jSNMP Enterprise™

Version 3.2

Java-based SNMP Package
User's Guide

Simplified SNMP Service for the Enterprise

jSNMP Enterprises

jSNMP Enterprise™ User's Guide

The software described in this book is furnished under a license agreement and may be used only in accordance with the terms of the agreement.

Copyright Notice

Copyright © 2003 jSNMP Enterprises

All Rights Reserved Worldwide

This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form, other than the form delivered directly from jSNMP Enterprises, without the prior consent in writing from jSNMP Enterprises, 829 Quail Court, Arroyo Grande, CA 93420.

ALL EXAMPLES WITH NAMES, COMPANY NAMES, OR COMPANIES THAT APPEAR IN THIS MANUAL ARE IMAGINARY AND DO NOT REFER TO, OR PORTRAY, IN NAME OR SUBSTANCE, ANY ACTUAL NAMES, COMPANIES, ENTITIES, OR INSTITUTION. ANY RESEMBLANCE TO ANY REAL PERSON COMPANY, ENTITIES, OR INSTITUTION IS PURELY COINCIDENTAL.

Every effort has been made to ensure the accuracy of this manual. However, jSNMP Enterprises makes no warranties with respect to this document and disclaims any implied warranties of merchantability and fitness for a particular purpose. jSNMP Enterprises shall not be liable for any errors or for incidental or consequential damages in connection with the furnishing, performance, or use of this manual or the examples herein. The information in this document is subject to change without notice.

Trademarks

jSNMP Enterprises, jSNMP, jSNMP Enterprise, and jMIBC are trademarks or U.S. registered trademarks of jSNMP Enterprises.

Other product names mentioned in this manual may be trademarks or registered trademarks of their respective companies and are the sole property of their respective manufactures.

Credits and Acknowledgments

Specials thanks to the following people who contributed to the success of this project:

Jim Pickering, Jay Chalfant, Amanda Goldner, Wes Strickland, Jim Mortensen, Beckie White, Mike Chuises, Henry Hernandez, Miles Clark, Garrett Conaty, Rick James, Gary Baker, and Chris Brannan.

Contents

INTRODUCTION **1**

DESCRIPTION OF JSNMP ENTERPRISE 1
TARGET AUDIENCE 2

INSTALLATION **3**

REQUIREMENTS 3
INSTALLATION 3
SETTING CLASSPATH 3

USING JSNMP **4**

OVERVIEW 4
INITIALIZATION 5
CONFIGURATION 5
VARIABLE BINDINGS 6
SNMPCUSTOMER 6
USER-INITIATED REQUESTS 6
USER-INITIATED NOTIFICATIONS 9
AGENT-INITIATED NOTIFICATIONS 9
MIBS 12

SNMP TO JAVA MAPPING **13**

VERSIONS OF SNMP SUPPORTED 13
SNMP OPERATIONS 13
BASIC DATA TYPES 13

CORE EXAMPLES **15**

OVERVIEW 15
JAVA APPLICATIONS 15
Building *15*
Applications *15*
JAVA APPLETS 19

RMI INTEGRATION **21**

INTRODUCTION 21
RMISnmpServer *21*
RMISnmpClient *21*
Test Applets *21*
RMI SERVER 21
Requirements *21*
Use *23*
RMI CLIENT 23
Requirements *23*
Use *23*

RMI EXAMPLES **26**

OVERVIEW 26
JAVA APPLICATIONS 26
Building *26*
Applications *26*
JAVA SERVER 26
Building *26*

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

Running ... *27*
JAVA APPLETS ... 27

CORBA INTEGRATION ... 28

INTRODUCTION ... 28
PREREQUISITES ... 28
CORBA SERVER ... 28
Requirements ... *28*
Use ... *29*
CORBA CLIENT (NON-JAVA) ... 29
CORBA JAVA CLIENT ... 29
Requirements ... *29*
Use ... *30*

CORBA EXAMPLES ... 31

OVERVIEW ... 31
JAVA APPLICATIONS ... 31
Building ... *31*
Applications ... *31*
JAVA SERVER ... 33
Building ... *33*
Running ... *33*
JAVA APPLET ... 33

ERROR MESSAGES ... 35

Invalid PDUs ... *35*
Socket Failure ... *35*
Low Memory Conditions ... *36*
Fatal Errors ... *36*
CORBA/RMI Errors ... *36*

ADDITIONAL RESOURCES ... 37

FIGURE 1: JSNMP INSTALLED DIRECTORY STRUCTURE ... 3
FIGURE 2: OVERVIEW OF JSNMP OPERATIONAL FLOW ... 4
FIGURE 3: SNMP TO JAVA DATA TYPE MAPPING USED BY JSNMP ... 13
FIGURE 4: ADDITIONAL RESOURCES ... 37

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

jSNMP Enterprise™

Java-based SNMP Package

User's Guide

Simplified SNMP Service for the Enterprise

Introduction

This manual provides the reader with information about the design, installation and use of the jSNMP Enterprise package. It contains both conceptual information as well as step-by-step examples. Topics include a design overview, installation, initialization and configuration, usage, SNMP to Java mapping, and distributed deployment. Everything you need to understand and take full advantage of jSNMP Enterprise is included.

Description of jSNMP Enterprise

jSNMP Enterprise provides application developers a cross-platform means of communicating with SNMP devices and services. The jSNMP API provides a clean, high-level interface to SNMP protocol operations in Java. The same API is also available through Java RMI (Remote Method Invocation) to support application development in distributed environments. Below we address the benefits of the jSNMP interfaces common to all implementations.

Traditional SNMP SDKs require the user to manually construct SNMP request packets. The interfaces of jSNMP, however, allow the user to communicate with SNMP agents by specifying object IDs of interest rather than worrying about the intricacies of SNMP like the Basic Encoding Rules (BER) or Protocol Data Unit (PDU) format. In addition, jSNMP has been optimized for minimizing network traffic and maximizing efficiency. jSNMP supports caching of recently retrieved results, combines multiple requests to the same agent into a single network packet, and detects duplicate requests. If a request is made for information that is currently being retrieved, jSNMP will not waste an extra packet to get the information. Instead, jSNMP will tie the two requests together, returning the SNMP result to both requesters. Finally, jSNMP optimizes the ASN.1 encoding/decoding layer by eliminating data copying in constructing and parsing SNMP packets.

To support distributed development and n-tier architectures, jSNMP Enterprise includes RMI interfaces to the jSNMP service. The jSNMP RMI services use the same optimizations described above and scale to support many clients concurrently. Additionally, jSNMP Enterprise includes a Java implementation of the jSNMP RMI client. The Java RMI client exposes the same jSNMP interface as the local Java interface. This design strategy provides two important benefits. First, it allows the Java developer to use the same interfaces for both local and remote calls to jSNMP services. Second, it shields the Java developer from the intricacies of RMI development. Thus, the jSNMP Enterprise developer can build distributed SNMP applications based upon the RMI standard with a minimal investment in training.

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

Target Audience

The developer does not need to be an expert in SNMP to develop software with jSNMP Enterprise. jSNMP was designed from the ground up to be simple to use and appeal to the Java developer with only a minimum of SNMP exposure. The developer doesn't need experience in constructing SNMP requests and other low-level protocol details. Developers should be familiar with the concept of an Object Identifier (OID) for referencing data. In addition, developers should be comfortable with the Management Information Base (MIB) structures for describing the data to be managed. To use the RMI facilities of jSNMP Enterprise, the developer need not to be fluent in RMI development. However, the developer should be comfortable using the RMI deployment environments and development tools.

