
iReasoning SNMP API User Guide

Copyright © 2002-2006 iReasoning Inc, All Rights Reserved.

The information contained herein is the property of iReasoning Inc. This document may not be copied, reproduced, reduced to any electronic medium or machine readable form, or otherwise duplicated, and the information herein may not be used, disseminated or otherwise disclosed, except with the prior written consent of iReasoning Inc.

Table of Contents

<i>INTRODUCTION</i>	<i>1</i>
<i>About this document</i>	<i>1</i>
<i>Target Audience</i>	<i>1</i>
<i>INSTALLATION</i>	<i>2</i>
<i>Requirements</i>	<i>2</i>
<i>Installation Procedures</i>	<i>2</i>
<i>USING iREASONING SNMP LIBRARY</i>	<i>4</i>
<i>Overview and Architecture</i>	<i>4</i>
<i>UML diagram</i>	<i>7</i>
<i>Configuration</i>	<i>8</i>
<i>Logger configuration</i>	<i>8</i>
<i>Usage examples</i>	<i>8</i>
<i>SnmpSession class</i>	<i>8</i>
<i>Synchronous SNMP Operations</i>	<i>9</i>
<i>Asynchronous SNMP Operations</i>	<i>10</i>
<i>SNMP Table Retrieval</i>	<i>11</i>
<i>SnmpTrapdSession class</i>	<i>12</i>
<i>SnmpTrapSender class</i>	<i>13</i>
<i>SnmpPoller class</i>	<i>13</i>
<i>Java Applet Example</i>	<i>14</i>
<i>FAQ</i>	<i>15</i>
<i>RESOURCES AND REFERENCE</i>	<i>19</i>
<i>GLOSSARY OF TERMS</i>	<i>20</i>

INTRODUCTION

About this document

The purpose of this document is to provide the reader with enough knowledge to start developing software using the iReasoning's SNMP toolkit. Some examples are provided to better illustrate the usage of APIs. This document is not intended to provide detailed information about the SNMP APIs. To effectively use the APIs, readers need to refer to javadoc help that has detailed instructions and examples.

The iReasoning SNMP API is a Java toolset that greatly simplifies the development of applications for managing SNMP agents. It provides full support for SNMPv1, SNMPv2c and SNMPv3. The design and implementation of this library were based on object-oriented methodology and followed recognized design patterns. These advantages, combined with a highly optimized code base, make iReasoning SNMP library stand out from the competition.

Target Audience

This document is intended for users and engineers who are responsible for developing SNMP based management applications. A user of this software does not have to be an expert in network management field. He or she should, however, be familiar with basic SNMP concepts, such as SNMP GET and GET_NEXT operations. Some basic knowledge about Java language is required to understand examples. This software is designed to minimize the learning curve for users and achieve a given programming job with less code. A new user should therefore find it quite easy to learn and master this product.

INSTALLATION

Requirements

- JDK1.2 or a later version must be installed. You can download JDK from the SUN's web site (<http://java.sun.com/j2se/>)
- At least 32MB memory is required

Installation Procedures

- Download and unzip

Download iReasoning SNMP library and unzip it to the desired directory, for example, C:\lib\iReasoning\snmp

The directory structure will look like this:

<i>Directory Name</i>	<i>Description</i>
<i>lib</i>	contains binary jar files
<i>javadoc</i>	contains javadoc HTML files
<i>examples</i>	contains examples source code
<i>config</i>	configuration files

➤ Set up CLASSPATH

`ireasoningsnmp.jar` needs to be added to CLASSPATH environment variable. On Windows:

If the iReasoning SNMP library is installed at `C:\lib\iReasoning\snmp`, use the command

```
set CLASSPATH=C:\lib\iReasoning\snmp\lib\ireasoningsnmp.jar;%CLASSPATH%
```

On Unix/Linux:

If iReasoning SNMP is installed at `/usr/local/ireasoning/snmp`, use the command

```
CLASSPATH=/usr/local/ireasoning/snmp/lib/ireasoningsnmp.jar ; export CLASSPATH
```

➤ Run example program (Optional)

Example code is located at `examples/snmp` directory. They were pre-compiled and jarred into `lib/examples.jar`. To verify that everything is working, you can follow the following steps to run the example programs.

