iReasoning SNMP Agent Simulator User Guide

Copyright © 2002-2026 iDeskCentric Inc, All Rights Reserved.

The information in this document is the property of iReasoning Inc. It may not be copied, reproduced, converted to an electronic or machine-readable format, or otherwise duplicated. The information may not be used, shared, or disclosed without the prior written consent of iDeskCentric Inc.

Table of Contents

REQUIREMENTS **4**

INSTALLATION PROCEDURES **4**

OVERVIEW **6**

MAJOR FEATURES **7**

USAGE OF GRAPHICAL USER INTERFACE **8**

COMMUNITY/CONTEXT INDEXING **11**

USAGE OF COMMAND LINE INTERFACE **12**

AUTOMATION AND REPEATABLE TESTING **15**

Create and manage labs in the GUI 15
Create and run scenarios in the GUI 15
Checks tab *16*
Advanced tab *18*
Copy commands and export standalone scenarios 18
Getting started 19
Run your first scenario 19
Control agent state 21
Override recorded values 22
Simulate faults and notifications 23
Write scenarios and assertions 23
Start the REST service 24
Create and control a lab through REST 25
Automate the workflow with Python 26
Operate the simulator with Claude Code 28
Requirements *28*
Install the skill *28*
Connect to the simulator *28*
Make a request in Claude Code *29*
Understand progress and results *29*
Troubleshooting and removal *30*
Read reports and diagnose problems 30
Credentials and remote deployment 31

HELPER FUNCTIONS **32**

SNMP WALK DATA FILE **37**

SCRIPTING LANGUAGE **50**

SNMP TRAP DATA FILE **50**

SIMULATOR CONFIGURATION **53**

Simulator Configuration Example 59

EXAMPLES **60**

Context indexing....60
Cisco router SNMP Agent Simulator....60
Windows 2000 SNMP Agent Simulator....60
Multi-Walk Agent Simulator....60

Chapter

1

Installation

Requirements

- Windows, Linux, macOS, and UNIX operating systems
- At least 1024 MB of memory is required.

Installation Procedures

1. **Installing the simulator**

   On Windows, run setup.exe to start the simulator installer. On Linux, unzip simulator_linux_x64.zip. On macOS, run simulator.pkg. On UNIX, unzip *simulator.zip* (Java 21 or later must be installed).

   The installation has the following directory structure:

Directory Name	Description
bin	Contains Windows scripts
devices	Contains device library data
doc	Contains the user guide in PDF and HTML formats
jre	Java Runtime Environment
mib	MIB files
lib	Contains binary JAR files
log	Stores log files
config	Stores the simulator configuration file
examples	Cisco router and Windows 2000 SNMP agent simulators

2. **Starting the simulator**

   On Windows, double-click the simulator icon on the desktop or in the Start menu. You can also run the bin\runUI.bat script from the command line.

   On macOS, click the “SnmpSimulator” icon in the Applications folder.

   On other platforms, run bin/runUI.sh.

   After startup, you can create a project or open an existing one. Opening a project for the first time, or after a data file changes, may be slow because the simulator converts the XML data and inserts it into the database. Subsequent openings are faster once the data is stored.

Chapter

2

Using SNMP Agent Simulator

Overview

SNMP Agent Simulator is a Java-based application that simulates SNMPv1, SNMPv2c, and SNMPv3 agents. It runs on platforms with a Java Virtual Machine, including UNIX, Linux, macOS, and Windows. You can develop, test, and train with SNMP management applications without purchasing or maintaining hardware devices.

Typical uses of SNMP Agent Simulator include:

- Developers and testers can work without physical SNMP agents. Simulator data files are easy to modify, making it possible to test a wider range of cases.
- Sales and training staff can give live demonstrations on a laptop without carrying network devices. A single laptop can simulate a complex network.

Major Features

- Ease of use
- Record and replay SNMP devices

  You can record SNMP agent MIB values to an XML file through the GUI or command-line interface. You can then edit the file to simulate different configurations.
- Multiple agents in one JVM
- Support for dynamic row creation and deletion
- BeanShell scripts in data files can model real-time SNMP agent behavior
- Trap simulation

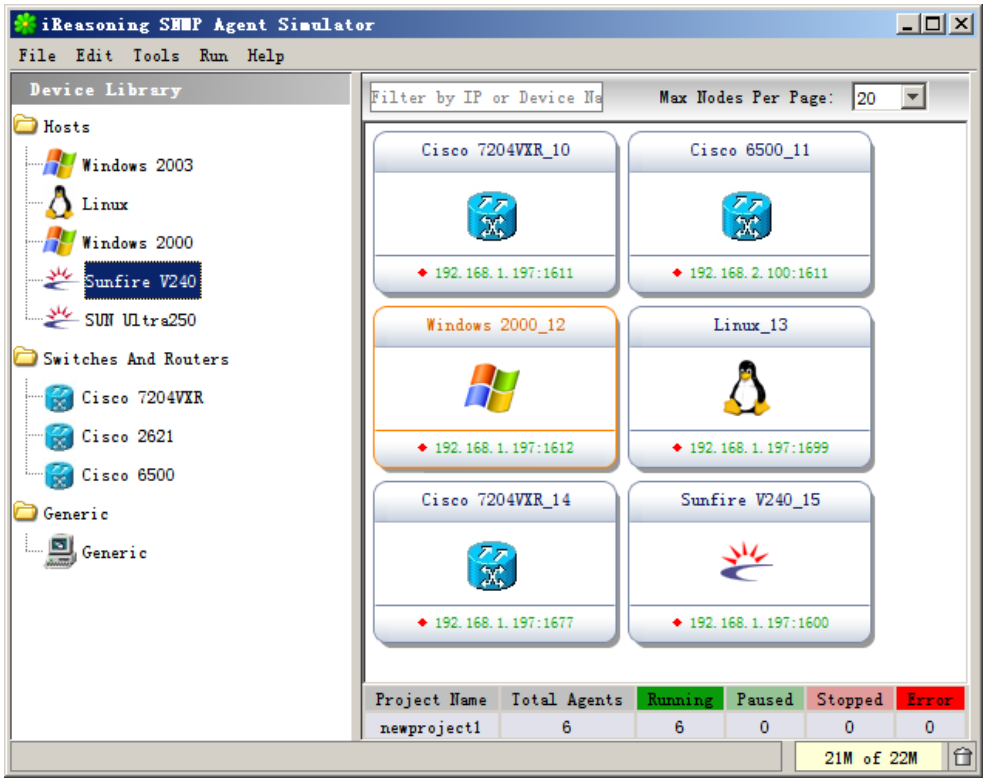
  The simulator can generate request-based, threshold-based, and timer-based SNMPv1, SNMPv2c, and SNMPv3 traps and informs. It can also simulate a trap storm.
- Error simulation
- Scalability

  The number of agents a computer can run depends on its available resources.
- Community/context indexing (different data for different communities or SNMPv3 contexts)

Release 1 adds scenarios, REST/Python control, OID overrides, fault simulation, and JSON/JUnit reports.

Usage of Graphical User Interface

- **Main window**



The main window has two panes. The left pane contains predefined devices that you can drag into the right pane. The right pane shows the simulators and their status. Right-click a simulator to change its status or settings from the context menu.

▪ Generating Data from MIBs

You can also generate SNMP walk data from MIB files.

Select "Tools/Generate Data from MIBs." To enter multiple MIBs, separate them with commas.

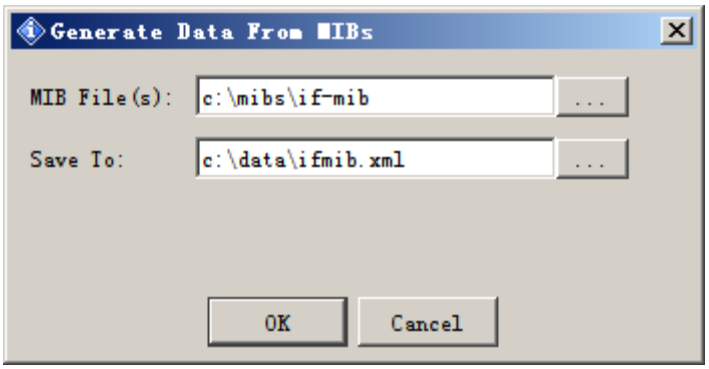


Figure: Generate Data from MIBs

In this example, SNMP walk data is generated in "c:\simulators\data\mibdata" from MIB-II and the host MIB.

To add the generated data to a simulator, right-click the simulator, select "Properties," and change its data file property.