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

Installation

Requirements

jSNMP Enterprise requires a JDK 1.1.x or later compatible development and runtime environment. To support DES encryption in SNMPv3 requests, a Java Cryptography Extension (JCE) must be installed along with a JDK 1.2.1 or later compatible development and runtime environment. If the encryption feature of SNMPv3 is not used, a JCE is not required. The additional RMI interfaces require a JDK 1.2.x or later compatible development and runtime environment or any other Java development and runtime environment with RMI support. **An RMI development environment is not required for local operation.**

Installation

After downloading jSNMP Enterprise, the distribution file should be uncompressed. The resulting directory structure should look like the following:

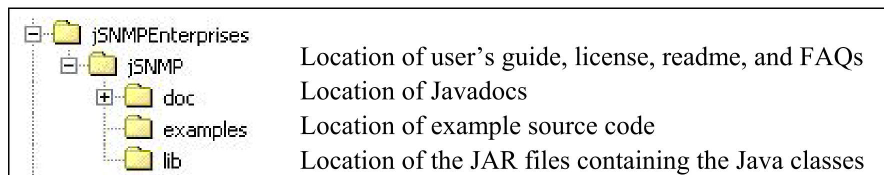

Figure 1: jSNMP Installed Directory Structure

Setting CLASSPATH

The core jSNMP service implementation is provided in the file `jSNMP.jar`, which is located in the *lib* subdirectory under the jSNMP Enterprise home directory. In order to incorporate the local jSNMP services into a Java application, the developer must set the CLASSPATH environment variable to include this JAR file. For example, this could be accomplished with the following command:

```
CLASSPATH=C:\jSNMP\lib\jSNMP.jar;%CLASSPATH%
```

for Win32 systems, or

```
CLASSPATH=/jSNMP/lib/jSNMP.jar:$CLASSPATH; export CLASSPATH
```

for Unix systems.

For detailed instructions on CLASSPATH settings for RMI environments, see the section *RMI Integration*. For detailed instructions on CLASSPATH settings for CORBA environments, see the section *CORBA Integration*.

> *NOTE: including more than one of the jSNMP JARs in the* CLASSPATH *will result in unpredictable behavior when using jSNMP*

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

Using jSNMP

Overview

jSNMP provides a natural object-oriented interface to low-level SNMP functions. The encapsulated SNMP operations are divided into three functional areas: user-initiated requests (*get, get-next, get-bulk, set, reports*), user-initiated notifications (*traps, informs*), and agent-initiated notifications (*traps, informs*). Each component is accessed through the interface `SnmpService`.

User-initiated requests are supported via asynchronous interfaces. A common analogy is the business model of ***Customers*** and ***Orders***. A customer is the entity that places orders and receives goods when the order is complete. However, the customer does not need to wait at the counter while the order is being filled. The `SnmpService` in this case acts as a virtual store and delivery agent, which receives orders, processes them, and delivers the results back to the customer.

This strategy, asynchronous completion, allows the programmer to complete network calls with potentially large latency without ever blocking. This benefit is so compelling that asynchronous completion is the sole method used for user-initiated requests.

Agent-initiated notifications, on the other hand, follow the publish-subscribe paradigm. Interested parties subscribe to the `SnmpService` with various subscription parameters. On receipt of an agent-initiated notification (*trap, inform*), the `SnmpService` will check if the notification matches a particular subscriber's profile and, if so, publish that notification to the subscriber.

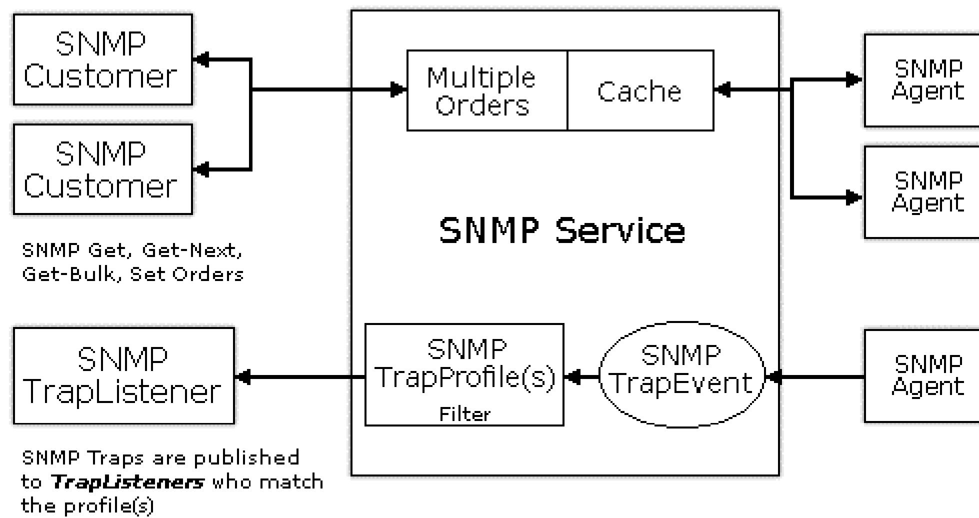

Figure 2: Overview of jSNMP Operational Flow

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

Initialization

A reference to `SnmpService` is needed in order to use jSNMP in an application. This is accomplished by using the class `SnmpLocalInterfaces` whose static method `getService()` will obtain a reference to the service. Only one implementation of this service will be created per Java virtual machine - repeated calls to `getService()` will always return the first instance.

Configuration

Configuration of the SnmpService can be done either by using the supplied `SnmpServiceConfiguration` interface, or by setting appropriate System properties. The following configuration options are available:

- socketBufferSize - the socket send and receive buffer sizes; default=dependent on Java VM
- bufferDelay - the amount of time in milliseconds the Shipper delays to accomplish PDU packing; default=20
- clerkThreadPool - the number of threads available for order delivery; default=10
- retrieveTimeStamps - add a sysUpTime request to every get/get-next request and return time stamps in returned varbind; default=false
- retryPackedTimeouts - if a packed PDU times out, break it up into multiple unpacked (single varbind) PDUs and retry with those; default=false
- trapQueueCapacity - the initial size of the system trap queue; default=1000
- trapQueueExpansion - the expansion size of the system trap queue; default=500
- cachePrunePeriod - the period of the cache cleanup thread in seconds; default=30
- cacheExpireFactor - a variable factor used during cache cleanup; larger values => larger caches; default=2
- cacheExpireFloor - the minimum lifetime of a cacheable object in seconds; default=60
- properties - dump all of the configuration parameters and exit
- maxClerks - the maximum number of clerks (request OIDs per non-atomic request PDU); default=25
- ignoreV1V2PduSizeLimit - ignore the 484 byte PDU size limit when sending SNMPv1 and SNMPv2 packets; set the PDU size limit to the socket send/receive buffer size (see "socketBufferSize"); default=false
- forceGC - forces garbage collection at the end of every cache cleanup; default=false
- dumpPackets - dump incoming and outgoing packets; the system property "outback.trace" must also be set to "trace"; default=false
- cacheDisable - turn off the cache of retrieved OID values; conserves memory
- loadRFC1213MIB - whether the RFC1213 MIB will be loaded by the SimpleMIBService; the SimpleMIBService will use less memory with this option disabled; default=true

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

Most configuration options are geared to performance tuning, and in most cases the default configuration will be fine. See the jSNMP Javadocs for details.