- 1) Add `examples.jar` to CLASSPATH. Take similar step, for instance, On Windows:

```
set CLASSPATH=C:\lib\iReasoning\snmp\lib\examples.jar;%CLASSPATH%
```

(assuming `examples.jar` is located at `C:\lib\iReasoning\snmp\lib\`)

On Unix/Linux

```
CLASSPATH /usr/local/ireasoning/snmp/lib/examples.jar ; export CLASSPATH
```

(assuming `examples.jar` is located at `/usr/local/ireasoning/snmp/lib/`)

- 2) Run

```
java snmpgetnext hostname .1.3
```

(Retrieve the first SNMP variable)

where `hostname` is the host name or IP address of an snmp agent.

If you do not have an snmp agent, you can run `snmpd.exe` (on Windows) to start an agent listening on port 161.

Or you can use `runexample.bat` script to run examples on Windows.

USING IREASONING SNMP LIBRARY

Overview and Architecture

The iReasoning SNMP library is designed and implemented using object oriented methodology and recognized design patterns. One of its design goals was to minimize the learning curve of SNMP library for developers. Users only need to know a few classes, most of complex works are hidden from them.

All versions of SNMP (SNMPv1, SNMPv2c and SNMPv3) are supported in one library, and users do not have to learn the detailed underlying differences between three SNMP versions. Users can always use a unified interface to access SNMP agents speaking different SNMP protocols. The example code shipped with this product clearly demonstrates the simplicity of this library. In those code, at first glance, users will realize how simple it is to write code to communicate with SNMP agents.

The central class is the `SmpSession`. It provides all the necessary mechanism to use SNMP operations. A session represents a communication channel between an SNMP manager and an agent. To communicate with multiple agents, multiple sessions have to be created. Each session is corresponding to each communication channel. A session needs to be created and then a connection needs to be established with the agent. After session is initialized, you are ready to issue SNMP commands against the agent. There are four basic SNMP operations: GET, GET_NEXT, SET, GET_BULK (SNMPv2c and SNMPv3). They are represented in `SmpSession` class as `snmpGetRequest`, `snmpGetNextRequest`, `snmpSetRequest`, `snmpGetBulkRequest` methods. High-level SNMP operations, such as GET_TABLE and WALK, are represented as `snmpGetTable` and `snmpWalk` methods. When a session is no longer needed, it needs to be closed to avoid resource leak.

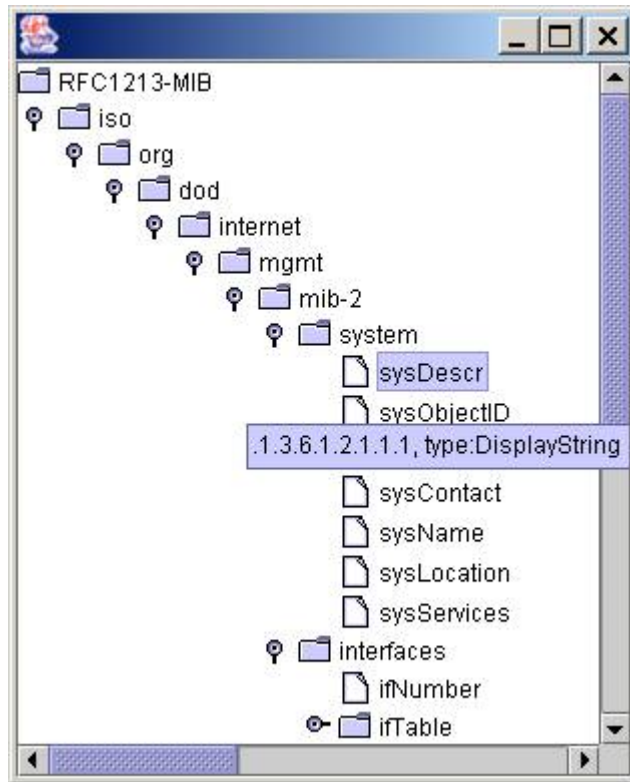
`SnmptrapdSession` class is provided for creating trap receiver programs. A trap session can be created and then waits for traps sent by agents. This session is able to understand all the SNMPv1, SNMPv2c and SNMPv3 traps.

Both synchronous and asynchronous requests are supported in `SnmptSession`. When sending synchronous requests, the session can only send one request at a time. It must block and wait for the reply to the request before issuing next one. This mode is easier to understand and implement. When using asynchronous requests, the session does not need to wait for each reply. In this case, a session has a thread handling the replies and notifying callers when replies arrive.

