps.

▪ Trap Node

1. SNMPv1 trap

Sample:

```
<?xml version="1.0" encoding="UTF-8"?>
<SnmpSimulatorData>
    <Instances readCommunity="public">
        <Instance oid=".1.3.6.1.2.1.1.4.0" valueType="OctetString" >
            <Value><![CDATA[somebody@testing.com]]></Value>
        </Instance>
    </Instances>

    <Scripts>
        <Script name="script2" depends="sysName, sysLocation">
            <![CDATA[
            Object getValue(String oid, SnmpPdu pdu)
            {
               sendTrap("trap1");
            }
            ]]>
        </Script>
    </Scripts>
    <Traps>
        <Trap
          name="trap1"
          generic="linkDown"
          specific="0"
        >
          <Variable use="sysDescr"/>
          <Variable
           oid=".1.3.6.1.2.1.1.2.0"
           valueType="OctetString"
          >
        </Trap>
    </Traps>
</SnmpSimulatorData>
```

Field	Description
name	Name of this trap. It's used by others to reference this trap.
generic	Generic-trap field, either numeric or string value.
specific	Specific-trap field, numeric value.

2. SNMPv2c/v3 trap

```
<Trap
 name="trap1"
 snmpTrapOID=".1.3.6.1.2.1.1.15135"
 >
      <Variable use="sysDescr"/>
</Trap>
```

Field	Description
name	Name of this trap. It's used by others to reference this trap.
snmpTrapOID	Object identifier of snmpTrapOID

▪ Trap's Variable Node

A Trap Node can have one or more Variable nodes. Variable node defines variable bindings of this trap. It can take the following formats:

1. Static value

The value is contained in CDATA node.

```
<Variable
oid=".1.3.6.1.2.1.1.2.0"
valueType="OctetString"
>
        <![CDATA[222etest faf a]]>
</Variable>
```

Field	Description
oid	Object identifier of this variable
valueType	Data type of value
CDATA	Value of this variable

2. Reference

The value is the same referenced node.

```
<Variable use="sysDescr"/>
```

Uses the OID and value of "sysDescr" node

3. Simple script

The value is the result of the simple script contained in CDATA node.

```
<Variable
 oid=".1.3.6.1.2.1.1.3.0"
 valueType="TimeTicks"
 isSimpleScript="yes"
 depends="sysName"
>
 <![CDATA[getSysUpTime() + 1000]]>
</Variable>
```

Field	Description
oid	Object identifier of this variable.
valueType	Data type of value.
isSimpleScript	If yes, the content of CDATA node is treated as a simple script.
depends	Optional. Names of nodes that are used in the simple script, separated by comma.
CDATA	Contain simple script

4. Full script

The value is the result of the script whose name is the value of script attribute.

```
<Variable
 oid=".1.3.6.1.2.1.1.4.0"
 valueType="OctetString"
 script="script2"
/>
```

Field	Description
oid	Object identifier of this variable.
valueType	Data type of value.
script	Name of script. The returned value of the script is used as the value of this variable.

▪ Thresholds Section (optional)

This section contains one or more Threshold nodes.

Sample:
<Thresholds interval="1">

interval: interval of invoking isTimeToSendTrap() function in each Threshold node, in seconds. Simulator invokes *isTimeToSendTrap()* function periodically to check if threshold is crossed or not.

Threshold Node

This node defines *isTimeToSendTrap()* function, which determines whether to send out trap, or not.

Sample:

```
<Threshold trapName="trap1" trapCount="1" depends="sysint">
<![CDATA[
boolean isTimeToSendTrap()
{
   int i = Integer.parseInt(sysint) ;
   if(i <= 200)
   {
      return true;
   }
   return true;
}
]]>
</Threshold>
```

trapName:	Name of trap that will be sent when isTimeToSendTrap() function returns true.
trapCount:	Number of traps sent when isTimeToSendTrap() function returns true.
depends:	Optional. Names of nodes, which are used in isTimeToSendTrap() function, separated by comma.

isTimeToSendTrap() function:

Return true to send out trap to all trap destinations defined in simulator's config file; false otherwise.

Scripting Language

Script uses BeanShell scripting language. BeanShell is a small, embeddable, scripting language with object scripting language features, written in Java. BeanShell executes standard Java statements and expressions, in addition to obvious scripting commands and syntax. BeanShell supports scripted objects as simple method closures like those in Perl and JavaScript. Click here to read complete manual of BeanShell.

SNMP Trap Data File

This file stores traps recorded from trap senders. It can be customized after recording or created from scratch.

Simulator sends out all traps in this file after it is started.

- **Root Node**

  <SnmpSimulatorTrapData ***trapTimestampOffset****="0">*

  trapTimestampOffset: Time offset of the sysUpTime.

- **Traps Section**

  This section contains one or more Trap nodes. Simulator sends out traps based on their time value. For instance, for a trap node with time="1000" a trap will be sent out 1000 milliseconds after simulator is started. Trap nodes do not have to be ordered by their time value.