Variable Bindings

All data retrieved by jSNMP from an SNMP agent will be encapsulated in a `SnmpVarBind` object. The `SnmpVarBind` class provides a generic way to return the various ASN.1 types used by SNMP. All successful results from user-initiated requests will be an instance of the class `SnmpVarBind`. In addition, SNMP notifications sent by a remote agent may contain optional information as a sequence of objects. Each of these objects will be encapsulated by the `SnmpService` as a `SnmpVarBind`. The methods provided by `SnmpVarBind` to access the returned data are as follows:

```
getTimeStamp()   // returns the timestamp which corresponds to
                 // the agent sysUpTime.0 value when the varbind
                 // was retrieved from the agent, expressed in
                 // seconds
getName()        //returns the object ID (OID) representing the data
getType()        //returns the ASN.1 type of the data
getValue()       //returns the actual value of the data
getStringValue() //returns the value of the data as a String
```

See the section **SNMP to Java Mapping** for definition of the types returned.

SnmpCustomer

An instantiated `SnmpCustomer` object represents the customer mentioned in the overview. `SnmpCustomers` are guaranteed to receive a result for all orders placed. Orders are delivered to the `SnmpCustomer` via the following callback methods:

```
deliverSuccessfulOrder()
deliverFailedOrder()
```

If the order was successful, a `SnmpVarBind` will be delivered. Because the results of all user-initiated requests are delivered asynchronously, the `SnmpCustomer` must associate a number with each order placed to act as a tag to associate delivered replies with the original request as discussed below. If the order was not successful, an error code representing the failure will be delivered. See the description of `SnmpConstants` in the jSNMP Javadocs for more details on error codes.

User-Initiated Requests

User-initiated requests are made by one of the following methods:

```
placeGetOrder()
placeGetNextOrder()
placeGetBulkOrder()
placeSetOrder()
placeReportOrder()
```

An important parameter in each of these methods is *iOrderNumStart.* This parameter identifies the first *iOrderNum* to be sent in the corresponding `SnmpCustomer` callbacks discussed above. For example, if the user calls `placeGetOrder()`with a single OID and *iOrderNumStart* equal to 1, the SnmpService will call an `SnmpCustomer` callback method once with the value of 1. If the user were to include three OIDs in the `placeGetOrder()`, the SnmpService will call an `SnmpCustomer` callback method three times with the values of 1, 2, and 3.

> *Note: it is not guaranteed that callbacks will occur in order*

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

It is the responsibility of the user and, in particular, of the `SnmpCustomer` implementation to provide for unique *iOrderNum's* and to map the responses received through the callbacks onto appropriate application logic. As a convenience to the programmer, a successful call to any of the above methods will return with a value which can be used as the next *iOrderNumStart* for succeeding `placeXXXOrder()` calls. For example, if *iOrderNumStart* was 55 and there were 5 OIDs in the szOIDs argument of a `placeGetOrder()`, then the successful `placeGetOrder()` would return 60.

The general process for using one of these methods is quite simple. First, create the `SnmpOrderInfo`, `SnmpSecurityInfo`, and `SnmpAuthoritativeSession` objects appropriate for the targeted agent. These objects store network and security parameters and can be reused by the programmer across multiple requests. They relieve the programmer of having to specify host, port, timeouts, password, etc. on every request.

With these objects instantiated, call one of the `placeXXXOrder()` methods listed above. When the order completes, the `SnmpCustomer` supplied in the `placeXXXOrder()` will be called. This is where the application logic continues after the `placeXXXOrder()` has been called. In order to retrieve the results of the `placeXXXOrder()` call, the programmer must provide a meaningful implementation of the SnmpCustomer interface.

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

As an example, the following code demonstrates how to place an order for the MIB-2 variable *sysName* to a SNMPv1 agent:

```
SnmpService cService;
SnmpOrderInfo cOrderInfo;
CSMSecurityInfo cSecurityInfo;
SnmpAuthoritativeSessionFactory cAuthoritativeSessionFactory;
SnmpAuthoritativeSession cRemoteAuthoritativeSession;
…

//Set the OID we want to retrieve
String[] szOIDs = new String[1];
szOIDs[0] = "sysName.0";

//Initialize the SNMP Service
cService = SnmpLocalInterfaces.getService();

//Specify order details..
//timeout, retries, cache threshold
cOrderInfo = new SnmpOrderInfo(2, 3, 0);

//read community, write community
cSecurityInfo = new CSMSecurityInfo("public", "");

cAuthoritativeSessionFactory =
   SnmpLocalInterfaces.getAuthoritativeSessionFactory();

cRemoteAuthoritativeSession =
   cAuthoritativeSessionFactory.createRemoteAuthoritativeSession(szHost,
                              161,
                              SnmpConstants.SNMP_VERSION_1,
                              cSecurityInfo);

//Place the order!
int iOrderNumStart = 1;
cService.placeGetOrder(cRemoteAuthoritativeSession,
                       cOrderInfo,
                       true,
                       szOIDs,
                       this,
                       iOrderNumStart);
```

Note that the order specifies **`this`** as the SnmpCustomer callback interface. Here is an example of the `deliverSuccessfulOrder()` implemented in the object that made the call:

```
public void deliverSuccessfulOrder(int iOrderNum,
                                   SnmpVarBind cSnmpVarBind)
{
   System.out.println("SysName.0 = " +
                    cSnmpVarBind.getStringValue());
   cSnmpService.stop();
   System.exit(0);
}
```

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

The example code above is provided for demonstration purposes only and leaves out some relevant detail. See a full example in ~/examples/SnmpV1GetSysInfo.java.

> *NOTE: a* placeReportOrder() *call will result in a callback through the* SnmpCustomer.deliverFailedOrder() *method, as SNMPv2 REPORTs are not currently defined to require responses*

User-Initiated Notifications

User-initiated informs are made by the `placeInformOrder()` method, and user-initiated traps are made by the `placeTrapOrder()` method.

A call to the `placeInformOrder()` method requires the same type of arguments and behaves like one of the user-initiated methods above. That is, it will immediately return the next *iOrderNumStart* upon success or 0 if a parameter is invalid. The successful returned value will be the next value to use as *iOrderNumStart* for succeeding `placeXXXOrder()` calls.

When the operation completes, the SnmpService will callback on the `SnmpCustomer` interface supplied with the order.

> *Note: the* placeInformOrder() *method will fail if the* cSnmpAuthoritativeSession *argument is a SNMPv1 session, the argument* iTrapNum *is less than zero, the* szEnterpriseOID *argument is null and the* iTrapNum *argument does not equal one of the generic traps, or any parameter is invalid*

Unlike the `placeInformOrder()` method, the `placeTrapOrder()` method will not make a `SnmpCustomer` callback, as remote receiving trap entities do not respond to traps.

Agent-Initiated Notifications

Traps and informs are generated by remote SNMP agents in response to certain conditions such as rebooting the machine. jSNMP acts as a broker for these incoming notifications, publishing them to the appropriate subscribers. The primary interfaces and objects involved in notification operations are the `SnmpTrapEvent`, `SnmpTrapProfile`, `SnmpTrapListener`, and the `SnmpService`. In brief, the developer creates a `SnmpTrapProfile`, modifies it to match the traps for which notification is desired, and then registers it along with an `SnmpTrapListener` to the `SnmpService`.