SNMP data types are represented as child classes of `SnmptDataType` class. The remote agent is represented as the `SnmptTarget` class. Each agent is represented by a single `SnmptTarget` object. An `SnmptTarget` can be instantiated with the IP address (or host name) and port number of the remote agent. Other properties of agent, such as community name, can also be specified. The class diagram is shown in figure 1. For detailed information about classes, you can refer to the javadoc files shipped with this library.

The SNMP protocol uses UDP as the default transport layer protocol. UDP is connectionless and the delivery of packets is not guaranteed, but it is an efficient way to transport small amount of data. The iReasoning SNMP API architecture also supports TCP besides UDP. SNMP over TCP is a good fit for transferring large amount of data. The built-in features of TCP, such as flow control and efficient segmentation, make TCP perform better than UDP in this case. The agent has to be able to support SNMP over TCP so that the SNMP manager can choose TCP as an underlying transport layer protocol.

A MIB parser utility is also included in the library. MibUtil class can be used to parse MIBs and resolve OID and variable binding's value. It can look up MIB node name from OID value as well. MibUtil's parseMib method can parse MIB and return a tree data structure. Here is an output from mibparser.java sample code, which illustrates how to use MIB parser.



UML diagram

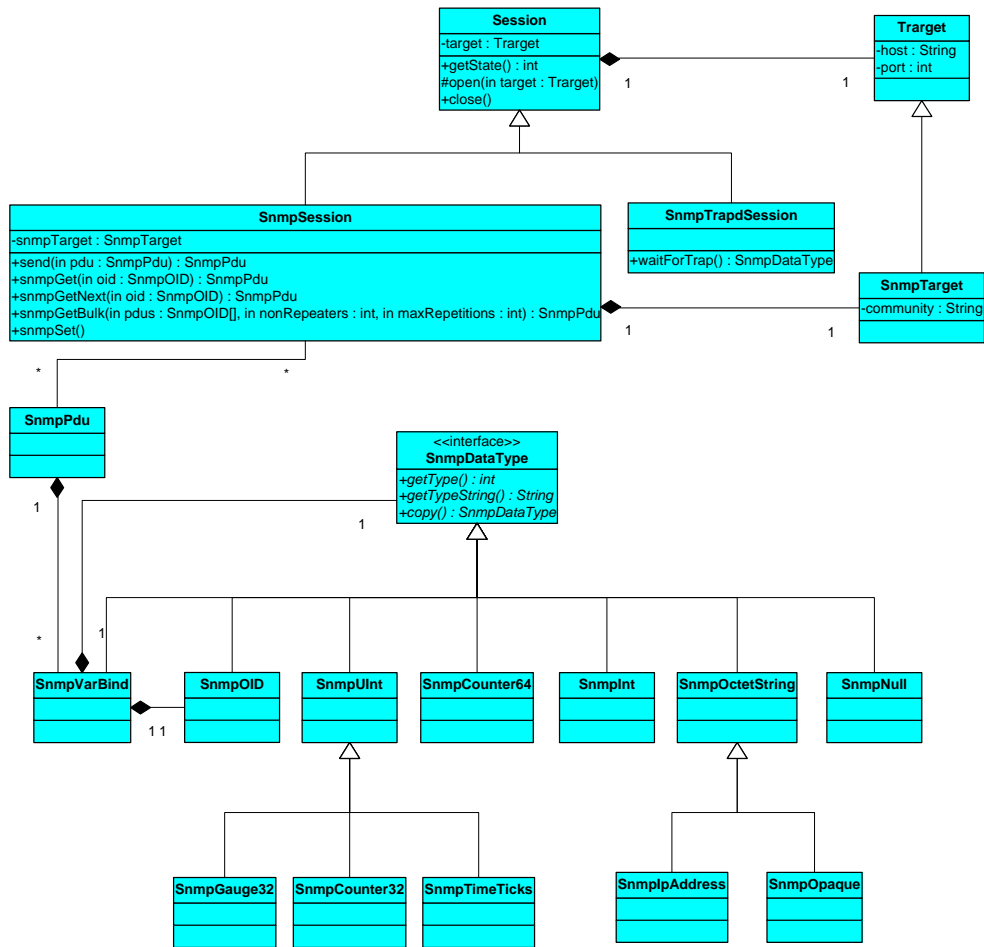


Figure 1. iReasoning SNMP library class diagram

Configuration

Config files are located at `./config` directory by default. You can add a java environment variable to set a new config directory. For example:

```
java -Dcom.ireasoning.configDir=d:\config ...
```

In this example “`d:\config`” is the new config directory.