▪ Recording SNMP Traps

You can record SNMP traps and add them to a simulator. When the simulator starts, it sends the recorded traps to the trap sinks defined in its configuration file at the specified times.

Select "Tools/Record SNMP Trap."

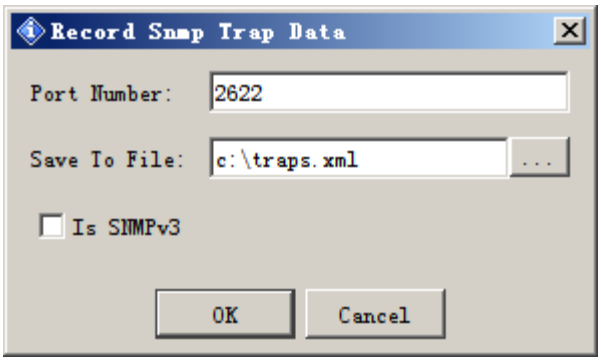


Figure: Record SNMP Trap

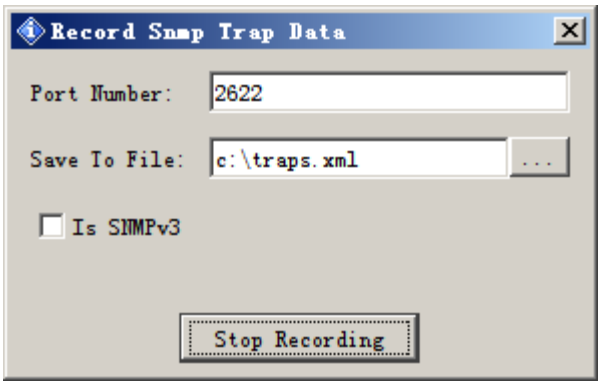


Figure: Recording traps...

Click "Stop Recording" when you have finished. In this example, the trap data is stored in "c:\traps.xml."

To add the recorded traps to a simulator, right-click the simulator, select "Properties," and change its trap file property.

Community/Context Indexing

An SNMP agent can have multiple data files, each associated with a different community or SNMPv3 context. The community or context in a request determines which data file the agent uses.

For example, the bundled context sample agent (in examples/context) specifies two data files in its project file:

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

At the beginning of linux1.xml:

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

At the beginning of linux2.xml:

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

The first data file uses the community/context *public1*, and the second uses public2.

For an SNMPv1 or SNMPv2c agent, these values are community names. For an SNMPv3 agent, the readCommunity value becomes the context name.

The SNMPv3 parameters are:

Field	Value
User Name	newUser
Auth Algorithm	MD5
Auth Password	abc12 345
Priv Algorithm	DES
Priv Password	abc1345
Context	public1 or public2

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

contextPrefix="pub"
match="prefix"

The default setting is match="exact". Because both context names begin with pub, the sample uses prefix matching.

When you query the agent with an SNMP manager, it returns different data for the two community or context names.

Usage of Command Line Interface

Go to the ireasoning/simulator/bin directory.

- runSimulator.bat/runSimulator.sh

  Usage: runSimulator projectFile

  This script runs a project file, the same type of file used by the GUI. A project file defines simulator properties, including data files and initial states. A simulator whose initial state is "up" starts when the script runs.

- runSnmpTrapRecorder.bat/runSnmpTrapRecorder.sh

  Usage: runSnmpTrapRecorder [options...]

  Available options:
  -? 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 saves them to an XML file. Press Ctrl+C to stop recording.

- 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

  Available options:
  -? 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 saves it to an XML file. Press Ctrl+C to stop recording.

- netsnmp.bat

  Converts Net-SNMP walk output to a data file.

Automation and Repeatable Testing

Release 1 provides desktop tools for labs, timed scenarios, fault simulation and results. Start with the GUI instructions below. The local runner, REST, Python client and Claude Code skill provide additional automation options.

Create and manage labs in the GUI

Open the **Labs** tab in the main simulator window. A lab groups agents, scenarios and results. You can complete the following workflow without writing JSON, XML or command-line scripts.

•Click **Create Example Lab** to try the supplied router and outage scenario. The example uses 127.0.0.1 and UDP port 1161. Alternatively, click **New Lab** for an empty lab or **Import Project** to import an existing .prj file and its resources.

•Select the lab icon. The right pane contains **Agents**, **Scenarios** and **Results**.

•In Agents, click **Add Agent**. Enter a name, IP address and UDP port, and choose an SNMP walk recording. Optional files provide MIB definitions, agent configuration and recorded traps. Communities and SNMPv3 settings come from the recording and configuration. Each agent needs a distinct IP/port combination. You can also select a device in the library and click **Add Library Device**, or drag it into the Agents pane.

•Right-click a lab icon to **Start**, **Stop**, **Pause**, **Resume**, **Reset**, **Rename** or **Delete Lab**. Right-click an agent for its lifecycle controls, properties or removal. Stop the whole lab before adding, removing or changing agents. Deleting a lab removes its saved scenarios and reports; export reports you want to keep.

Labs, imported assets, saved scenarios and reports are stored in the desktop workspace and survive application restarts. Recovered agents remain stopped. A run interrupted by application shutdown is recorded as an error rather than silently resumed. The **Project** tab retains the existing project view.

Create and run scenarios in the GUI

•Select a lab and open **Scenarios**. Click **New Scenario**, enter its name and duration in milliseconds, then add actions. One second is 1000 ms.

•For each action, choose its time and action type. Agent actions select target agents; Send trap / INFORM opens a Trap Sender-style form for destination settings and editable trap content. Supported types are Start agents, Stop agents, Apply recorded-value overrides, Clear overrides, Apply fault, Clear fault and Send trap. Targets are shown by name and endpoint. When an agent has multiple recordings, choose the recording to change.

•Use the parameter forms to enter typed OID values, configure delay, loss, SNMP error, outage or reboot faults, or select the notification content from a trap recording. **Outage template** supplies a six-second scenario that starts agents at 0 ms and interrupts responses at 2000 ms for one second.

•Optional **Checks** verify observed requests, SET values and other traffic evidence. Choose an agent for agent checks, or a notification action for notification counts, plus a time window and expected counts or values. An external SNMP manager must generate requests; the simulator does not generate manager traffic for these checks.

•Actions and checks save automatically after Add, Edit, Duplicate or Remove. Name, duration and seed changes save automatically after typing. Import JSON and Outage template also save automatically. The editor shows Saving or Saved; invalid fields must be corrected before they can be saved. Stop the lab, then click **Run** in the editor or select the saved scenario and click Run in the Scenarios tab. Changes saved during a run apply to later runs.

•Open **Scenarios** and its lower **Run results** section to see active scenario elapsed/total time and paused state above the displayed run's outcome, check results and events. Active progress remains visible when a historical run is selected. Run history contains all runs for this lab; Recent lab events also includes manual agent activity. Successful runs select the newly started run. Adjust the divider between scenarios and results, or use its collapse/expand controls. The result text can be selected and copied with Ctrl/Cmd+A and Ctrl/Cmd+C, or the right-click Select All and Copy menu. Live refresh preserves the scroll position and text selection. The vertical scrollbar stays visible. Run events belong to the selected run and are saved with its report, including agent notifications and requests. Completed or cancelled run histories remain available after restarting and in JSON and lab archive exports. Each run retains its latest 10,000 events; any omitted earlier events are counted in the results. Older reports created before event history was saved explain that the missing events cannot be recovered. Recent lab events shows separate activity from the current application session. Use **Cancel Run** to end the active run. Export a selected result as JSON or JUnit XML.

Pause freezes simulation time; Resume continues it. Reset returns a stopped lab to its initial recording state. The scenario editor also supports JSON import/export for advanced automation. The seed is available under Advanced for repeatable fault behavior.

In **Send trap / INFORM**, configure the destination IP/port, protocol, community or SNMPv3 security, notification identifier and uptime in seconds. The Community field is visible and defaults to `public`. Retries and timeout apply to INFORMs. Add, modify or delete typed variable bindings; **Suffix** appends numeric arcs to all binding OIDs. Octet strings accept text or hexadecimal bytes. **SNMPv3 Settings…** offers security level, USM user, authentication/privacy algorithms and passwords, context, and an optional Engine ID. Blank uses an automatic identity; INFORMs discover the receiver engine.

Load Recorded Trap… imports a lab recording or another file. Choose an entry by position, identifier and recorded time, then edit its copied content. Destination, type and security settings stay as configured. The saved action is independent of agents and recording files. **OK** adds the scheduled action; it does not send immediately. Cancel discards the draft. Versions 1 and 2 scenarios must be recreated as version 3.

