IREASONING INC.

iReasoning SNMP API User Guide

User Guide

iReasoning Inc.
101 Convention Center Drive, 7th floor, Las Vegas, NV 89109

Table of Contents

Chapter 1

INTRODUCTION

About this document

The purpose of this document is to provide the reader with enough knowledge to start software development with 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 APIs. To effectively use APIs, readers need to refer to javadoc help that has detailed instructions and examples.

iReasoning SNMP API is a set of high performance Java library that greatly simply 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 well-recognized design patterns, and combined with highly optimized code base, all those make iReasoning SNMP library stand above the competition.

Target Audience

This document is intended for users and engineers who are responsible for developing SNMP based management applications. User does not have to be an expert in network management field. However, She or he should be familiar with basic SNMP concepts, such as SNMP GET, GET_NEXT operations. And some basic knowledge about java language is required to understand examples. One of the design goals of this library is to minimize the learning curve for users, make user write less code to achieve their programming job. So user will find it quite easy to learn and master this library.

Chapter 2

INSTALLATION

Requirements

- JDK1.2 or later versions are installed. You can download JDK from SUN's web site (http://java.sun.com/j2se/)
- At least 32MB memory is required

Installation Procedures

1. Download and unzip

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

   The directory structure will look like the following:

Directory Name	*Description*
lib	contains binary jar files
javadoc	contains javadoc HTML files
examples	contains examples source code
config	configuration files

2. Set up CLASSPATH

   ireasoningsnmp.jar needs to be added to CLASSPATH environment variable.

   On Windows:

   If iReasoning SNMP library is installed at C:\lib\iReasoning\snmp

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

On Unix/Linux:

If iReasoning SNMP is installed at /usr/local/ireasoning/snmp

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

3. 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 run the example programs.

1) Add examples.jar to CLASSPATH. Take similar step, such as
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

where hostname is the host name or ip address of snmp agent.
If you do not have a 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.

Chapter 3

USING iREASONING SNMP LIBRARY

Overview and Architecture

iReasoning SNMP library is designed and implemented using object oriented methodology and well-recognized design patterns. One of the design goals is 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 user does not have to learn the detailed underlying differences between three SNMP versions. User can always uses 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, user will realize how simple it is to write code to talk SNMP agent.

The central class is the SnmpSession. It provides all the necessary mechanism to use SNMP operations. A session represents a communication channel between SNMP manager and an agent. To communicate with multiple agents, multiple sessions have to be created. Each session is correspondent to each communication channel. 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 agent. There are four basic SNMP operations: GET, GET_NEXT, SET, GET_BULK (SNMPv2c and SNMPv3). They are represented in SnmpSession class as snmpGetRequest, SnmpGetNextRequest, snmpSetRequest, snmpGetBulkRequest methods. For high-level SNMP operations, GET_TABLE and WALK, they are represented as snmpGetTable and snmpWalk methods.

SnmpTrapdSession class is provided for collecting SNMP traps. A trapd session can be created and then waits for traps sent be agents. This session is able to understand all SNMPv1, v2c and v3 traps.

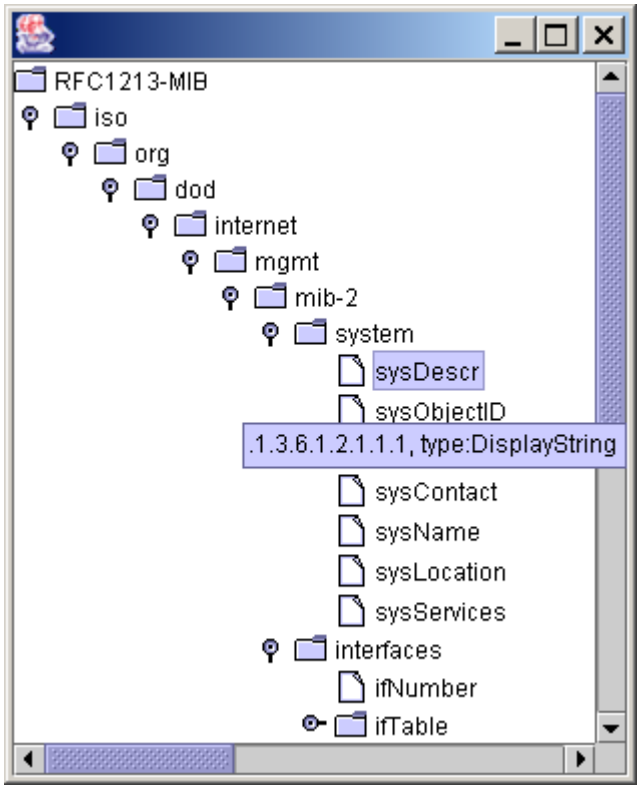
Both synchronous and asynchronous requests are supported in SnmpSession. When sending synchronous requests, the session can only send one request at a time. It must block and wait for the reply to each request before issuing next one. This mode is easier to understand and implement. When using asynchronous