Logger configuration

The iReasoning SNMP library has a built-in logger. The logger is configurable through the `Logger.prop` file located at `./config` directory. You can change the logging level or just disable the logger. The output of logging message is also configurable. It can be system's standard out or file. Check out `Logger.prop` for detailed information.

The popular open source `log4j` package is also supported. To switch to `log4j` logger, you just need to add one line of code in the beginning of your program:

```
com.ireasoning.util.Logger.setUseLog4j(true);
```

If using `log4j`, the `Logger.prop` will no longer be used. You need to configure the `log4j` logger or provide a config file for it. You can still use the log methods in the `Logger` class to log messages, then they will be delegated to corresponding methods in `log4j` logger.

Usage examples

In your source file, you need to import the following iReasoning SNMP packages:

```
import com.ireasoning.protocol.*;  
import com.ireasoning.protocol.snmp.*;
```

SnmpSession class

The core class of iReasoning SNMP library is `SnmpSession` class. It represents a communication channel between an SNMP manager and an agent. Similar to other communication classes, it needs remote host name and port number so it can connect to the remote agent, and the channel needs to be closed after it is finished. This class provides necessary methods to do SNMP queries, such as SNMP `GetRequest`, `GetNextRequest`, `SetRequest`, `InformRequest`, `GetTable`, and `Walk`. The lowest level method is "*public SnmpPdu send(SnmpPdu pdu)*",

which encodes and sends out `SnmpPdu` object to the agent. The request methods, such as `snmpGetRequest`, `snmpGetNextRequest` etc., use “*send(SnmpPdu pdu)*” method internally to do the job. So if in some cases you find the higher-level methods cannot meet your needs, you can use *send* method directly. The `snmpget.java` and `snmpgetnext.java` demonstrate how to use that method. But in most cases, it is recommended to use higher-level methods to reduce the chance of programming mistakes.

Synchronous SNMP Operations

SNMP manager application sends out a synchronous SNMP request to an agent and blocks until response comes back. Usually, synchronous operations are easier to program than asynchronous operations.

Synchronous operations in `SnmpSession` class include `snmpGetRequest`, `snmpGetNextRequest`, `snmpGetBulkRequest`, and `snmpGetTable`, etc.

The following is a simple example on how to use `SnmpSession` class to do synchronous SNMP `GetRequest`.

```

1.      SnmpTarget target = new SnmpTarget(host, port,
readCommunity,
2.                                     writeCommunity, version);
3.      SnmpSession session = new SnmpSession(target);
4.      if(isSnmpV3)
5.      {
6.          session.setV3Params(user, authProtocol,
7.                              authPassword, privPassword);
8.      }
9.
10.     SnmpPdu retPdu = session.snmpGetRequest(oids); //send out get request
11.
12.     System.out.println(retPdu);
13.     session.close();

```

In this example, it creates an `SnmpTarget` object that represents a remote agent. An `SnmpSession` object is created next with the `SnmpTarget` object. If remote agent only supports SNMPv3, additional information has to be provided. Then we call *snmpGetRequest* method of `SnmpSession` class to send out the request to the agent. Check out `snmpget.java`, `snmpgetnext.java`, `snmpgetbulk.java`, `snmpwalk.java`, and `snmpgettable.java` (in `./examples/snmp/`) for complete implementation examples.

SnmpSession.snmpGetRequest and other methods also take MIB node name besides numerical OID. However you need to call `loadMib` method beforehand to load corresponding MIB files first, then `SnmpSession` can translate name into numerical OID.

Asynchronous SNMP Operations

`SnmpSession` class provides several asynchronous SNMP operation methods, such as `asyncSnmpGetRequest`, and `asyncSnmpGetNextRequest`, etc. Compared with synchronous SNMP operations, asynchronous SNMP operations can send out more SNMP requests in the same amount of time because `SnmpSession` is not blocked waiting for responses. To receive responses, we need a class implementing `Listener` interface, therefore “`handleMsg(Object session, Msg msg)`” method will be called when a response is received.