Scenario notification actions send once at their scheduled time (INFORMs may retry while awaiting acknowledgement).
Agents started by a scenario also send their normal trap recordings and threshold/script/SET notifications.
These agent notifications are independent of the one-time scenario actions.
Notification checks select an action
and count only its events.

Checks tab

The optional **Checks** tab defines what a scenario must observe to pass. Checks monitor events while the scenario runs; they do not send SNMP requests or schedule actions. Use **Add** to create a check, **Edit** to change the selected check,

and **Remove** to delete it. Confirmed changes save automatically. Changes made during a run apply to later runs.

At normal completion, all checks must meet their minimum and maximum counts for the run to be **SUCCEEDED**. A count outside its range makes the run **FAILED**. Runtime errors or cancellation have their own outcomes. With no checks configured, success means the actions completed; it does not verify that an external manager received a trap or that a device responded correctly.

Available checks

- **Requests received:** counts decoded, authorized requests received by the selected agent. An external SNMP manager, such as MIB Browser, must generate the traffic.
- **SET value received:** counts SET requests containing the configured object identifier, data type and expected value. This verifies receipt of that value, not successful application of the SET. Use **Expected SET value > Configure...** to enter the OID, type and value.
- **SNMP errors returned:** counts transmitted responses containing an SNMP error. Leave **Error status** blank to count any nonzero status, or enter a particular numeric status, such as `5` for `genErr`.
- **Notifications attempted:** counts attempts by the selected scenario **Send trap / INFORM** action.
- **Notifications sent:** counts local transmissions by that selected action. A successful UDP transmission does not prove that the receiver received the notification.
- **INFORM acknowledgements:** counts successful acknowledgements received for the selected scenario notification action. Use this check with an INFORM action.
- **Requests during outage:** counts packets received while the selected agent is unavailable because of an outage or reboot. These packets are discarded before decoding; leave OID, recording and PDU filters blank.

Agent checks select an **Agent**. Notification checks select a **Notification action** already present in the Actions tab. They count only that action's events. Automatic traps from agent recordings, thresholds, scripts or SET behavior are independent and do not contribute to a notification action's check.

Configure a check

- **Window start (ms)** and **Window end (ms, exclusive)** define the observation window relative to the start of the scenario. The start is included and the end is excluded: a window from `3000` to `4000` counts events at 3000 ms through just before 4000 ms. Use a window within the scenario duration. Pausing the lab freezes simulation time.
- **Minimum count** and **Maximum count** define the acceptable range, including both limits. A blank maximum means unlimited. Set both to `1` to require exactly one event; set minimum to `1` and leave maximum blank to require at least one; set both to `0` to require none. The default minimum is `1`.
- **OID filter (optional)** matches an exact object identifier in a decoded request, rather than an OID subtree. Leave it blank to accept any OID.
- **Recording** selects a recording/context or **All recordings**. This is useful when an agent serves several recordings through community/context indexing.

- **PDU type (optional)** accepts a numeric SNMP command code, such as `0` for GET, `1` for GETNEXT, `3` for SET, or `5` for GETBULK. Leave it blank to accept any request type.
- **Error status (optional)** applies to **SNMP errors returned**. **Expected SET value** applies to **SET value received**. Notification checks do not use the agent-specific filters.

Give actions enough time to complete when choosing a window. An action scheduled at 3000 ms can transmit slightly later, and an INFORM acknowledgement can arrive later still. Requests and responses are separate events and can fall into different windows.

Check examples

Send exactly one trap: add a Send trap / INFORM action at `3000` ms. Add a **Notifications sent** check, select that action, set the window to `3000` through `10000` ms, and set minimum and maximum to `1`. Use a scenario duration of at least 10000 ms. Agent-generated traps do not affect this count.

Confirm an INFORM acknowledgement: configure an INFORM action and a reachable receiver. Add an **INFORM acknowledgements** check for that action, use a window that includes the configured retry/timeout period, set minimum to `1`, and leave maximum blank.

Observe manager requests: start an agent at 0 ms. Add a **Requests received** check for that agent with a window from `0` to `10000` ms, minimum `3`, and maximum blank. During the run, send at least three authorized requests from MIB Browser. If no manager sends requests, the check fails.

The Run results panel shows PASS or NOT PASSED and the observed count for each check. Check results are retained in the saved report and available in JSON and JUnit exports.

Advanced tab

The **Advanced** tab currently contains **Repeatable random seed**, which defaults to `0`. Enter a whole number using 1 to 30 decimal digits, with an optional leading minus sign. The seed saves automatically and is included in the run report.

The seed controls the simulator's random sequences, including probabilistic packet loss, probabilistic fault selection, variable delays and modeled random values. It does not change action times, scenario duration, notification destinations or SNMP credentials.

For a repeatable test, keep the seed, agent identities, fault rule identities, recordings, configuration and ordered request traffic the same. For example, use seed `42` with a packet-loss fault to repeat the same modeled sequence of decisions for the same traffic. Change the seed to `43` to exercise a different sequence; this does not change the configured loss probability or delay range.

Repeatable random decisions do not guarantee identical wall-clock timings or packet traces. Operating-system scheduling, external manager traffic, retries and scripts with their own clocks or random generators can still change the observations. Keep the default seed when you do not need to choose a different test sequence. See Simulate faults and notifications for fault configuration.

Copy commands and export standalone scenarios

Right-click a lab and select **Show Command Line**, or use the toolbar button. Choose an operation and platform: macOS/Linux shell or Windows PowerShell. For Run, first select a saved scenario; for Report, first select a result. The

command uses the installed simulator-control launcher, which waits for asynchronous operations. Click **Copy** or **Save Script**, then run it in a terminal.

Commands control the same labs as the desktop through its REST API. Click **Start REST API** if necessary. When executing from another computer, replace the URL and launcher path with the server's LAN URL and your local installation path.

For offline execution, select a saved scenario and stop the lab, then click **Export Standalone Run**. Choose an empty directory. The export contains lab.prj, its recording/configuration/MIB/trap files, scenario.json and a run script. Set the license-file path in the script before running it. Use simulator-run from an installation with Java 21 or later. This local run uses a separate workspace and writes JSON/JUnit reports.

Getting started

A recording is an existing SNMP walk XML file. A project describes agents and their recording, configuration, MIB, and trap files. A lab groups agents for automation. A workspace is a private directory that stores imported assets, lab definitions, operations, and reports. A scenario is a saved timeline (also available as JSON) that schedules actions and optionally checks observed SNMP traffic.

Use Java 21 or later. If your installation supplies a suitable runtime, use that runtime; otherwise make sure the java command selects a compatible installation. The local runner and REST server require an existing, valid, signed simulator license supplied with --license-file. The desktop retains its normal licensing workflow. Agent capacity is shared across labs in one service, including agents that reserve capacity before they are started.

Run the commands below from the simulator installation directory. The examples use a writable automation-demo folder, a separate workspace, and a separate reports folder. Replace license-file paths with your own license. Quote paths containing spaces. The management service uses TCP port 18080; the example SNMP agent uses UDP port 1161. These are separate endpoints.

On macOS/Linux, the Java classpath separator is a colon. On Windows it is a semicolon. lib/* loads the simulator and its installed dependencies; classpath/* also supports a source checkout where dependency JARs are kept separately. The direct Java commands work without the optional simulator-run and simulator-server wrapper scripts.

```
macOS/Linux: java -cp "lib/*:classpath/*" \
  com.ireasoning.simulator.cli.SimulatorMain --help

Windows PowerShell: java -cp "lib/*;classpath/*" `
  com.ireasoning.simulator.cli.SimulatorMain --help
```

Use one workspace per running process. The desktop and its embedded REST server share one workspace. A separate runner or headless service needs its own workspace. Keep recordings and configuration files private if they contain communities, passwords, or scripts. The service copies imported files into its asset store; reset does not modify your original files.