requests, the session does not need to wait for each reply. In this case, session has another thread handle the reply and notify caller when reply arrives.

SNMP data types are represented as child class of SnmpDataType class. The remote agent is represented as SnmpTarget class. Each agent is represented by a single SnmpTarget object. An SnmpTarget can be instantiated with the IP address (or host name) and port number of 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 HTML files shipped with this library.

SNMP protocol uses UDP as the default transport layer protocol. UDP is connectionless and delivery of packets is not guaranteed, but it is an efficient way to transport small amount of data. iReasoning SNMP architecture also support TCP besides UDP. SNMP over TCP is a good fit for transferring large amount of data. The built in features in 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 SNMP manager can choose TCP as 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 lookup 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

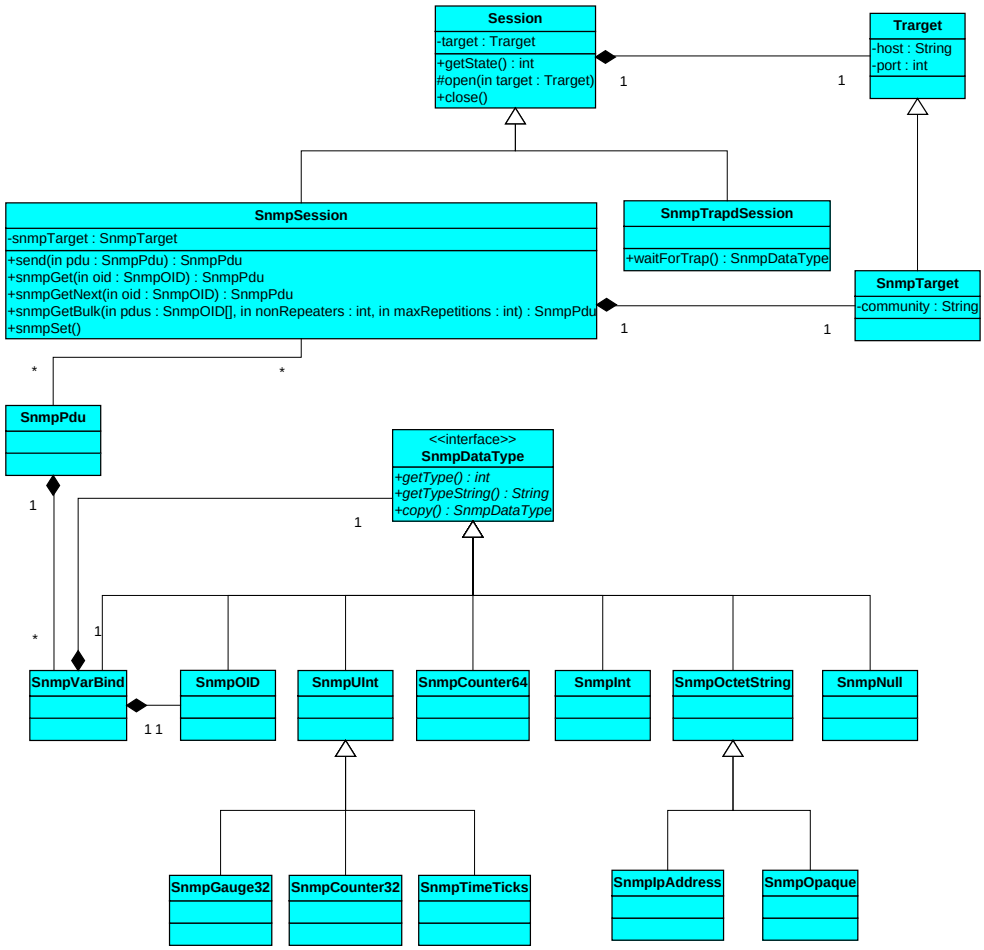


Figure 1. iReasoning SNMP library class diagram

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

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

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

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

If using log4j, 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 Logger 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 SNMP manager and agent. Similar to other communication classes, it needs remote host name and port number so it can connect to 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 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 a synchronous SNMP request to 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, 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);
11.
12.   System.out.println(retPdu);
13.   session.close();
```

In this example, it creates an SnmpTarget object that represents 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 the request to the agent. Check out snmpget.java, snmpgetnext.java, snmpgetbulk.java, snmpwalk.java, 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 to load corresponding MIB file first, so SnmpSession can translate name into numerical OID.