Once registered, the `SnmpService` will forward `SnmpTrapEvents` to all listeners whose profile matches the notification generated. A `SnmpTrapProfile` object corresponds to the subscription parameter in the publish-subscribe paradigm. `SnmpTrapEvents` created by the `SnmpService` will be checked against each `SnmpTrapProfile`. If the event matches a profile, the event will be delivered to the appropriate subscriber. A subscriber of `SnmpTrapEvents` is represented by the interface `SnmpTrapListener`. The interface has one method:

```
trapReceived()
```

The implementation of this method is provided by the developer and should execute the business logic associated with the receipt of a trap.

To begin, the developer must create an `SnmpTrapProfile`. `SnmpTrapProfiles` are created from a `SnmpTrapProfileFactory`. The local `SnmpTrapProfileFactory` may be obtained from the `SnmpLocalInterfaces` method `getTrapProfileFactory()`. Next, if notification for only a subset of traps is desired, the `SnmpTrapProfile` must be manipulated to specify the appropriate filter. `SnmpTrapProfile` uses the following methods to define notification filters:

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

```
addTrapSession()
addInformSession()
addTrapInform()
removeTrapSession()
removeInformSession()
listTrapSessions()
listInformSessions()
removeTrapInform()
removeTrapInform()
removeTrapsInforms()
listTrapInformTypes()
listEnterpriseOIDs()
```

Finally, a developer wishing to direct notifications to a `SnmpTrapListener` interface must register that interface with the `SnmpService`. The `SnmpService` uses the following methods for managing TrapListener registrations:

```
addTrapListenerProfile()
removeTrapListenerProfile()
listTrapListenerProfiles()
removeTrapListener()
```

The `SnmpService` creates a `SnmpTrapEvent` object for every notification it receives. This object contains all the information present in the low-level trap notification and is delivered to the `SnmpTrapListener` by the `trapReceived()` call. The contents of the `SnmpTrapEvent` are accessible through the following methods:

```
getType()               // returns notification type, either
                        // SnmpConstants.SNMP_TRAPV1,
                        // SnmpConstants.SNMP_TRAPV2,
                        // or SnmpConstants.SNMP_INFORM
getPort()               // returns port notification received on
getEnterpriseOID()      // return enterprise OID of notification
getAgentIPAddress()     // returns IP address of notification
                        // agent
getSendersIPAddress()   // returns IP Address of the sender of trap's
                        // datagram packet
getSession()            // returns session of notification agent
getTimeStamp()          // returns notification time stamp
getTrapType()           // returns notification trap type
getNumberOfVarBinds()   // returns number of varbinds in
                        // notification
getVarBind()            // returns specific varbind in notification
isGeneric()             // returns true if the trap is "generic"
                        // (the Enterprise OID is 1.3.6.1.6.3.1.1.5)
```

Since all traps that match the `SnmpTrapProfile` associated with this `SnmpTrapListener` will be returned to the same instance of `trapReceived()`, that implementation may need to query the `SnmpTrapEvent` to determine the specific trap received. SNMP standards are somewhat inconsistent regarding the trap type definition for “Generic” versus “Specific” traps. In jSNMP, traps are based upon the SNMPV2 trap specification (RFC 1905 - Protocol Operations for Version 2 of the

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

Simple Network Management Protocol). That is, all traps are specified by an enterprise OID and trap code. The generic SNMPV1 traps, coldStart, warmStart, linkDown, linkUp, authenticationFailure, and egpNeighborLoss, are converted to traps with an enterprise OID of 1.3.6.1.6.3.1.1.5 and corresponding trap codes of 1, 2, 3, 4, 5, and 6 (see RFC 2576 - Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework). The SNMPv1 deprecated getGenericTrapType and getSpecificTrapType methods have been superceded by the getTrapType method, but will work as before.

Let's consider a simple example application for monitoring notification information. In this case, remote SNMP agents will generate a *coldStart* trap when started. Thus, since we want to be notified of *coldStart* traps, we must implement a `SnmpTrapListener` for the *coldStart* trap. The following code demonstrates how the application would register itself with the `SnmpService` to receive the *coldStart* trap:

```
// The SnmpService (assumed to have been initialized).
SnmpService cService;

// Implementations of the following interface
// is user-specific.
SnmpTrapListener cTrapListener;

// Obtain the factory for creating profiles
SnmpTrapProfileFactory cTrapProfileFactory =
   SnmpLocalInterfaces.getTrapProfileFactory();

// Create the trap profile that will be triggered
// whenever an agent is booted.
SnmpTrapProfile cTrapProfile =
   cTrapProfileFactory.createSnmpTrapProfile(162);

// The profile initially matches any notification, but after adding
// the following line, we restrict it to only match a notification
// which is a "coldStart".
cTrapProfile.addTrapInform("1.3.6.1.6.3.1.1.5",
             newInteger(SnmpConstants.TRAP_COLD_START.intValue() + 1));

// Register the profile with the appropriate Listener.
cService.addTrapListenerProfile(cTrapListener, cTrapProfile);

// Now, whenever the "coldStart" trap is received by the
// cService, the cTrapListener listener will be notified.
```

> *NOTE: the remote SNMP agent must be configured to deliver traps to the host machine on which the jSNMP service is running; when using the RMI or CORBA components, the service runs on the server; traps should not be sent to the client*

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

MIBs

The `SnmpMIBService` provides translation to/from OIDs and common names and provides retrieval of an OID's access, status, description, type, and 'abstract' type. It also performs translation to/from enumerated values and strings. The `SnmpMIBService` is precompiled with knowledge of RFC1213-MIB (MIB II). If you want to use the `SnmpMIBService` with other MIBs, you must either provide the appropriate MIBs or provide precompiled `jmib` dictionary files appropriate to those MIBs. jSNMP Enterprise includes jMIBC, a Java-based MIB Compiler, which produces `.jmib` dictionary files from MIBs. See the jMIBC User's Guide included in this distribution for more information. The sample HOST-RESOURCES MIB and its dependent MIBs are included in the *examples* directory of the distribution.

There are two interfaces associated with the MIB service: `SnmpMIBService` and `SnmpMIBDictionary`. `SnmpMIBService` is the store of multiple `SnmpMIBDictionaries`, each of which represent a single MIB. For a complete description of the interfaces, see the Javadocs. An overview of the primary APIs is provided below.

```
// load a MIB so we can request by name
SnmpMIBService cMIBService = SnmpLocalInterfaces.getMIBService();
InputStream cInputStream = jMIBC.loadMib("HOST-RESOURCES-MIB.my");
cSnmpMIBService.loadMIB("HOST-RESOURCES-MIB", cInputStream);
// Or alternately, load the HOST-RESOURCES-MIB jMIB file into the service
//cSnmpMIBService.loadMIB("HOST-RESOURCES-MIB",
//     new FileInputStream("HOST-RESOURCES-MIB.jmib"));
```

We can now request OIDs by name instead of by number (e.g., hrSystemUptime.0). We can also map the enumerated values returned onto the strings as defined in the MIB. For example:

```
// convert enumerated value to a string
SnmpMIBDictionary cMIBDictionary =
    cMIBService.getMIBDictionary("HOST-RESOURCES-MIB");
String szEnumValue =
    cMIBDictionary.resolveEnum("hrDeviceStatus", 2);
```

We can also retrieve the string representation of an OID's access, status, description, type, and 'abstract' type as defined in the MIB. For example:

```
// get access, status, description, type and 'abstract' type by
// object name
SnmpMIBDictionary cMIBDictionary =
    cMIBService.getMIBDictionary("HOST-RESOURCES-MIB");
String szAccess =
    cMIBDictionary. resolveNameAccess ("hrDeviceStatus");
String szStatus =
    cMIBDictionary. resolveNameStatus ("hrDeviceStatus");
String szDescription =
    cMIBDictionary. resolveNameDescription ("hrDeviceStatus");
String szType =
    cMIBDictionary. resolveNameType ("hrDeviceStatus");
String szAbstractType =
    cMIBDictionary. resolveNameAbstractType ("hrDeviceStatus");
```

For a complete example that uses the SnmpMIBService for MIB translation, see SnmpV1GetSysInfo.java in the examples directory.