Run your first scenario

Create automation-demo/data.xml with the following recording. The communities public and private are sample values for a local test. The recording contains a description and one interface status object.

```
<?xml version="1.0" encoding="UTF-8"?>
<SnmpSimulatorData sysUpTimeOffset="0">
  <Instances interval="1" readCommunity="public"
             writeCommunity="private">
    <Instance oid=".1.3.6.1.2.1.1.1.0" valueType="OctetString">
```

```
      <Value><![CDATA[Release 1 demonstration]]></Value>
    </Instance>
    <Instance oid=".1.3.6.1.2.1.2.2.1.8.1" valueType="Integer">
      <Value><![CDATA[1]]></Value>
    </Instance>
  </Instances>
</SnmpSimulatorData>
```

Create automation-demo/lab.prj. File references in a local project are resolved relative to that project file. The local scenario runner assigns agent-0 to the first agent, agent-1 to the second, and so on.

```
<?xml version="1.0" encoding="UTF-8"?>
<Networks name="Release 1 demonstration">
  <Simulator name="Router" ipAddress="127.0.0.1"
             port="1161" initialState="up">
    <dataFile name="data.xml"/>
  </Simulator>
</Networks>
```

Create automation-demo/scenario.json. It starts the agent, changes ifOperStatus to down after one second, introduces a one-second outage, and restores the recorded value. Times and durations are in milliseconds.

```
{
  "schemaVersion": 1,
  "name": "outage-and-recovery",
  "seed": "42",
  "durationMs": 6000,
  "actions": [
    {"id":"start", "atMs":0, "kind":"start",
     "agentIds":["agent-0"]},
    {"id":"interface-down", "atMs":1000,
     "kind":"applyOverrides", "agentIds":["agent-0"],
     "values":[{"oid":".1.3.6.1.2.1.2.2.1.8.1",
                "type":"Integer", "value":"2"}]},
    {"id":"outage", "atMs":2000, "kind":"applyFault",
     "agentIds":["agent-0"],
     "fault":{"id":"temporary-outage", "kind":"outage",
              "durationMs":1000}},
    {"id":"recover", "atMs":4000, "kind":"clearOverrides",
     "agentIds":["agent-0"]}
  ],
  "assertions": []
}
```

Run the scenario. Replace /path/to/license.dat or C:\private\license.dat. Use the same workspace only after the previous process has exited.

```
macOS/Linux
java -cp "lib/*:classpath/*" \
  com.ireasoning.simulator.cli.SimulatorMain run \
  --workspace automation-demo/run-workspace \
  --license-file /path/to/license.dat \
  --project automation-demo/lab.prj \
  --scenario automation-demo/scenario.json \
  --report-dir automation-demo/reports

Windows PowerShell
java -cp "lib/*;classpath/*" `
  com.ireasoning.simulator.cli.SimulatorMain run `
  --workspace automation-demo/run-workspace `
  --license-file C:\private\license.dat `
  --project automation-demo/lab.prj `
  --scenario automation-demo/scenario.json `
  --report-dir automation-demo/reports
```

The runner resets the lab before execution and stops its agents when the scenario finishes. It writes report.json and report.xml, a JUnit report, to the report directory. The first example has no assertions: success means the scenario completed without a reported operational failure. It does not prove that an SNMP manager detected the outage.

To observe the changes, configure your SNMP manager to poll 127.0.0.1:1161 using SNMPv2c and the public community. Query .1.3.6.1.2.1.2.2.1.8.1 frequently enough to see the short phases. Expect value 1 initially, value 2 after one second, no responses during the outage, value 2 after the outage, and value 1 after recovery. Allow for request timing and manager retries.

For a traffic check, replace the empty assertions array with the example below. Start the external poller before running the scenario and keep it polling throughout. The simulator observes traffic but does not create the manager requests for you.

```
"assertions": [
  {"kind":"requestCount", "agentId":"agent-0",
   "oid":".1.3.6.1.2.1.2.2.1.8.1",
   "fromMs":0, "toMs":2000, "min":1},
  {"kind":"outagePacketCount", "agentId":"agent-0",
   "fromMs":2000, "toMs":3000, "min":1},
  {"kind":"requestCount", "agentId":"agent-0",
   "fromMs":4000, "toMs":6000, "min":1}
]
```

Exit code	Meaning
0	The run passed.
1	An assertion failed.
2	Setup, execution, or evidence was incomplete or failed.
130	The run was cancelled or interrupted.

In a shell, inspect $? immediately after the command. In PowerShell, inspect $LASTEXITCODE. Check the report state as well as the exit code when integrating with a test pipeline.

Control agent state

Action	Effect
start	Starts stopped agents; also resumes paused agents.
stop	Stops the selected agents. Stopping the whole lab cancels its running scenario.
pause	Freezes selected agent clocks, playback, thresholds, and scheduled traps. Lab pause also freezes the scenario timeline.
resume	Continues from the paused point without catching up the elapsed wall time.
reset	Ends a running scenario, clears overrides and faults, restores the imported baseline, and leaves agents stopped.

While paused, UDP listeners remain open and GET requests return frozen values. SET requests and management changes to agent state are rejected. Existing fault conditions remain active, so an agent paused during an outage still suppresses

packets. Stop and reset retain reports; deleting a lab removes its stored reports. Export reports before deletion.

In the desktop, right-click an agent and choose Pause Agent. Choose Start Agent to resume it. Stop Agent stops it. The scenario, override, fault, and report controls described below use the automation interfaces; they are not additional desktop menu items.

For REST lifecycle commands, use scope="lab" to apply an action to the whole lab. Use scope="agents" with agentIds to select agents. API-created labs initially contain stopped agents; their project initialState does not start them. The existing runSimulator project launcher continues to honor initialState.

```
{"action":"pause", "scope":"agents",
 "agentIds":["agent-0"]}

{"action":"reset", "scope":"lab"}
```

Override recorded values

An override changes the value returned by an existing recorded OID while keeping the recording unchanged. Its type must match the recorded SNMP type. Overrides cannot create an object absent from the recording. An active override takes precedence over an SNMP SET to the same object.

```
{"dataContextId":"context-0", "values":[
  {"oid":".1.3.6.1.2.1.2.2.1.8.1",
   "type":"Integer", "value":"2"}
]}
```

For an agent with multiple recordings, specify dataContextId: context-0 identifies the first recording, context-1 the second, and so on. These automation IDs are different from community names and SNMPv3 context names. You may omit the context only when the agent has one recording. A PUT replaces the complete override set for that context. Send an empty values array, or DELETE the overrides resource, to clear it.

Type	JSON value encoding
Integer or Integer32	Signed 32-bit decimal string, for example "-1".
Counter32, Gauge32 or Unsigned32, TimeTicks	Unsigned 32-bit decimal string.
Counter64	Unsigned 64-bit decimal string. Keep it a string to preserve large values.
OctetString	Base64-encoded bytes. For example, "Um91dGVy" represents Router.
OID	Dotted numeric OID string.
IpAddress	IP address string.
Null	Use type "Null"; no payload value is required.

Automation type names are case-sensitive. The automation names Gauge32 and Unsigned32 differ from some legacy XML valueType names. Use the names in this table for JSON overrides.

Simulate faults and notifications

Apply faults to a running agent. Each rule has a stable id and a kind. Replacing the REST rules array replaces all fault rules for that agent and restarts their random streams. An empty rules array clears them. In a scenario, applyFault adds or replaces its named rule and clearFault removes the rule named by faultId.

Kind	Important fields and behavior
delay	delayMs, optionally maximumDelayMs. Adds one fixed or random-range delay to a response PDU.
loss	probability from 0 to 1; burstLength defaults to 1. Discards responses after request evaluation.
error	errorStatus and errorIndex; optional probability, oidPrefix, and numeric pduTypes filters. Returns a modeled SNMP error.
outage	Positive durationMs. Discards incoming packets before decoding; suppressTraps optionally suppresses notifications.
reboot	Positive durationMs. Stops transport, clears overrides, and restarts from baseline after the interval.

```
{"rules":[
  {"id":"slow-response", "kind":"delay", "delayMs":100},
  {"id":"intermittent-loss", "kind":"loss",
   "probability":0.1, "burstLength":3}
]}
```

durationMs is optional for delay, loss, and error rules; zero means no automatic expiry. Outage and reboot require a positive duration. Delay and duration values cannot exceed seven days. Rules are evaluated in outage/reboot, loss, error, then delay order. A loss or error decision ends the evaluation of that response. A probability of 0.1 models a 10 percent chance; it does not promise exactly 10 losses out of 100 requests.

Use the same scenario seed, agent IDs, rule IDs, and ordered traffic to reproduce modeled random decisions. Operating-system scheduling, external manager timing, manager retries, scripts that use their own clocks or random generators, and SNMPv3 security bytes can still differ between runs.

To send a named trap or INFORM, define the notification and its destinations in the existing recording/configuration format described later in this guide. Use sendTrap in a scenario or POST to the agent traps resource with {"trapName":"trap1"}. The name must exist in the recording. A locally transmitted UDP trap does not prove the receiver received it; an INFORM acknowledgement provides a stronger delivery check.