Trap Node

- **SNMPv1 Trap**

```
<Trap
   generic="linkDown"
   specific="0"
   time="200"
   count="3"
   ipaddress="192.168.1.101"
>
```

- **SNMPv2 Trap**

```
<Trap
   snmpTrapOID=".1.3.6.1.2.1.1.1.15511"
   time="200"
   count="3"
>
```

generic:	Generic-trap field, either numeric or string value.
specific:	Specific-trap field, numeric value.
enterpriseOID:	Enterprise OID of SNMPv1 traps.
snmpTrapOID:	Object identifier of snmpTrapOID (SNMPv2/v3 traps).
time:	The time when this trap should be sent, in centiseconds (1/100 seconds).
count:	Number of traps to be sent out.
ipAddress:	IP address field in SNMPv1 traps.

Variable Node

Variable nodes are child nodes of Trap node. They define variable bindings of trap nodes.

Sample:

```
<Variable
oid=".1.3.6.1.2.1.1.1.0"
valueType="OctetString"
>
      <![CDATA[111etest faf a]]>
</Variable>
```

oid:	Object identifier of this variable.
valueType:	Data type of value.
CDATA:	Value of this variable.

Simulator Configuration

Simulator configuration specifies properties of trap sinks and SNMPv3 properties.

SNMPv3 properties Section

The properties node defines some agent properties.

version	version number of this agent. 1 for snmpv1, 2 for snmpv2c, 3 for snmpv3
engineID	SNMPv3's engineID of this agent. If it's empty, the first IP address found for the hostname of the machine will be used.
engineBoots	Number of times of agent reboots. Its value is updated whenever agent restarts if agent is a SNMPv3 agent

If version number is 3, so agent is an SNMPv3 agent. It then will reject all SNMPv1/v2c requests for security concerns. But if version number is 2, agent still can handle SNMPv1 requests.

TrapSink Section

This section defines the properties of trap receivers. Agent will send out traps to all those trap receivers.

- **SNMPv1 and SNMPv2 TrapSink**

hostname	host name or IP address of trap receiver
port	port number of trap receiver
community	trap receiver's community name
version	trap receiver's SNMP version number
isInform	set "yes" to send SNMP INFORM request instead of trap. INFORM request is more reliable than trap.

- **snmpV3TrapSink**

hostname	host name or ip address of trap receiver
port	port number of trap receiver
isInform	set "yes" to send SNMP INFORM request instead of trap
userName	one of user names in trap receiver's user list
auth	authentication algorithm, one of "MD5" and "SHA"
authPassword	authentication password
priv	privacy algorithm, one of "DES" and "AES". Default is "DES"
privPassword	privacy password

In snmpV3TrapSink section, security level is determined by the authPassword and privPassword. If both of them are empty strings, security level is noAuthNoPriv. If privPassword is empty string, security level is authNoPriv. If both of them are non-empty strings, security level is authPriv.

For example, we define three trapsinks, one SNMPv1 , one SNMPv2 and one SNMPv3 trapsinks. If agent sends a SNMPv2 trap, this trap will be sent to all of the three trapsinks. It'll be converted to SNMPv1 trap before sending to SNMPv1 trapsink. If agent sends a SNMPv1 trap, it will be converted to SNMPv2 trap before sending to SNMPv2 and SNMPv3 trapsinks.

User Section

This section defines the authorized users' properties.

name	User name. It should be unique, that is, two users can't have same name.
auth	Authentication algorithm. One of "MD5" and "SHA"
authPassword	Authentication password
priv	privacy algorithm, one of "DES" and "AES". Default is "DES"
privPassword	Privacy password
group	The group that this user associated with

User's security level is determined by the security level of its group, not by the values of authPassword and privPassword. This is different from snmpV3TrapSink node. If security level is authNoPriv, privPassword field's value will be ignored.

Group Section

This section defines SNMPv3 group properties.

name	Group name. It should be unique, that is, two groups can't have same name.
securityLevel	Security level. One of { noAuthNoPriv, authNoPriv, authPriv}
match	Context match. Either "prefix" or "exact".
contextPrefix	If the match is "prefix", context match only checks if context starts with this contextPrefix. But if the match is "exact", context must be exactly matched, so this contextPrefix is ignored in this case.
readView	The view associated with this group for "READ" operations, such as GET, GETNEXT, GETBULK.
writeView	The view associated with this group for "WRITE" operations, such as SET
notifyView	The view associated with this group for notification operations

Note: SecurityLevel includes { noAuthNoPriv, authNoPriv, authPriv}. "noAuthNoPriv" means no authentication and encryption apply to packets. "authNoPriv" means only authentication applies to packets. "authPriv" means both authentication and encryption applies to packets. All users in a group have same security level.