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

SNMP to Java Mapping

Versions of SNMP Supported

jSNMP Enterprise currently implements the entire SNMPv1, v2c, & v3 specifications.

SNMP Operations

The following table shows the SNMP operations supported by jSNMP by SNMP version:

jSNMP API	SNMP Operation	V1	v2	v3
PlaceGetOrder()	GET	x	x	x
PlaceGetNextOrder()	GETNEXT	x	x	x
PlaceGetBulkOrder()	GETBULK		x	x
PlaceSetOrder()	SET	x	x	x
PlaceInformOrder()	INFORM		x	x
PlaceTrapOrder()	TRAP	x	x	x
PlaceV1TrapOrder()	TRAP	x		
PlaceReportOrder()	REPORT		x	x

Figure 3: SNMP to Java Request Type Mapping used by jSNMP

Basic data types

The following table shows the mapping of SNMP data types to Java:

ASN.1	Java
NULL	Null
INTEGER and Integer32	java.math.BigInteger
OCTET STRING	byte[ ]
Opaque	byte[ ]
IpAddress	java.lang.String
OBJECT IDENTIFIER	java.lang.String
Counter and Counter32	java.lang.Long
Gauge, Gauge32, and Unsigned32	java.lang.Long
TimeTicks	java.lang.Long
Counter64	java.math.BigInteger
UInteger32 (historic)	java.lang.Long

Figure 3: SNMP to Java Data Type Mapping used by jSNMP

A question may arise regarding the mapping of the ASN.1 types `INTEGER` and `Integer32` to `java.math.BigInteger` rather than to `java.lang.Integer`. This was done to maintain correctness for the ASN.1 `INTEGER` and `Integer32` types, as they are not defined as having a fixed length where as the `java.lang.Integer` has a fixed length of 4 bytes. `java.math.BigInteger` was selected because it does not have this limitation.

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

Also, the `Counter` and `Counter32`, `Gauge` and `Gauge32`, `Unsigned32` and `UInteger32`, and `TimeTicks` ASN.1 types are defined as unsigned 32-bit numbers but in Java all numbers are signed. To preserve correctness and to prevent the user from having to perform conversions, a `java.lang.Long` is used. The only potential issue with this mapping is if the user supplies jSNMP with a value that is bigger than an unsigned 32-bit value can represent. In this case, jSNMP will throw an exception.

An IP Address will be returned as a string in dotted quad notation (e.g. "10.0.0.1").

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

Core Examples

Overview

jSNMP Enterprise is shipped with several applications and applets that demonstrate basic jSNMP functionality. The applications are simple command line tools. The applets demonstrate the behavior of jSNMP within a web browser.

Java Applications

The command line example applications for the local Java interfaces are `SnmpV1GetSysInfo.java`, `SnmpV1TrapSenderTest.java`, `SnmpTrapListenerTest.java`, `SnmpV1WalkerTest.java`, `SnmpV2WalkerTest.java`, and `SnmpV3WalkerTest.java`, which are found in the *examples* directory of the distribution.

The `SnmpV1GetSysInfo` application illustrates how to query a remote SNMPv1 agent for basic system information objects using the jSNMP's `com.outbackinc.services.protocol.snmp.mib` package. The `SnmpV1TrapSenderTest` application illustrates how to send SNMPv1 traps. The `SnmpTrapListenerTest` application illustrates how to receive and filter incoming traps and informs. The `SnmpV1WalkerTest`, `SnmpV2WalkerTest`, and `SnmpV3WalkerTest` applications illustrate how to walk a MIB on remote SNMPv1, SNMPv2, and SNMPv3 agents.

Building

Building these examples is straightforward. First, the CLASSPATH must be set correctly as described in the *Installation* section above. Next compile the Java source for these applications with `javac` as shown:

```
>javac SnmpV1GetSysInfo.java
>javac SnmpV1TrapSenderTest.java
>javac SnmpV1V2V3TrapListenerTest.java
>javac SnmpV1WalkerTest.java
>javac SnmpV2WalkerTest.java
>javac SnmpV3WalkerTest.java
```

This will put the Java class files in the local directory. Make sure the local directory is on the CLASSPATH or move the class files to a directory that is on a CLASSPATH.

Applications

Each application requires different command line parameters, which can be ascertained by running an application without any parameters.

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

SnmpV1GetSysInfo
A query and output of the `SnmpV1GetSysInfo` application to determine its arguments will look like the following:

```
>java SnmpV1GetSysInfo
Usage : java SnmpV1GetSysInfo <hostname readcommunity>
        hostname : host name or IP address of agent to query
        readcommunity : read community of agent to query
```

A correct query and partial output of the `SnmpV1GetSysInfo` application will look something like the following:

```
>java SnmpV1GetSysInfo foobar public
sysName.0 found in MIB RFC1213-MIB
sysName.0 (1.3.6.1.2.1.1.5.0) value = foobar
sysName.0 (1.3.6.1.2.1.1.5.0) type = OctetString
sysName.0 (1.3.6.1.2.1.1.5.0) abstract type = DisplayString
sysName.0 (1.3.6.1.2.1.1.5.0) access = read-write
sysName.0 (1.3.6.1.2.1.1.5.0) status = mandatory
sysName.0 (1.3.6.1.2.1.1.5.0) description = An administratively-assigned
   name for this managed node. By convention, this is the node's fully-
   qualified domain name.
```

SnmpV1TrapSenderTest
A query and output of the `SnmpV1TrapSenderTest` application to determine it arguments will look like the following:

```
>java SnmpV1TrapSenderTest
Usage : java SnmpV1TrapSenderTest targethostname fromhostname timeticks
   communityname enterpriseOID trapnumber
        targethostname : host to send trap to
        fromhostname : host trap from
        timeticks : time ticks of host trap from
        communityname : community name for trap
        enterpriseOID : trap enterprise OID
        trapnumber : starting trap number for enterprise OID ... trap will
          be sent this many times with incrementing trap number
```

A correct query and output of the `SnmpV1TrapSenderTest` application will look something like the following:

```
>java SnmpV1TrapSenderTest foo.jsnmp.com fi.jsnmp.com 500 public
   1.3.6.1.6.3.1.1.5 2
placeTrapOrder(1.3.6.1.6.3.1.1.5.1) succeeded
placeTrapOrder(1.3.6.1.6.3.1.1.5.2) succeeded
```