Write scenarios and assertions

A scenario uses schemaVersion 1, a descriptive name, a decimal-string seed, a positive durationMs, an actions array, and an assertions array. Each action needs a unique id, atMs within the scenario duration, a supported kind, and at least one valid agent ID. Equal-time actions retain document order; agents selected by an action execute in sorted ID order.

Action kind	Additional fields
start or stop	agentIds

applyOverrides	values; optional dataContextId
clearOverrides	Optional dataContextId
applyFault	fault object
clearFault	faultId
sendTrap	trapName

A failed action fails the run and stops its agents. Pause and resume are external lifecycle controls, not scenario action kinds. Reset ends the scenario. Before starting another scenario, leave the lab stopped; every scenario starts from a reset baseline.

Assertions select an agentId and count matching events during [fromMs,toMs): the start time is included and the end time is excluded. min sets the minimum count; optional max sets the maximum. Omit the time fields to cover the run. Specify dataContextId when you need to distinguish recordings.

Assertion kind	What it measures
requestCount	Authorized, decoded requests; optional oid, numeric pduType, and context filters.
outagePacketCount	Packets discarded during an outage, including discovery traffic. OID and PDU filters do not apply to undecoded packets.
setValue	Incoming SET requests matching an expected typed value. Does not prove that the SET was applied.
responseErrorCount	Transmitted responses; optional errorStatus filter.
trapAttemptCount	Notification attempts.
trapSentCount	Local UDP notification transmissions. Does not establish receiver delivery.
informAckCount	Successful INFORM responses.

For setValue, provide expected using the same oid/type/value encoding as an override. For a successful read-value check, use your SNMP manager or test client to verify the returned value; requestCount checks traffic, not returned object values.

Start the REST service

Open **Tools > REST API** or click REST API in the Labs toolbar. Enter a bind address and port, then click **Start**. The desktop defaults to 0.0.0.0 and TCP port 18080. Local clients connect to http://127.0.0.1:18080; LAN clients use the simulator computer's LAN IP address. No private token or Authorization header is required.

Select **Start with application** to remember automatic startup. Click Stop to stop REST without stopping the agents. The desktop and its embedded Java web server share the same labs, saved scenarios and results.

For a separate headless service, run the following from the installation directory. Supply your signed simulator license and a separate writable workspace. Headless mode defaults to loopback unless --bind is specified. HTTPS is optional.

```
bin/simulator-server --workspace automation-demo/api-workspace \
  --license-file /absolute/path/to/license.dat --bind 0.0.0.0 --port
18080

curl http://127.0.0.1:18080/health/ready
```

On Windows, use bin/simulator-server.cmd. The direct Java entry point is com.ireasoning.simulator.cli.SimulatorMain server, using the same options. The management API is under /api/v1.

Create and control a lab through REST

Upload data.xml as raw bytes. The response contains an id that identifies the stored recording. Copy that id into the lab definition; do not substitute the local file path.

```
curl --data-binary \
  @automation-demo/data.xml http://127.0.0.1:18080/api/v1/assets
```

Save automation-demo/lab.json using the returned id in place of RECORDING_ID. Keep agent-0 as the agent ID to reuse the scenario from this chapter.

```
{"schemaVersion":1, "name":"Release 1 demonstration",
 "agents":[{
   "id":"agent-0", "name":"Router",
   "endpoint":{"address":"127.0.0.1", "port":1161},
   "dataAssetIds":["RECORDING_ID"]
 }]}
curl --header \
  "Content-Type: application/json" --header \
  "Idempotency-Key: create-demo-1" --data-binary \
  @automation-demo/lab.json http://127.0.0.1:18080/api/v1/labs
```

Lab creation returns HTTP 202 and an operation id, not the completed lab. Poll the operation, replacing OPERATION_ID below. Wait for status SUCCEEDED and obtain result.id as LAB_ID. FAILED and CANCELLED are terminal states; inspect errorCode on failure.

```
curl \
  http://127.0.0.1:18080/api/v1/operations/OPERATION_ID
```

Save automation-demo/start.json as {"action":"start","scope":"lab"}. Submit the lifecycle command, then poll its operation before polling the SNMP endpoint.

```
curl --header \
  "Content-Type: application/json" --data-binary \
  @automation-demo/start.json \
  http://127.0.0.1:18080/api/v1/labs/LAB_ID/actions
```

Change the file action to pause, resume, stop, or reset for the other lifecycle commands. To run scenario.json, first stop the whole lab and wait for that operation to complete, then submit the scenario.

```
curl --header \
  "Content-Type: application/json" --data-binary \
  @automation-demo/scenario.json \
  http://127.0.0.1:18080/api/v1/labs/LAB_ID/runs
```

Poll the returned operation to completion. Its result includes runId; use this as RUN_ID to retrieve the report. An operation can succeed while its scenario report records an assertion failure, so inspect both operation status and report state.

```
curl --output \
  automation-demo/api-report.json \
  http://127.0.0.1:18080/api/v1/labs/LAB_ID/runs/RUN_ID/report
```

```
curl --output \
  automation-demo/api-report.xml \
  "http://127.0.0.1:18080/api/v1/labs/LAB_ID/runs/RUN_ID/report?
format=junit"
```

After exporting reports, DELETE the lab and poll the resulting operation. Stop the server with Ctrl+C when finished.

```
curl --request DELETE --header \
  @automation-demo/private/auth-header.txt \
  http://127.0.0.1:18080/api/v1/labs/LAB_ID
```

If you retry a mutation after losing its response, reuse the same Idempotency-Key and the same request body on the same route. A different body with the same key returns a conflict. Do not reuse a key for a new command. To request cancellation, POST an empty JSON object to /api/v1/operations/OPERATION_ID/cancel and poll that cancellation operation. A short command may finish before cancellation takes effect; cancellation does not undo a trap already sent.

Resource under /api/v1	Methods and purpose
/assets	POST: upload a recording or other project resource.
/labs and /labs/{labId}	GET: inspect; POST /labs: create; DELETE a lab: remove it and its reports.
/labs/{labId}/actions	POST: lifecycle action.
/labs/{labId}/agents	GET: list; POST: add to a stopped lab. DELETE /agents/{agentId}: remove from a stopped lab.
/labs/{labId}/agents/{agentId}/overrides	PUT: replace typed values; DELETE: clear the selected context.
/labs/{labId}/agents/{agentId}/faults	PUT: replace the rules array.
/labs/{labId}/agents/{agentId}/traps	POST: send trapName.
/labs/{labId}/runs	POST: start a scenario; GET /runs/{runId}: inspect its report.
/labs/{labId}/events	GET: read events using after and limit query parameters.
/operations/{operationId}	GET: poll completion; POST /cancel: request cancellation.

To import an existing project through REST, upload the project and every referenced file, then POST /labs with projectAssetId and a resources object mapping each exact project reference to its uploaded asset ID. The server does not interpret a project reference as a path on its own filesystem. See the API reference openapi.json, when supplied with your distribution, for the complete interface.

Automate the workflow with Python

The standard-library client requires Python 3.9 or later. Place ireasoning_simulator.py in the same folder as your test script, or add the

distribution python folder to PYTHONPATH. In a source checkout the client is in simulator/python. No pip package is needed for the client itself.

Save automation-demo/run_api.py with the following code. Run it from the installation directory while the REST server is running. The example uses the no-assertions scenario; if you add traffic assertions, run your manager at the same time.

```
import json
from pathlib import Path
from ireasoning_simulator import SimulatorClient

root = Path("automation-demo")
client = SimulatorClient("http://127.0.0.1:18080")
asset_id = client.upload(root / "data.xml")
definition = {
    "schemaVersion": 1, "name": "Python demonstration",
    "agents": [{
        "id": "agent-0", "name": "Router",
        "endpoint": {"address": "127.0.0.1", "port": 1161},
        "dataAssetIds": [asset_id]
    }]
}
scenario = json.loads((root / "scenario.json").read_text())
with client.lab(definition) as lab:
    report = client.run(lab["id"], scenario, timeout=120)
    run_id = report["id"]
    (root / "python-report.json").write_text(
        json.dumps(report, indent=2), encoding="utf-8")
    (root / "python-report.xml").write_text(
        client.report(lab["id"], run_id, format="junit"),
        encoding="utf-8")
    if report["state"] != "SUCCEEDED":
        raise RuntimeError("Scenario result: " + report["state"])
```

The lab context manager deletes the lab on exit, including on an exception, so export reports inside the with block. The client waits for asynchronous operations, but run returns the terminal report even when its assertions fail. Always check report["state"].