The following is a simple example on how to use `SnmpSession` class to do asynchronous SNMP `GetRequest`.

```
1.      SnmpTarget target = new SnmpTarget(host, port,
readCommunity,
2.          writeCommunity, version);
3.      SnmpSession session = new SnmpSession(target);
4.      if(isSnmpV3)
5.      {
6.          session.setV3Params(user, authProtocol,
7.              authPassword, privPassword);
8.      }
9.      session.addListener(this);
10.     session.asyncSnmpGetRequest(oids);
11.     ...
12.     //In a class implementing Listener interface
13.     public void handleMsg(Object session, Msg msg)
14.     { // received a pdu
15.         SnmpPdu pdu = (SnmpPdu) msg;
16.         print(pdu);
17.     }
```

In this example, it creates an `SnmpTarget` object that represents a remote agent. An `SnmpSession` object is created next with the `SnmpTarget` object. If remote agent only supports SNMPv3, additional information has to be provided. A listener needs to be registered with this session, so when this session receives a response, listener’s `handleMsg` method will be invoked. Then we call `asyncSnmpGetRequest` method of `SnmpSession` class to send out the request to the agent. It returns immediately after PDU is sent, unlike synchronous methods that block until response comes back. Check out `snmpasyncget.java` (in `./examples/snmp/`) for complete implementation examples.

SNMP Table Retrieval

SnmpSession class provides a useful method to retrieve a whole MIB table.

```

1. SnmpSession session = new SnmpSession(host, port, community,
2.     community, version);
3. if(isSnmpV3)
4. {
5.     session.setV3Params(user, authProtocol, authPassword,
6.         privPassword);
7. }
8. session.loadMib2();//load MIB-II
9. SnmpTableModel table = session.snmpGetTable("ifTable");
10. for (int i = 0; i < table.getRowCount() ; i++)
11. {
12.     print(table.getRow(i));
13. }

```

In this example, MIB-II is loaded by calling *loadMib2* method. Then we can use *snmpGetTable* method with MIB node name "ifTable" to retrieve whole table. Table information is stored in SnmpTableModel object. SnmpTableModel implements Swing's TreeModel interface, so it can be easily used to create a JTable component. The following is a screenshot of tcpConnTable retrieval (result of running snmpgettextable.java example that is bundled with this product).

tcpConnState	tcpConnLocalAddress	tcpConnLocalPort	tcpConnRemAddress	tcpConnRemPort
2	0.0.0.0	135	0.0.0.0	2281
2	0.0.0.0	199	0.0.0.0	2240
2	0.0.0.0	445	0.0.0.0	34859
2	0.0.0.0	1025	0.0.0.0	43042
2	0.0.0.0	1027	0.0.0.0	2256
2	0.0.0.0	2316	0.0.0.0	2128
2	0.0.0.0	3000	0.0.0.0	43015
2	192.168.2.6	139	0.0.0.0	35067
8	192.168.2.6	2316	216.239.51.101	80
2	192.168.18.1	139	0.0.0.0	26658
2	192.168.153.1	139	0.0.0.0	43146

SnmpSession also provides *snmpGetTableColumn* method to retrieve a specific column of a table.

SnmptTrapdSession class

SNMP traps are initiated by agent or manager signaling changes or alarms occurred. This class is a daemon session that can be easily used to create SNMP trap receiver. Trap version contained in the UDP packet can be automatically figured out, so it can understand all types of trap.

The following is a simple example on how to use `SnmptTrapdSession` class to create a trap receiver.

```
1. SnmptTrapdSession session = new SnmptTrapdSession(port);
2. if(isSnmptV3)
3. {
4.     session.addV3Params(user, authProtocol, authPassword,
5.                         privPassword, trapSenderEngineID);
6.     session.setEngineID(trapdEngineID);
7. }
8. //register with session, so handleMsg will be called when trap comes
9. session.addListener(this);
10. //blocks and wait for trap
11. session.waitForTrap();
12. ...
13. public void handleMsg(Object msgSender, Msg msg)
14. {
15.     System.out.println((SnmptDataType)msg);
16. }
```

The first line of code creates an `SnmptTrapdSession` listening on a certain port number. Additional information is needed if it expects SNMPv3 traps. Line 9 registers for trap event, so *handleMsg* will be called when a trap arrives. Line 11 blocks and waits for traps. Line 13-16 is a callback method that prints out received traps. Check out `snmpttrapd.java` for a complete implementation.

SnmpTrapSender class

This class can be used to send out all three versions of SNMP traps. Although `SnmpSession` class also can be used to send out trap if properly configured, it is much easier to do it using this class instead.