SnmpV1V2V3TrapListenerTest
The `SnmpV1V2V3TrapListenerTest` application will run correctly, but with no output unless the remote SNMP device sends its notifications to the local host. This is typically configurable at the remote device. See your device's documentation for details. Once the local host is set as a notification recipient, start the `SnmpV1V2V3TrapListenerTest` application as follows:

```
>java SnmpV1V2V3TrapListenerTest
```

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

The correct output of this application will look something like the following when the remote agent is stopped:

```
Listening for traps and informs ...
Received a SNMPv2 trap ...
        Port : 162
        Generating Agent : 207.114.146.70
        Sending Agent : 207.114.146.70
        Time Stamp : 82035
        Enterprise OID : 1.3.6.1.4.1.2021.251
        Trap Type : 2
Received a SNMPv2 inform ...
        Port : 162
        Generating Agent : 207.114.146.70
        Sending Agent : 207.114.146.70
        Time Stamp : 82036
        Enterprise OID : 1.3.6.1.4.1.2021.251
        Trap Type : 2
Received a SNMPv1 trap ...
        Port : 162
        Generating Agent : 207.114.146.70
        Sending Agent : 207.114.146.70
        Time Stamp : 82036
        Enterprise OID : 1.3.6.1.4.1.2021.251
        Trap Type : 2
```

The correct output of this application will look something like the following when the remote agent is started:

```
Received a SNMPv2 trap ...
        Port : 162
        Generating Agent : 207.114.146.70
        Sending Agent : 207.114.146.70
        Time Stamp : 33
        Enterprise OID : 1.3.6.1.6.3.1.1.5
        Trap Type : 0
        VarBinds:
                1.3.6.1.6.3.1.1.4.3.0 (1.3.6.1.4.1.2021.250.10)
Received a SNMPv2 inform ...
        Port : 162
        Generating Agent : 207.114.146.70
        Sending Agent : 207.114.146.70
        Time Stamp : 33
        Enterprise OID : 1.3.6.1.6.3.1.1.5
        Trap Type : 0
        VarBinds:
                1.3.6.1.6.3.1.1.4.3.0 (1.3.6.1.4.1.2021.250.10)
Received a SNMPv1 trap ...
        Port : 162
        Generating Agent : 207.114.146.70
        Sending Agent : 207.114.146.70
        Time Stamp : 33
```

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

```
Enterprise OID : 1.3.6.1.6.3.1.1.5
Trap Type : 0
```

SnmpV1WalkerTest and SnmpV2WalkerTest

The `SnmpV1WalkerTest` and `SnmpV2WalkerTest` applications have identical arguments, as shown by the following example:

```
>java SnmpV1WalkerTest
Usage : java SnmpV1WalkerTest <hostname readcommunity>
        hostname : szHost on which agent resides to walk
        readcommunity : read community of agent to walk
```

The difference between the two applications is that `SnmpV1WalkerTest` assumes the remote agent *speaks* SNMPv1 and that `SnmpV2WalkerTest` assumes the remote agent *speaks* SNMPv2c. The following illustrates a sample query of an SNMPv1 agent with the `SnmpV1WalkerTest` application:

```
>java SnmpV1WalkerTest foobar.jsnmp.com public
Walking MIB Tree of foobar using SNMPv1
1: 1.3.6.1.2.1.1.1.0 (Linux foobar.jsnmp.com 2.2.12-20 #1 Mon Sep 27
10:25:54 EDT 1999 i586)
2: 1.3.6.1.2.1.1.2.0 (1.3.6.1.4.1.2021.250.10)
3: 1.3.6.1.2.1.1.3.0 (40934162)
4: 1.3.6.1.2.1.1.4.0 (jim)
5: 1.3.6.1.2.1.1.5.0 (foobar.jsnmp.com)
6: 1.3.6.1.2.1.1.6.0 (Right here, right now.)
7: 1.3.6.1.2.1.1.8.0 (0)
8: 1.3.6.1.2.1.1.9.1.2.1 (1.3.6.1.2.1.31)
9: 1.3.6.1.2.1.1.9.1.2.2 (1.3.6.1.6.3.1)
…
880: 1.3.6.1.6.3.16.1.5.1.0 (0)
881: 1.3.6.1.6.3.16.1.5.2.1.3.3.97.108.108.1 (Ç)
882: 1.3.6.1.6.3.16.1.5.2.1.4.3.97.108.108.1 (1)
883: 1.3.6.1.6.3.16.1.5.2.1.5.3.97.108.108.1 (4)
884: 1.3.6.1.6.3.16.1.5.2.1.6.3.97.108.108.1 (1)
885: order failed (No Such Name)
Ending MIB Walk
```

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

SnmpV3WalkerTest

The `SnmpV3WalkerTest` arguments are illustrated by the following example:

```
>java SnmpV3WalkerTest
Usage : java SnmpV3WalkerTest hostname username
        [ -auth AuthNoPriv|AuthPriv –authscheme MD5|SHA
        -authpwd userspassword -privpwd userspassword ]
hostname : Host on which agent resides to walk
username : user with permission to query agent
-auth AuthNoPriv : use authentication without privacy to query agent
-auth AuthPriv : use authentication with privacy to query agent
    (requires -privpwd)
-authscheme MD5 : use MD5 authentication scheme to query agent
-authscheme SHA : use SHA authentication scheme to query agent
-authpwd userspassword : use "userspassword" as authentication
    password to query agent
-privpwd userspassword : use "userspassword" as privacy
    password to query agent (required for -auth AuthPriv)
```

The `SnmpV3WalkerTest` application assumes the remote agent *speaks* SNMPv3. The following illustrates a sample query using authentication and encryption of an SNMPv3 agent with the `SnmpV3WalkerTest` application, where `joeuser` has permissions to make queries on the remote agent using the MD5 authentication scheme, `joeusersauthpwd` as an authentication password, and `joeusersprivpwd` as a privacy password:

```
>java SnmpV3WalkerTest foobar joeuser –auth AuthPriv -authscheme MD5
 -authpwd joeusersauthpwd -privpwd joeusersprivpwd
Walking MIB Tree of foobar using SNMPv3
1: 1.3.6.1.2.1.1.1.0 (Linux foobar.jsnmp.com 2.2.12-20 #1 Mon Sep 27
10:25:54 EDT 1999 i586)
2: 1.3.6.1.2.1.1.2.0 (1.3.6.1.4.1.2021.250.10)
3: 1.3.6.1.2.1.1.3.0 (41217383)
4: 1.3.6.1.2.1.1.4.0 (jim)
5: 1.3.6.1.2.1.1.5.0 (foobar.jsnmp.com)
6: 1.3.6.1.2.1.1.6.0 (Right here, right now.)
7: 1.3.6.1.2.1.1.8.0 (0)
8: 1.3.6.1.2.1.1.9.1.2.1 (1.3.6.1.2.1.31)
9: 1.3.6.1.2.1.1.9.1.2.2 (1.3.6.1.6.3.1)
…
880: 1.3.6.1.6.3.16.1.5.1.0 (0)
881: 1.3.6.1.6.3.16.1.5.2.1.3.3.97.108.108.1 (Ç)
882: 1.3.6.1.6.3.16.1.5.2.1.4.3.97.108.108.1 (1)
883: 1.3.6.1.6.3.16.1.5.2.1.5.3.97.108.108.1 (4)
884: 1.3.6.1.6.3.16.1.5.2.1.6.3.97.108.108.1 (1)
885: order failed (End Of MIB View)
Ending MIB Walk
```

Java Applets

In order to demonstrate the use of jSNMP from within a Java applet, jSNMP Enterprise includes the following sample applets:

```
SnmpV1GetSetGetNextApplet.java
SnmpV1TrapListenerApplet.java
SnmpV1TrapSenderApplet.java
```

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

```
SnmpV1WalkerApplet.java
SnmpV2GetBulkApplet.java
SnmpV2GetSetGetNextApplet.java
SnmpV2TrapInformSenderApplet.java
SnmpV2TrapListenerApplet.java
SnmpV2WalkerApplet.java
SnmpV3WalkerApplet.java
SnmpV3GetSetGetNextApplet.java
```

These applets provide examples of one approach to running jSNMP within a browser. They are very similar in nature to the *Java Local Applications* above. Each has a corresponding HTML page (i.e. `SnmpV1TrapListenerApplet.html`) in the *examples* directory of the distribution. Each HTML page describes how to build and run its corresponding applet.