Use client.actions(lab_id, "start"), client.overrides(lab_id, agent_id, values), client.faults(lab_id, agent_id, rules), and client.trap(lab_id, agent_id, name) for individual operations. Provide context_id to overrides for multiple recordings. SimulatorError exposes code and an optional HTTP status without printing credentials. The client does not automatically retry mutations after a transport failure.

Operate the simulator with Claude Code

The optional ireasoning-simulator skill lets you operate SNMP Agent Simulator using natural language in Claude Code. You can search the device library, create agents, control labs, save and run scenarios, and explain or export run results. It uses the simulator REST API, so the same labs and results are visible in the desktop application.

Requirements

Install and sign in to a current Claude Code release that supports skills. An active Claude account with available usage is required. Install Python 3.9 or later on the computer running Claude Code. Use python3 on macOS or Linux, or py -3 on Windows. The skill uses only the Python standard library; no additional Python packages are required.

Use a simulator distribution containing skills/ireasoning-simulator or ireasoning-simulator-skill.zip. The simulator and its REST service must be running with the usual simulator license. This integration targets Claude Code terminal sessions; it does not provide an embedded chat window or a Claude Desktop MCP connection.

Install the skill

For personal use, place the ireasoning-simulator folder inside your user account's .claude/skills directory. On Windows this is %USERPROFILE% \.claude\skills; on macOS or Linux it is ~/.claude/skills. If using the ZIP, extract it into that skills directory. The final layout must contain .claude/skills/ireasoning-simulator/SKILL.md, with scripts and references alongside it, rather than a second nested ireasoning-simulator folder.

Alternatively, copy the supplied folder. Open a terminal in the simulator distribution directory, which contains the skills folder, and run the commands for your platform.

macOS or Linux:

```
mkdir -p ~/.claude/skills
cp -R skills/ireasoning-simulator ~/.claude/skills/
```

Windows PowerShell:

```
New-Item -ItemType Directory -Force "$HOME\.claude\skills"
Copy-Item -Recurse .\skills\ireasoning-simulator `
  "$HOME\.claude\skills\"
```

For a skill used only in one project, install the same folder under that project's .claude/skills directory instead. Inspect or back up an existing copy before replacing it. Start a new Claude Code session after installing the skill, or use the skill reload command supported by your Claude Code release.

Connect to the simulator

In SNMP Agent Simulator, choose Tools > Start REST API Service. Check the bind address and port, then click Start. For Claude Code running on the same computer, the skill connects to http://127.0.0.1:18080 by default. Starting the REST service does not start the lab agents.

For a simulator on another computer, start the REST service on a reachable interface and tell Claude its explicit service URL, for example http://192.168.1.20:18080. Use your existing network and HTTPS configuration as appropriate. The skill does not change server settings or switch to another endpoint if the specified service is unavailable.

Make a request in Claude Code

Invoke the skill explicitly with /ireasoning-simulator followed by your request. Claude Code can also select it when you ask naturally about iReasoning Simulator. Begin by listing the available labs so that Claude can resolve the real lab and agent IDs.

```
/ireasoning-simulator Show my labs and their agent states.
```

Example requests:

- In Sample Simulators, show the agents and start only cisco.
- Create a new lab with ten Cisco Catalyst 9300 agents from the device library on 127.0.0.1, using ports 1161 through 1170. Leave them stopped.
- In this lab, create a 30-second scenario that starts the selected agent, applies a five-second outage at 10 seconds, and checks for requests after recovery. Save the scenario before running it.
- Send one linkDown trap to 127.0.0.1 port 1162 at three seconds into the scenario. Leave the agent's normal trap recording behavior unchanged.
- Explain why the selected run failed, including failed checks and observed counts, and export its JUnit report to my chosen directory.

Specify the lab, agent names, timing and destination when relevant. If names match more than one lab or agent, choose the exact ID when Claude asks. Device-library requests use the currently installed catalog; an entry with a missing recording cannot be instantiated. Several agents on one IP address with different UDP ports are separate endpoints, not separate discoverable IP addresses.

The skill uses the existing simulator operations and their state requirements. Structural changes require a stopped lab. Starting a scenario resets its baseline and stops its agents when the run finishes. Check these effects when asking Claude to run a scenario on an existing lab. Claude Code's normal tool permissions still apply.

Understand progress and results

REST operations run asynchronously. The helper normally waits for completion and reports operation and run IDs. For a long operation, ask Claude to submit it without waiting and check its operation ID later. A timeout or interrupted command does not cancel the server operation. Ask Claude to inspect or resume that operation rather than submitting the same action again with a new idempotency key, especially when sending notifications.

A SUCCEEDED run without checks means its actions completed; it does not prove that a network manager received traffic or recovered correctly. Checks need real observations from your SNMP manager, poller or receiver. Sending a UDP trap only establishes local transmission; an INFORM acknowledgement provides receiver evidence. Reports identify incomplete evidence and dropped display events. Ask Claude to export the full report when the event history is large.

Agent trap recordings, threshold rules and scripts continue to send their normal notifications independently of scenario Send trap / INFORM actions. The skill can explain available reports and events; it does not provide CPU or thread profiling or undo already transmitted notifications.

Troubleshooting and removal

If Claude cannot connect, check that the REST service is started and that the selected host and port are correct. If the skill is not found, verify the SKILL.md location and start a new Claude Code session. If Python is missing, install Python 3.9 or later and check python3 --version or py -3 --version. Claude account or credit errors must be resolved in Claude; restarting the simulator does not resolve them.

An agent that cannot start may have an endpoint conflict, a license capacity limit or a recording that requires trusted scripts. A headless service disallows recording scripts unless explicitly started with --trusted-scripts. The skill does not bypass this policy. Keep actual communities and SNMPv3 passwords in private input files; do not paste them into chat or request that they be printed.

To update the skill, back up any local customizations and replace its installed folder with the new distribution copy. To uninstall it, remove only the installed ireasoning-simulator skill folder. This leaves the simulator, labs, scenarios and reports unchanged.

Read reports and diagnose problems

Report state	Meaning
RUNNING	The scenario has not finished.
SUCCEEDED	The run completed and all defined assertions passed.
FAILED	An assertion did not meet its count bounds.
ERROR	An operational failure prevented normal completion.
CANCELLED	The run was cancelled.
INCOMPLETE	Required runtime evidence was lost or could not be collected reliably.

Reports contain the seed, baseline asset references, runtime version, assertion counts, and error classification. They do not include packet payloads or configured expected SET values. JUnit XML distinguishes failed assertions from execution errors, so a test pipeline can report both correctly. Upload report.xml as a JUnit test result after a local run.

The event list retains up to 10,000 display events. Read pages with after set to the previous nextCursor and limit between 1 and 1,000. droppedEvents reports retention loss. Assertions are counted as events arrive before old display events are evicted, so retention loss alone does not erase previously counted assertions. Runtime saturation or missing evidence must not be treated as a passing test.

Reopening a workspace recovers stored lab definitions and reports but leaves agents stopped. Interrupted operations are marked failed. Start agents explicitly after a server restart. SNMPv3 engine boots persist and increase across restart and reset; reset restores simulation values without reusing old SNMPv3 engine state.

Problem	What to check
Java reports an unsupported class version	Select a Java runtime compatible with your installed JARs; Release 1 requires at least Java 21.

Problem	What to check
LICENSE_INVALID or LICENSE_CAPACITY	Use a valid signed simulator license and stay within its agent count. Delete unused labs to release reservations.
Workspace cannot be opened	Check write access and whether another process holds the workspace lock. Use a separate workspace per process.
Agent cannot start or SNMP times out	Check the bind address, occupied UDP port, community/context, agent state, firewall, and active loss/outage rules.
HTTP 409	Check lifecycle state and idempotency-key reuse. Stop the lab before scenario execution.
HTTP 400 or 422	Check JSON syntax, field names, IDs, value types, and scenario times. Inspect the error code.
HTTP 413	Management JSON is limited to 1 MiB; an uploaded asset is limited to 64 MiB.
HTTP 503	Check license capacity, service readiness, and whether shutdown is underway.
An accepted operation later fails	Poll its operation resource and inspect errorCode; HTTP 202 means acceptance, not completion.
A traffic assertion fails	Check polling intervals and time windows. Requests during an outage count as outage packets, not decoded requests.

Credentials and remote deployment

The local runner and server accept --credentials-file private.json. This file maps a credentialRef in the lab definition to a native configuration XML overlay. It replaces credentials in memory without editing the source asset. Protect the file and never place real credentials in scripts or reports.

```
{"test-device":"<SnmpAgent><context id=\"context-0\"
readCommunity=\"test-read\"
writeCommunity=\"test-write\"/></SnmpAgent>"}
```

The example credentialRef is test-device. For multiple recordings, supply every required context. An empty writeCommunity disables SET in that context. SNMPv3 overlays use the existing user, group, view, and optional properties elements described in the configuration chapter. Engine boots remain managed by the service.