Here is a simple example demonstrating sending out SNMPv1 traps.

```
1.     SnmpTrapSender trapSender = new SnmpTrapSender();
2.     SnmpV1Trap trap = new SnmpV1Trap(enterprise);
3.     trap.setTimeStamp(sysUpTime);
4.     trap.setGeneric(code);
5.     trapSender.sendTrap(host, port, trap, community);
```

Line 1 creates an `SnmpTrapSender` class. Line 2 to 4 form an `SnmpTrap` object. Line 5 sends out the trap to the destination. Check out `snmptrap.java` for a complete implementation.

SnmpPoller class

This class can send out SNMP `GetRequest` and `GetNextRequest` to agents periodically. The interval is configurable.

Here is a simple example demonstrating using `SnmpPoller` class.

```
1. SnmpTarget target = new SnmpTarget(host, port, community,
2.                               community, version);
3. SnmpSession session = new SnmpSession(target);
4. if(isSnmpV3)
5. {
6.     session.setV3Params(user, authProtocol,
7.                         authPassword, privPassword);
8. }
9. SnmpPoller poller = new SnmpPoller(session);
10. poller.addListener(this);
11. poller.snmpGetNextPoll(oids, interval);
...
...
12. public void handleMsg(Object sender, Msg msg)
13. {
14.     SnmpPdu pdu = (SnmpPdu) msg;
15.     print(pdu);
16. }
```

Line 1 to 8 create an `SnmpSession` and open a connection to the remote agent. Line 9 creates an `SnmpPoller` class using the just created `SnmpSession` object. Line 10 registers itself with the `SnmpPoller` object, so its `handleMsg` method will get called if `SnmpPoller` finishes a request. Line 11 starts the poller and tells poller to send `GetNextRequest` to the agent at specified intervals. Check out `snmppoll.java` example for a complete implementation.

Java Applet Example

Java code and HTML files are located at `examples/snmp/applet`. This example shows how to retrieve agent information within a java applet. Because of security restrictions of Java applet, the snmp agent and the web server have to be running on the same machine.

FAQ

Q. What is iReasoning SNMP API?

A. iReasoning SNMP API is the industry leading SNMP library, which provides a high performance, cross platform Java API for building network management applications. All SNMP versions (SNMPv1, SNMPv2c and SNMPv3) are fully supported. It is written in Java, and designed from the ground up to support all SNMP versions. There is no legacy code which only for a certain version of SNMP protocol. All code base are highly optimized to maximize performance and minimize overhead.

Q. What're the advantages of iReasoning SNMP API over its competitors?

A. Here are just some of the advantages over our competitors.

- Ease of use. You can take a look at the example code, such as [snmpgetnext.java](#) or [snmpwalk.java](#), to see how easy to implement SNMPv1/v2c/v3 operations
- High performance. All code bases are highly optimized to maximize performance and minimize overhead
- The first Java SNMP product to support both DES and strong 128-bit AES encryption algorithms
- Support all SNMP versions (SNMPv1, SNMPv2c, SNMPv3)
- Conform to the EJB specification
- Robust and powerful SMIV1/SMIV2 MIB parser
- Support both UDP and TCP transport layers

Q. Do SNMP security [vulnerabilities](#) reported by CERT affect iReasoning SNMP API?

A. The Finland Oulu University Secure Programming Group (OUSPG) discovered numerous vulnerabilities in SNMP implementation from many different vendors. Vulnerabilities in the decoding and subsequent processing of SNMP messages by both managers and agents may result in unauthorized privileged access, denial-of-service attacks, or cause unstable behavior. iReasoning has investigated how these vulnerabilities may impact our SNMP library and has found the following results:

- [VU#107186](#) - Multiple vulnerabilities in SNMPv1 trap handling
SNMP trap messages are sent from agents to managers. A trap message may indicate a warning or error condition or otherwise notify the manager about the agent's state. SNMP managers must properly decode trap messages and process the resulting data. In testing, OUSPG found multiple vulnerabilities in the way many SNMP managers decode and process SNMP trap messages.
iReasoning SNMP library successfully passed all the 24100 tests in OUSPG test suite! We conclude this advisory does not affect iReasoning SNMP library.
 - [VU#854306](#) - Multiple vulnerabilities in SNMPv1 request handling
SNMP request messages are sent from managers to agents. Request
-

messages might be issued to obtain information from an agent or to instruct the agent to configure the host device. SNMP agents must properly decode request messages and process the resulting data. In testing, OUSPG found multiple vulnerabilities in the way many SNMP agents decode and process SNMP request messages.

This advisory is not applicable to iReasoning SNMP library because it does not have SNMP agent functionality and does not accept SNMP request messages.