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

RMI Integration

Introduction

The RMI enhancement to jSNMP adds a wrapper around the jSNMP core that allows remote client applications to access a jSNMP-based server application using the Java RMI distributed object mechanism. It does this by creating remote versions of the key jSNMP interfaces, along with wrappers that marshal methods and parameters back and forth between the remote (RMI) versions and the local versions. The goal is to make programming RMI-based jSNMP applications as similar as possible to local applications.

RMISnmpServer

The `RMISnmpServer` class creates the core RMI remote object implementations (the `SnmpService, SnmpAuthoritativeSessionFactory,` and `SnmpTrapProfileFactory`), and publishes them in the RMI Registry. This makes them accessible to remote clients. Note that `RMISnmpServer` does NOT include a `main()`method. An example server application (`RMIServer.java`) is included in the *examples* directory of the distribution. Any changes to the core `SnmpConfiguration` should be done in the RMIServer application before creating the `RMISnmpServer` object.

RMISnmpClient

The `RMISnmpClient` class is the main interface used by RMI client applications to access remote services. It provides a similar role to RMI client applications that `SnmpLocalInterfaces` does for normal jSNMP programs. The class constructor takes a single parameter, which is the hostname of the jSNMP RMI server host. Internally, it connects and holds remote references to the various RMI-exposed services.

Test Applets

RMI versions of the various test applets are included in the *examples* directory of the distribution. They follow the same naming convention used by the original examples, with *Snmp* replaced with *Rmi*.

RMI Server

Requirements

The RMI Server implementation is provided in the file `jSNMPEnterprise.jar` located in the *lib* subdirectory under the jSNMP Enterprise home directory. In order to incorporate the jSNMP RMI Server services into a Java application, the developer must set the CLASSPATH environment variable to this include this JAR file. For example, this could be accomplished with the following command:

```
CLASSPATH=C:\jSNMP\lib\jSNMPEnterprise.jar;%CLASSPATH%
```

for Win32 systems or

```
CLASSPATH=/jSNMP/lib/jSNMPEnterprise.jar:$CLASSPATH; export CLASSPATH
```

for Unix systems.

> *NOTE: including more than one of the jSNMP JARs in the* CLASSPATH *will result in unpredictable behavior*

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

Additionally, the RMI Server must be run under an RMI registry server and with a security policy. For example, to run the example RMIServer, you might do the following:

- Create a java security policy file `policy` with the following lines:
```
grant {
    // Allow everything for now
    permission java.security.AllPermission;
};
```
- Launch the RMI registry server as follows:
  Windows - `start rmiregistry`
  UNIX - `rmiregistry&`
- Launch the jSNMP RMI server as follows:
```
>java -Djava.security.policy=policy RMIServer
```

Please refer to the documentation for Windows or Unix for specific details on the use of `rmiregistry`.

A web server provides the best way to source classes under RMI. The following instructions illustrate running a jSNMP RMI Server under Red Hat Linux 6.2 using the Apache web server and the Java 2 SDK 1.3:

1. Create the `jSNMPEnterprises/jSNMP/examples/classes` directory under the document root (e.g. `/home/httpd/html`).
2. Place all jSNMP classes and jars into the `jSNMPEnterprises/jSNMP/examples/classes` directory.
3. Place all of your HTML files (i.e., `RmiV1WalkerApplet.html`) referencing the jSNMP classes in the `jSNMPEnterprises/jSNMP/examples` directory.
4. In a console window, set the CLASSPATH to the `jSNMPEnterprises/jSNMP/examples/classes` directory and start the RMI registry and jSNMP RMI server, specifying the RMI codebase:
```
>cd /home/httpd/html/jSNMPEnterprises/jSNMP/examples/classes
>export JAVA_HOME=/usr/java/jdk1.3.0_02/jre
>export PATH=$JAVA_HOME/bin:$PATH
>export CLASSPATH=.:./jSNMPEnterprise.jar: \
$JAVA_HOME/lib/ext/jce1_2_1.jar: \
$JAVA_HOME/lib/ext/sunjce_provider.jar: \
$JAVA_HOME/lib/ext/local_policy.jar: \
$JAVA_HOME/lib/ext/US_export_policy.jar
>rmiregistry&
>java -Djava.security.policy=policy \
-Djava.rmi.server.codebase= \
http://myserver/jSNMPEnterprises/jSNMP/examples/classes \
-Djava.security.policy=policy RMIServer
```
5. Ensure that the web server can resolve the hostnames of any clients that attach to it, either through DNS or a local host file.
6. Use a browser or the JDK appletviewer to view the applet:
```
>appletviewer \
http://myserver/jSNMPEnterprises/jSNMP/examples/RmiV1WalkerApplet.html
```

Note that Sun also provides a simple web server that can be used in place of the Apache web server. Also, more discussion on the RMI registry, servers, and applets can be found at http://java.sun.com/j2se/1.3/docs/guide/rmi/getstart.doc.html#7445.

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

Use

Using the jSNMP Enterprise RMI server is very simple; the class `RMISnmpServer` is provided to handle the service initialization. Simply constructing the class will start the server. The server may then be queried via the methods `getService()`, `getAuthoritativeSessionFactory()`, and `getTrapProfileFactory()` to retrieve the references which a client may use to establish communication to the server. The following code shows how an application would start the server:

```
//Start the Server
RMISnmpServer cRMIServer = new RMISnmpServer();

//You now have a RMI service waiting
//to handle SNMP requests
```

RMI Client

Requirements

The Java implementation of the RMI Client is provided in both the `jSNMPRmiClient.jar` and in the `jSNMPEnterprise.jar` which are located in the *lib* subdirectory under the jSNMP Enterprise home directory. The former is smaller and is appropriate for distribution to thin clients. In order to incorporate the Java RMI Client into an application, the developer must set the CLASSPATH environment variable to include one of these JAR files. For example, this could be accomplished with the following command:

```
CLASSPATH=C:\jSNMP\lib\jSNMPRmiClient.jar;%CLASSPATH%
```

for Win32 systems or

```
CLASSPATH=/jSNMP/lib/jSNMPRmiClient.jar:$CLASSPATH; export CLASSPATH
```

for Unix systems.

> *NOTE: including more than one of the jSNMP JARs in the* CLASSPATH *will result in unpredictable behavior*

Use

Using jSNMP Enterprise with RMI in a client application is no different than using jSNMP in a stand-alone application. Once the interfaces have been retrieved, the application developer can continue programming to jSNMP Enterprise just like it was a local application.