Headless automation rejects recordings containing BeanShell scripts unless you explicitly add --trusted-scripts. Enable this only for recordings you trust: scripts execute with the service account privileges and can run external or blocking commands. Desktop scripts retain their existing behavior.

LAN management can use HTTP with --bind and the service interface address. Optional HTTPS uses --tls-keystore with a PKCS12 keystore and --tls-password-file with its password file. For HTTPS, connect with https and validate the certificate.

If you have a packaged Release 1 container image, mount a writable workspace at /work and mount license and credential files read-only under /run/secrets. The supplied image uses UID/GID 10001, so make the workspace writable by that account. Its default management listener is container loopback. To accept other containers, bind management explicitly with --bind 0.0.0.0 (and optional HTTPS).

SNMP agent addresses must also bind to 0.0.0.0 or a container interface for traffic arriving from outside the container. Management uses TCP; SNMP uses UDP. Publish only the ports you need. Twenty ports on one container address represent twenty SNMP endpoints, not twenty separately discoverable IP addresses. Keep the management endpoint private unless protected by HTTPS.

For normal desktop use, continue with the graphical-interface instructions earlier in this guide. For XML scripts, recording formats, and SNMPv3 configuration details, see Helper Functions, SNMP Walk Data File, SNMP Trap Data File, and Simulator Configuration.

Helper Functions

You can use helper functions in scripts within the SNMP walk data file.

- **long getSysUpTime()**

Returns the simulator uptime in centiseconds (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)**

Returns the number of MIB instances under the specified subtree.

Parameters:

subTreeOID: 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)**

Pauses the current simulator for the specified number of milliseconds.

Parameters:

milliseconds: pause duration, in milliseconds

Usage example:

```
sleep(1000);//sleep for 1 second
```

- **VarBind createVarBind(String oid, String type, String value)**

Creates an SNMP variable binding.

Parameters:

oid:	object identifier of the variable binding
value:	value of the variable binding, as a string
type:	data type of the value

The available 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"
// with an integer value of 5
VarBind varbind1 = createVarBind(".1.3.6.1.2.1.1.1.5", "integer", "5");
```

- **void sendV1Trap(int generic, int specific, VarBind varbinds)**

Sends an SNMPv1 trap.

Parameters:

generic:	value of the generic field
specific:	value of the 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", "51 211"));
```

- **void sendV2Trap(String snmpTrapOid, VarBind varbinds)**

Sends an SNMPv2 trap.

Parameters:
snmpTrapOid: OID value of the 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",
  "51 211"));
```

- **void sendTrap(String trapName)**

Sends an SNMPv1 or SNMPv2 trap to all destinations defined in the simulator configuration file.

Parameters:
trapName: name of a trap node in the data XML file

Usage example:

```
sendTrap("trap1");
```

- **int atoi(String stringValue)**

Converts a string to an integer.

Parameters:
stringValue: string to convert to an integer

Usage example:

```
int i = atoi("511");
```

- **String itoa(int i)**

Converts an integer to a string.

Parameters:
i: integer to convert to a string

Usage example:

```
String s = itoa(511);
```

- **int random(int uppperbound)**

  Returns a uniformly distributed pseudorandom integer from 0 (inclusive) to uppperbound (exclusive).

  Parameters:

  uppperbound: exclusive upper bound; must be positive

  Usage example:

  i = 1000 + random(500);

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

  Returns a uniformly distributed pseudorandom integer from lowerbound (inclusive) to uppperbound (exclusive).

  Parameters:

  lowerbound: inclusive lower bound; must be positive
  uppperbound: exclusive upper bound; must be positive

  Usage example:

  i = 1000 + random(100, 500);

- **previousValue(initialValue)**

  Returns the previous value of a MIB instance: the value returned for the preceding SNMP request.

  Parameters:

initialValue: initial value of the 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 from one or more SNMP walks of a single agent.

A simulator can use multiple SNMP walk data files. Each file must have a different community name (community-string indexing); otherwise, only the first file is used.

▪ Root Node

<SnmpSimulatorData
sysUpTimeOffset="111 111 111"
delayRange="50-200"
>

sysUpTimeOffset:	offset applied to sysUpTime, in milliseconds. The value returned by getSysUpTime() is shifted by this amount.
delayRange:	range of response delays, in milliseconds. For example, delayRange="50-200" selects a random delay between 50 and 200 milliseconds. If this attribute is absent, no intentional delay is added.

▪ Instances Section

The Instances section stores MIB walk instances. Each Instance node represents a MIB node.

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

interval:	time between SNMP walks, in seconds. After each interval, the simulator advances to the next set of walk data. If this attribute is absent, only the first set is used.
readCommunity:	SNMP community name for read operations, such as GET, GET_NEXT, and GET_BULK.
writeCommunity:	SNMP community name for SET operations.

Instance Node

Each Instance node represents a MIB instance.

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	Data type of this node. Possible values are: {OctetString, Null , OID , BITS , UnsignedInteger , Counter32 , Gauge , TimeTicks , IpAddress , Opaque , Integer , Counter64 }
trapOnSet	Optional. If specified, a trap is generated for each SET request for this node. The value names the trap.
trapOnSetCount	Number of traps to send.
name	Optional. Name used by other nodes to reference this node.

Data Types

Name	Description
BITS	A set of integers in mathematical notation, such as {1, 3, 8}.
Counter32	A nonnegative 32-bit SNMP counter that increases monotonically until it reaches its maximum value.
Counter64	A nonnegative 64-bit SNMP counter that increases monotonically until it reaches its maximum value.
Gauge	A nonnegative 32-bit gauge that can increase or decrease.
Integer	A positive or negative integer. The minimum value is −2 147 483 648; the maximum value is 2 147 483 648.
IpAddress	An IP address, such as 121.12.77.126.
Null	Indicates the effective absence of a sequence element.
OctetString	An octet string. A hexadecimal string can use the format (0x[0-9A-Fa-f][0-9A-Fa-f])+, such as 0x12 0xA1 0x30 or 0x12A130.
OID	An object identifier, such as 1.3.6.1.4.6.
TimeTicks	A non-negative integer.
UnsignedInteger	Unsigned 32-bit integer.

ScriptOnSet

Optional. Runs a script when a SET request is received.

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 the values of a MIB node. A node can have one or more Value nodes, each corresponding to an SNMP walk. The Instances node specifies the interval between walks. For example, if the interval is one minute and each Instance node has three Value nodes, the simulator uses the first value when it starts, the second after one minute, and the third after two minutes. It continues to use the third value until it stops.

If an Instance node has fewer Value nodes than other Instance nodes, the simulator continues to use its last value after the earlier values are exhausted.

1. Static value

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

The CDATA node stores a static value that does not change.

2. Same value

<Value same="yes"/>

This cannot be the first Value node. It uses the same value as the preceding Value node.

3. Disabled value

<Value na="yes"/>

The simulator skips this MIB node and returns no value for SNMP requests.

4. Error simulation

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

Field	Description
errorOnSet	If specified, the response PDU uses this error status for an SNMP SET request.
errorOnGet	If specified, the response PDU uses this error status for an SNMP GET, GET_NEXT, or GET_BULK request.
CDATA	If the error condition is not met, the response has no error and returns the CDATA value.

5. Simple script

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

isSimpleScript: If set to "y" or "yes", the CDATA value is treated as a one-line script, and the expression result is returned.

6. Full script

<Value script="script2"/>

script: Names the script whose result is returned. In this example, the result of the script named *"script2"* is used as the value of this MIB node.

Scripts Section (optional)

This section contains one or more Script nodes. Each Script node defines 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>
```

Scripts in this node run when the simulator starts.

ScriptDestroy Node

Sample:

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

Scripts in this node run when the 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, used by other nodes to reference it.
depends (optional)	Names of nodes used in this script, separated by commas.

getValue function:

Object getValue(String oid, SnmpPdu pdu)

Parameters:

oid: object identifier in the SNMP request being processed

pdu: SNMP PDU being processed

Returns:

The return value must be a string or an object implementing the SnmpDataType interface, such as SnmpOctetString or SnmpOID.

▪ Traps Section (optional)

This section contains one or more Trap nodes that 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, used by other nodes to reference it.
generic	Generic-trap field; accepts a numeric or string value.
specific	Specific-trap field; accepts a 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, used by other nodes to reference it.
snmpTrapOID	Object identifier of the snmpTrapOID object.