➢ Asynchronous Snmp Operations

SnmpSession class provides several asynchronous SNMP operation methods, such as *asyncSnmpGetRequest*, *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, so "*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.   }
```

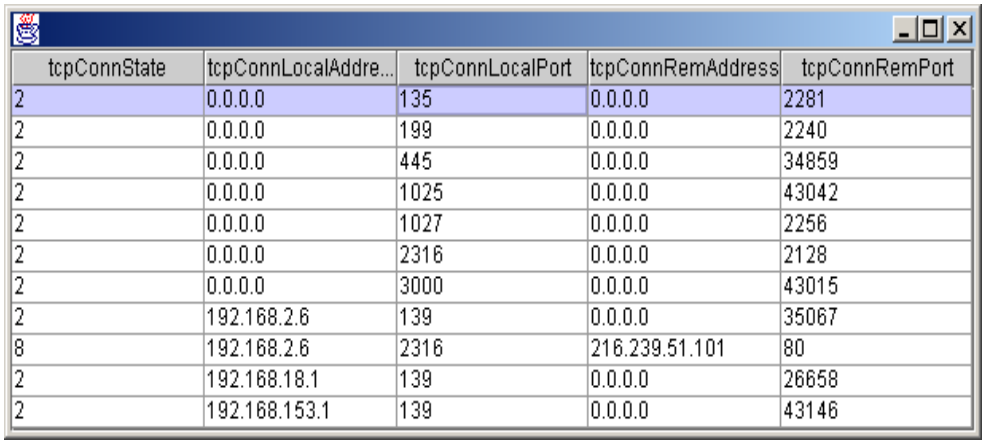
In this example, it creates an SnmpTarget object that represents remote agent. An SnmpSession object is created next with the SnmpTarget object. If remote agent only supports SNMPv3, additional information has to be provided. A listener needs to be registered with this session, so when this session receives a response, listener's *handleMsg* method will be invoked. Then we call *asyncSnmpGetRequest* method of SnmpSession class to send the request to the agent. It returns immediately after PDU is sent, not like 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 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. The following is a screenshot of tcpConnTable retrieval (result of running snmpgettable.java example which is bundled with this product).



tcpConnState	tcpConnLocalAddre...	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.

➢ SnmpTrapdSession 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 SnmpTrapdSession class to create a trap receiver.

```
1.   SnmpTrapdSession session = new SnmpTrapdSession(port);
2.   if(isSnmpV3)
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((SnmpDataType)msg);
16.  }
```

The first line creates an SnmpTrapdSession 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 trap arrives. Line 11 blocks and wait for traps. Line 13-16 is callback method that prints out received traps. Check out snmptrapd.java for a complete implementation.

➢ SnmpTrapSender class

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

Here is a simple example demonstrating sending 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 SNMP GetRequest and GetNextRequest to agent 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, snmp agent and web server have to be running on the same machine.

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.