Q. Does iReasoning SNMP API conform to the EJB specification?

A. If asynchronous mode is not used, no thread is created. And EJB specification allows client side socket. So SNMP library conform to the EJB specification. However, if `asyncSend` method in `SmpSession` is used, it creates a new thread and violates EJB specification. Asynchronous mode is not recommended to be used in enterprise java beans. SNMP libraries from other vendors create thread no matter which mode to use, they generally do not conform to EJB specification.

Q. How do I get started with this library?

A. First of all, a basic understanding of SNMP and Java is required. This user guide provides nice introduction and usage examples about this library. Then you can start with example code shipped with this product. Those example code clearly illustrates how to write simple programs to do SNMP `GetRequest`, `GetNextRequest`, `Walk`, etc. Example code is also integrated into [javadoc](#) help. So you can read them from browsers. You also need an SNMP agent to test with most of example code. If you do not have a agent for testing purpose, you can run the [snmpd.exe](#) on windows to start an SNMP agent listening on port 161.

Q. Can I build an SNMP agent with iReasoning SNMP API?

A. No. You can use iReasoning Agent Builder to build SNMP agent. iReasoning Agent Builder is a superset of SNMP API, it also provides tools for building agent.

Q. Which versions of SNMP are supported by iReasoning SNMP library?

A. iReasoning SNMP library support SNMPv1, SNMPv2c and SNMPv3. However if you only purchase license for iReasoning SNMPv2 library, SNMPv3 support is not included.

Q. How's the SNMPv3 support?

A. iReasoning SNMP library fully supports SNMPv3, including the complete VACM and USM security model (HMAC-MD5, HMAC-SHA, CBC-DES, **CFB128-AES-128**). It has successfully passed a number of interoperability tests with other SNMPv3 vendors and their SNMPv3 implementations. Now it is used as a de-facto reference SNMPv3 implementation for other implementers.

Q. What is AES standard? And how does 128-bit AES encryption compare to DES?

A. Excerpt from [NIST \(National Institute of Standards and Technology\) website](#):

"The Advanced Encryption Standard (AES) is a new Federal Information Processing Standard (FIPS) Publication that will specify a cryptographic algorithm for use by U.S. Government

organizations to protect sensitive (unclassified) information. NIST also anticipates that the AES will be widely used on a voluntary basis by organizations, institutions, and individuals outside of the U.S. Government - and outside of the United States - in some cases.

The AES is being developed to replace DES, but NIST anticipates that Triple DES will remain an approved algorithm (for U.S. Government use) for the foreseeable future. **Single DES is being phased out of use, and is currently permitted in legacy systems, only.**

Assuming that one could build a machine that could recover a DES key in a second (i.e., try 255 keys per second), then it would take that machine approximately 149 thousand-billion (149 trillion) years to crack a 128-bit AES key. To put that into perspective, the universe is believed to be less than 20 billion years old."

See "[The AES Cipher Algorithm in the SNMP User-based Security Model](#)" for more details on AES in SNMP.

Q. How's the performance of your trap receiver program?

A. In our test environment, the [snmptrapd.java](#) sample program can handle trap at about 2700 traps/second (which can be higher if trap sender can reach higher sending rate) on a Pentium III 700 machine with 512MB memory, it consistently receives all the traps, while some popular trap receivers cannot catch all of the traps.

Q. How do I run example code?

A.

1. Set up java classpath.
For example:
`set classpath=lib\examples.jar;lib\ireasoningsnmp.jar`
(on windows, and assuming on a directory a level higher than lib)
2. Start an SNMP agent. You can run [snmpd.exe](#) on windows.
3. Run
`java snmpget -?`
to see the usage help.
Run
`java snmpget localhost sysUpTime`
to get the sysUpTime value from the agent

Then you can play with other example code.

Q. Is MIB parser included in your API?

A. Yes. `MibUtil.parseMib(String fileName)` method can parse MIB file and return a data structure representing MIB tree. A sample code, [mibparser.java](#), is included to illustrate the use of MIB parser.

Q. Does your SNMP API support IPv6?

A. Yes, if it's used with J2SDK/JRE 1.4. See "[Networking IPv6 User Guide for J2SDK/JRE 1.4](#)" for more information. As of JVM 1.4.2, supported operating systems are Solaris (ver 8 and up) and Linux (kernel 2.1.2 and up).