View Section

This section defines SNMPv3 VACM view properties.

name	View name. Multiple views can have same name.
type	View type, either "included" or "excluded". Each view subtree in the MIB view is specified as being included or excluded. That is, the MIB view either includes or excludes all object instances contained in that subtree.
subTree	Subtree oid. A subtree is a node in the MIB's naming hierarchy plus all of its subordinate elements.
mask	A list of "0" or "1", separated by '.' or ':'. View mask is defined in order to reduce the amount of configuration information required when fine-grained access control is required. Each bit of this bit mask specifies whether or not the corresponding sub-identifiers must match when determining if an OBJECT IDENTIFIER is in this family of view subtrees; a '1' indicates that an exact match must occur; a '0' indicates 'wild card', i.e., any sub-identifier value matches.

For example:

```
<view  name    = "view1"
       type    = "included"
       subTree = ".1.3 "
       mask    = ".1.1"
/>
```

It defines a view1, which includes all tree nodes whose OIDs start with ".1.3"

```
<view  name   = "view1"
       type   = "included"
       subTree = ".1.3.6.1.2.1"
       mask    = ".1.1.1.1.1.0"
/>
```

It defines a view1, which includes all tree nodes whose OIDs start with ".1.3.6.1.2". The last digit of mask is 0, which means it does not care about the subtree's OID at that index.

```
<view  name   = "view1"
       type   = "included"
       subTree = ".1.3.6.1.2.1.7"
       mask    = ".1.1.1.1.1.0.1"
/>
```

In this view, all OIDs in .1.3.6.1.2.[1, 2, ...].7.* are included.

```
<view  name   = "view1"
       type   = "included"
       subTree = ".1.3.6.1.2.1"
       mask    = ".1.1.1.0.1.0"
/>
```

In this view, all OIDs such as ".1.3.6.1.2.1.*", ".1.3.6.2.2.1.*", ".1.3.6.3.2.2.*" are included.

Simulator configuration Example

Here is a sample agent configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<SnmpAgent>
  <properties
      version            = "3"
      engineID           = "12345"
      engineBoots        = "0"
  />
  <trapSink
        hostname     = "localhost"
        port         = "162"
        community    = "public"
        version      = "2"
        isInform     = "yes"
  />
  <user
       name          = "newUser"
       auth          = "MD5"
       authPassword  = "abc12345"
       privPassword  = "abc12345"
       group         = "aGroup"
  />
  <group  name           = "aGroup"
          securityLevel  = "authPriv"
          contextPrefix  = ""
          match          = "exact"
          readView       = "view1"
          writeView      = "view1"
          notifyView     = "view1"
  />
  <view name     = "view1"
       type     = "included"
       subTree  = ".1.3"
       mask     = ".1.1"
  />
</SnmpAgent>
```

In the properties sections, we can tell that it is a SNMPv3 agent. It defines a user called "newUser", which belongs to the *aGroup*. This *aGroup* maps to *view1*, which allows access to the subtree of ".1.3" (which means all SNMP OID starts with ".1.3" are allowed). All *aGroup*'s member users require authPriv, so authentication and privacy password have to be provided for user "newUser". Authentication password is "abc12345", and authentication uses MD5 algorithm. Privacy password is also "abc12345" and encryption use "DES" algorithm by default.

Examples

- ## Context indexing

This sample simulator is located at examples/context. This samples demonstrates context indexing, that is, different data is returned for different community names or contexts.

- ## Cisco router SNMP Agent Simulator

This sample simulator is located at examples/cisco. To run this example, launch simulator GUI and click on "File/Open Project" menu to load examples/cisco/cisco.prj. Or run examples/runCiscoAgent.bat. This simulator has data collected from a CISCO router.

- ## Windows 2000 SNMP Agent Simulator

This sample simulator is located at examples/win2000. To run this example, launch simulator GUI and click on "File/Open Project" menu to load examples/win2000/win2000.prj. Or run examples/runWin2000Agent.bat. This simulator has data collected from default SNMP agent on Windows 2000.

- ## Multi-Walk Agent Simulator

This sample simulator is located at examples/multiwalk. This example demonstrates an SNMP walk data file that has four SNMP walks. The interval between walks is 10 seconds. After agent starts, values of MIB instance nodes change every 10 seconds until the last set of SNMP walk data is loaded. It also shows the usage of error simulation, scripting language, threshold nodes, and trap nodes.

Resources & Reference

- iReasoning Home Page

  http://www.iReasoning.com/

- SNMP FAQ

  http://www.faqs.org/faqs/snmp-faq/part1/

- SNMP RFCs
  http://directory.google.com/Top/Computers/Internet/Protocols/SNMP/RFCs/

- SNMP General Information

  http://www.simpleweb.org/

- Understanding SNMP MIBs, David T. Perkins, Prentice Hall PTR, 1997

  http://vig.prenhall.com/catalog/academic/product/1,4096,0134377087.html,00.html

- JAVA web site

  http://java.sun.com/