▪ Trap's Variable Node

A Trap node can have one or more Variable nodes. Each Variable node defines a variable binding in one of the following formats:

1. Static value

The value is stored in a 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 this value.
CDATA	Value of this variable.

2. Reference

The value comes from the referenced node.

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

Uses the OID and value of the “sysDescr” node.

3. Simple script

The value is the result of the simple script in the 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 set to yes, the CDATA content is treated as a simple script.
depends	Optional. Names of nodes used in the simple script, separated by commas.
CDATA	Contains the simple script.

4. Full script

The value is the result of the script named by the 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 the script whose result is used as this variable's value.

▪ Thresholds Section (optional)

This section contains one or more Threshold nodes.

Sample:
<Thresholds interval="1">

interval: time between calls to isTimeToSendTrap() on each Threshold node, in seconds. The simulator calls *this* function periodically to check whether a threshold has been crossed.

▪ Threshold Node

This node defines the *isTimeToSendTrap()* function, which determines whether to send a trap.

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 the trap sent when isTimeToSendTrap() returns true.
trapCount:	number of traps sent when isTimeToSendTrap() returns true.
depends:	Optional. Names of nodes used in isTimeToSendTrap(), separated by commas.

isTimeToSendTrap() function:

Return true to send a trap to all destinations defined in the simulator configuration file; otherwise, return false.

Scripting Language

Scripts use BeanShell, an embeddable scripting language based on Java. BeanShell runs standard Java statements and expressions and supports scripted objects as simple method closures, similar to Perl and JavaScript. Click here for the complete BeanShell manual.
http://www.beanshell.org/manual/contents.html

SNMP Trap Data File

This file stores traps recorded from trap senders. You can edit it after recording or create it from scratch.

The simulator sends all traps in this file after it starts.

- **Root Node**

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

trapTimestampOffset: offset applied to sysUpTime.

- **Traps Section**

  This section contains one or more Trap nodes. The simulator sends each trap at the time specified by its time value. For example, a node with time="1000" is sent 10 000 milliseconds after the simulator starts. Trap nodes do not need to be ordered by time.

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; accepts a numeric or string value.
specific	SNMPv3 engineID of this agent. If empty, the first IP address resolved for the host name is used.
enterpriseOID	enterprise OID of an SNMPv1 trap.
snmpTrapOID	object identifier of the snmpTrapOID object (SNMPv2/v3 traps).
time	time to send this trap, in centiseconds (1/100 second).
count	number of traps to send.
ipAddress	IP address field in an SNMPv1 trap.
port	port number of the trap receiver.

Variable Node

Variable nodes are children of a Trap node. They define its variable bindings.

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

The simulator configuration specifies trap sink and SNMPv3 properties.

- **SNMPv3 properties Section**

The properties node defines agent properties.

version	SNMP version of this agent: 1 for SNMPv1, 2 for SNMPv2c, or 3 for SNMPv3.
engineID	SNMPv3 engineID of this agent. If empty, the first IP address resolved for the host name is used.
engineBoots	Number of agent restarts. For an SNMPv3 agent, this value is updated on each restart.

A version 3 agent rejects SNMPv1 and SNMPv2c requests for security reasons. A version 2 agent can also handle SNMPv1 requests.

TrapSink Section

This section defines trap receiver properties. The agent sends traps to all configured receivers.

- ***SNMPv1 and SNMPv2 TrapSink***

hostname	Host name or IP address of the trap receiver.
port	Port number of the trap receiver.
community	Community name of the trap receiver.
version	SNMP version of the trap receiver.
isInform	Set to "yes" to send an SNMP INFORM request instead of a trap. INFORM requests are more reliable because the receiver acknowledges them.

- ***snmpV3TrapSink***

hostname	Host name or IP address of the trap receiver.
port	Port number of the trap receiver.
isInform	Set to "yes" to send an SNMP INFORM request instead of a trap.
userName	One of the user names in the trap receiver's user list.
auth	Authentication algorithm: "MD5," "SHA," "SHA224," "SHA256," "SHA384," or "SHA512."
authPassword	Authentication password.
priv	Privacy algorithm: "DES," "AES," "AES192," or "AES256." The default is "DES."
privPassword	Privacy password.

In the snmpV3TrapSink section, authPassword and privPassword determine the security level. If both are empty, the level is noAuthNoPriv. If only privPassword is empty, the level is authNoPriv. If both are nonempty, the level is authPriv.

For example, define three trap sinks: one each for SNMPv1, SNMPv2, and SNMPv3. If the agent sends an SNMPv2 trap, it goes to all three sinks and is converted to SNMPv1 for the SNMPv1 sink. An SNMPv1 trap is converted to SNMPv2 before it is sent to the SNMPv2 and SNMPv3 sinks.

- **User Section**

This section defines the properties of authorized users.

name	User name. Each user must have a unique name.
auth	Authentication algorithm: "MD5," "SHA," "SHA224," "SHA256," "SHA384," or "SHA512."
authPassword	Authentication password.
priv	Privacy algorithm: "DES," "AES," "AES192," or "AES256."The default is "DES."
privPassword	Privacy password.
group	Group associated with this user.

A user's security level is determined by its group, not by its authPassword and privPassword values. This differs from the snmpV3TrapSink node. If the security level is authNoPriv, privPassword is ignored.

▪ Group Section

This section defines SNMPv3 group properties.

name	Group name. Each group must have a unique name.
securityLevel	Security level: noAuthNoPriv, authNoPriv, or authPriv.
match	Context match: “prefix” or “exact.”
contextPrefix	If match is “prefix,” the context must begin with contextPrefix. If match is “exact,” the context must match exactly and contextPrefix is ignored.
readView	View associated with this group for read operations, such as GET, GETNEXT, and GETBULK.
writeView	View associated with this group for write operations, such as SET.
notifyView	View associated with this group for notification operations.

The security levels are noAuthNoPriv, authNoPriv, and authPriv. noAuthNoPriv uses neither authentication nor encryption; authNoPriv uses authentication only; and authPriv uses both. All users in a group have the same security level.

View Section

This section defines SNMPv3 VACM view properties.

name	View name. Multiple views can share a name.
type	View type: "included" or "excluded." Each subtree in a MIB view is included or excluded along with all object instances it contains.
subTree	Subtree OID: a node in the MIB naming hierarchy and all of its subordinate elements.
mask	A sequence of 0s and 1s separated by periods or colons. The mask supports fine-grained access control: each bit determines whether the corresponding OID subidentifier must match. A 1 requires an exact match; a 0 is a wildcard that matches any value.

For example:

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

This defines view1, which includes all tree nodes whose OIDs begin with ".1.3."

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

This defines view1, which includes all tree nodes whose OIDs begin with ".1.3.6.1.2." The final 0 in the mask means that the corresponding subtree subidentifier can have any value.

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

This view includes all OIDs in .1.3.6.1.2.[1, 2, ...].7.*.

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

This view includes OIDs such as ".1.3.6.1.2.1.*," ".1.3.6.2.2.1.*," and ".1.3.6.3.2.2.*."

Simulator Configuration Example

Here is an example 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>
```

The properties section specifies an SNMPv3 agent. It defines a user named “newUser” in the *aGroup* group. The g*roup* maps to *view1*, which allows access to the “.1.3” subtree and all OIDs beneath it. Because group members require authPriv, “newUser” needs both authentication and privacy passwords. The authentication password is “abc12 345,” and the authentication algorithm is MD5. The privacy password is also “abc12 345,” and the default encryption algorithm is DES.

Examples

- Context indexing

  This sample simulator is in examples/context. It demonstrates context indexing: different community names or contexts return different data.

- Cisco router SNMP Agent Simulator

  This sample simulator is in examples/cisco. To run it, launch the simulator GUI and select “File/Open Project” to load examples/cisco/cisco.prj. Alternatively, run examples/runCiscoAgent.bat. The sample contains data collected from a Cisco router.

- Windows 2000 SNMP Agent Simulator

  This sample simulator is in examples/win2000. To run it, launch the simulator GUI and select “File/Open Project” to load examples/win2000/win2000.prj. Alternatively, run examples/runWin2000Agent.bat. The sample contains data collected from the default Windows 2000 SNMP agent.

- Multi-Walk Agent Simulator

  This sample simulator is in examples/multiwalk. Its SNMP walk data file contains four walks, with a 10-second interval between them. After the agent starts, MIB instance values change every 10 seconds until the last set of walk data is loaded. The sample also demonstrates error simulation, scripting, 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/1,4096,0134 377 087.html,00.html

- Java website

  http://java.sun.com/