Q. What operating systems does iReasoning SNMP library run on?

A. iReasoning SNMP library is written in Java, so it can run on any OS which has JVM support.

Q. How can I make SNMP requests with object names instead of numeric OIDs?

A. iReasoning SNMP library includes a MIB parser. You need to load MIB files first by using `MibUtil.loadMib` method. To load MIB-II (RFC1213), you can call `MibUtil.loadMib2()` method. After necessary MIB files are loaded, you then can use `MibUtil` class' other method to do translation between object name and numeric OID. Check out example code for more information.

Q. Can I use your SNMP API to develop SNMP agent?

A. No. Our SNMP agent builder is the right tool for agent development.

Q. I don't want to see log message, can I disable Logger?

A. Yes. Add one line of code like the following:

```
Logger.setLevel(Logger.NONE);
```

Q. Can I put config files in a directory other than "./config"?

A. Yes. For instance, you want config files in "d:\config", just add one more java environment variable:

```
java -Dcom.ireasoning.configDir=d:\config ...
```

Q. Can I use SNMP API to access agent within a Java applet?

A. Yes. Because of security restrictions of Java applet, agent and web server have to be running on the same machine. Check out this [applet example](#).

Q. How can I change the community name after an `SnmpSession` is created?

A. You can do this way:

```
SnmpSession session = new SnmpSession(...);  
SnmpTarget target = (SnmpTarget) session.getTarget();  
target.setReadCommunity(...);
```

RESOURCES AND 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/>

- ❖ JAVA web site

<http://java.sun.com/>

GLOSSARY OF TERMS

JAVA

Java is a programming language expressly designed for use in the distributed environment of the Internet. It was designed to have the "look and feel" of the C++ language, but it is simpler to use than C++ and enforces an object-oriented programming model.

MIB

A management information base (MIB) is a formal description of a set of network objects that can be managed using the Simple Network Management Protocol (SNMP). The format of the MIB is defined as part of the SNMP. (All other MIBs are extensions of this basic management information base.) MIB-I refers to the initial MIB definition; MIB-II refers to the current definition. SNMPv2 includes MIB-II and adds some new objects.

OID

An Object Identifier (OID) is a text string that can be represented in either decimal or English notation. Every OID represents a branch on a tree. The tree begins with the root which is represented by a `.`. The root contains several branches below it. Each of these branches in turn has sub-branches beneath it. Branches at every level are labeled with a text name and an unsigned decimal identifier.

PDU

Protocol Data Unit (PDU), basically a fancy word for packet. PDUs are the building blocks of SNMP messages.

SNMP

Simple Network Management Protocol (SNMP) is the protocol governing network management and the monitoring of network devices and their functions. SNMP is described formally in the Internet Engineering Task Force (IETF) Request for Comment (RFC) 1157 and in a number of other related RFCs.

TCP

TCP (Transmission Control Protocol) is a set of rules (protocol) used along with the Internet Protocol (IP) to send data in the form of message units between computers over the Internet. While IP takes care of handling the actual delivery of the data, TCP takes care of keeping track of the individual units of data (called packets) that a message is divided into for efficient routing through the Internet. TCP is known as a connection-oriented protocol, which means that a connection is established and maintained until such time as the message or messages to be exchanged by the application programs at each end have been exchanged.

Transport Layer

In the Open Systems Interconnection (OSI) communications model, the Transport layer ensures the reliable arrival of messages and provides error checking mechanisms and data flow controls. The Transport layer provides services for both "connection-mode" transmissions and for "connectionless-mode" transmissions. For connection-mode transmissions, a transmission may be sent or arrive in the form of packets that need to be reconstructed into a complete message at the other end.

TRAP

A trap is basically an asynchronous notification sent from an SNMP agent to a network-management station to report a problem or a significant event. Like everything else in SNMP, traps are sent using UDP (port 162) and are therefore unreliable. This means that the sender cannot assume that the trap actually arrives, nor can the destination assume that it's getting all the traps being sent its way.

UDP

UDP (User Datagram Protocol) is a communications protocol that offers a limited amount of service when messages are exchanged between computers in a network that uses the Internet Protocol (IP). UDP is an alternative to the Transmission Control Protocol (TCP) and, together with IP, is sometimes referred to as UDP/IP. Like the Transmission Control Protocol, UDP uses the Internet Protocol to actually get a data unit (called a datagram) from one computer to another. Unlike TCP, however, UDP does not provide the service of dividing a message into packets (datagrams) and reassembling it at the other end.

UML

UML (Unified Modeling Language) is a standard notation for the modeling of real-world objects as a first step in developing an object-oriented design methodology.