To retrieve the interfaces to the `SnmpService, SnmpAuthoritativeSessionFactory`, and `SnmpTrapProfileFactory`, you need only to construct the `RMISnmpClient` passing in as a parameter the address of the remote jSNMP RMI server. Then, the methods `getService()`, `getAuthoritativeSessionFactory()`, and `getTrapProfileFactory()` on `RMISnmpClient` will return the interfaces. The following code shows how this can be accomplished:

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

```
try {
    //Create the RMI client
    RMISnmpClient cRMIClient = new RMISnmpClient("foo.foobar.com");

    //Get the Interfaces
    SnmpService cService = cRMIClient.getService();
    SnmpAuthoritativeSessionFactory cAuthoritativeSessionFactory =
        cRMIClient.getAuthoritativeSessionFactory();
    SnmpTrapProfileFactory cTrapProfileFactory =
        cRMIClient.getTrapProfileFactory();
}
catch (RemoteException re) {
    System.out.println("Couldn't create RMI client");
}
```

As an example, the following code demonstrates how to place an order for the MIB-2 variable *sysName* to a SNMPv1 agent:

```
RMISnmpClient cRMIClient;
SnmpService cService;
SnmpOrderInfo cOrderInfo;
CSMSecurityInfo cSecurityInfo;
SnmpAuthoritativeSessionFactory cAuthoritativeSessionFactory;
SnmpAuthoritativeSession cRemoteAuthoritativeSession;

try {
    //Create the RMI client
    cRMIClient = new RMISnmpClient("foo.foobar.com");

    //Set the OID we want to retrieve
    String[] szOIDs = new String[1];
    szOIDs[0] = "sysName.0";

    //Initialize the Snmp Service
    cService = cRMIClient.getService();

    //Specify order details..
    //timeout, retries, cache threshold
    cOrderInfo = new SnmpOrderInfo(2, 3, 0);

    //read community, write community
    cSecurityInfo = new CSMSecurityInfo("public", "");

    cAuthoritativeSessionFactory =
        cRMIClient.getAuthoritativeSessionFactory();

    cRemoteAuthoritativeSession =
        cAuthoritativeSessionFactory.createRemoteAuthoritativeSession(
                                          szHost,
                                          161,
                                          SnmpConstants.SNMP_VERSION_1,
                                          cSecurityInfo);
```

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

```
    //Place the order!
    int iOrderNumStart = 1;
    cService.placeGetOrder(cRemoteAuthoritativeSession,
                           cOrderInfo,
                           true,
                           szOIDs,
                           this,
                           iOrderNumStart);
}
catch (SnmpSecurityException sse) {
    System.out.println("Unable to create authoritative session");
}
catch (RemoteException re) {
    System.out.println("Couldn't create RMI client");
}
catch (UnknownHostException uhe) {
    System.out.println("Unknown host " + szHost);
}
```

Note that the order specifies **`this`** as the SnmpCustomer callback interface. Here is an example of the `deliverSuccessfulOrder()` implemented in the object that made the call:

```
public void deliverSuccessfulOrder(int SnmpVarBind cSnmpVarBind)
{
    System.out.println("SysName.0 = " + cSnmpVarBind.getStringValue());
    cSnmpService.stop();
    System.exit(0);
}
```

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

RMI Examples

Overview

jSNMP Enterprise is shipped with several RMI applications and applets that demonstrate basic jSNMP RMI functionality. The applications are simple command line tools. The applets demonstrate the behavior of jSNMP under RMI within a web browser.

Java Applications

The command line example applications for the local Java interfaces are `RmiV1WalkerTest.java`, `RmiV2WalkerTest.java`, and `RmiV3WalkerTest.java`, which are found in the *examples* directory of the distribution. These applications illustrate how to walk a MIB on remote SNMPv1, SNMPv2, and SNMPv3 agents.

Building

Building these examples is straightforward. First, the CLASSPATH must be set correctly as described in the *Installation* section above. Next compile the Java source for these applications with `javac` as shown:

```
>javac RmiV1WalkerTest.java
>javac RmiV2WalkerTest.java
>javac RmiV3WalkerTest.java
```

This will put the Java class files in the local directory. Make sure the local directory is on the CLASSPATH or move the class files to a directory that is.

Applications

Each application requires different command line parameters, similar to the corresponding core applications described in the section *jSNMP Examples* above. The only addition is the name of the RMI server. For example,

```
>java SnmpV1WalkerTest agenthost public
```

becomes

```
>java RmiV1WalkerTest rmihost agenthost public
```

Java Server

In order to demonstrate the RMI Server component of jSNMP Enterprise, a sample RMI Server application (`RMIServer.java`) is included.

Building

Building this example is straightforward. First, the CLASSPATH must be set correctly as described in the *Installation* section above. Next compile the Java source with `javac` as shown:

```
javac RMIServer.java
```

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

Running

The RMI Server must be run under an RMI registry server and with a security policy. For example, to run the example `RMIServer`, you would do the following:

- Create a java security policy file `policy` with the following lines:
  ```
  grant {
      // Allow everything for now
      permission java.security.AllPermission;
  };
  ```
- Launch the RMI registry server as follows:
  Windows - `start rmiregistry`
  UNIX - `rmiregistry&`
- Launch the jSNMP RMI server as follows:
  ```
  >java -Djava.security.policy=policy RMIServer
  ```

Java Applets

In order to demonstrate the use of jSNMP under RMI from within a Java applet, jSNMP Enterprise includes the sample applets `RmiV1GetSetGetNextApplet.java`, `RmiV1TrapListenerApplet.java`, `RmiV1TrapSenderApplet.java`, `RmiV1WalkerApplet.java`, `RmiV2GetBulkApplet.java`, `RmiV2GetSetGetNextApplet.java`, `RmiV2TrapInformSenderApplet.java`, `RmiV2TrapListenerApplet.java`, and `RmiV2WalkerApplet.java`. These applets provide examples of one approach to running jSNMP under RMI within a browser. They are very similar in nature to the *Java Local Applications* above. Each has a corresponding HTML page (i.e. `RmiV1TrapListenerApplet.html`) in the *examples* directory of the distribution. Each HTML page describes how to build and run its corresponding applet.

> *NOTE: in order to use Microsoft's Internet Explorer and RMI, you need to download the RMI classes for Internet Explorer; download the RMI classes from Microsoft and unzip the* rmi.zip *file into your Windows java directory; any unzip utility capable of handling long filenames can be used, such as WinZip; alternately, you can get a nice bundled package from IBM that installs the RMI classes in the right location*

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

CORBA Integration

Introduction

jSNMP Enterprise 3.x is shipped with CORBA support to maintain backward compatibility with jSNMP Enterprise 2.x.

> *NOTE: the 3.x implementation of the jSNMP CORBA service* ***only*** *includes those interfaces previously available in jSNMP Enterprise 2.x and does not support SNMPv2c or SNMPv3. To use jSNMP with SNMPv2c or SNMPv3 support under CORBA, we suggest using* [RMI over IIOP]()

Prerequisites

In order to use the supplied CORBA interfaces, a Java ORB compatible with the OMG IDL to Java mapping is required as well as a CORBA development environment, which includes an IDL compiler.

jSNMP Enterprise is distributed with the IDL for the CORBA interfaces. The IDL is in the file `jSNMPEnterprises/jSNMP/idl/OcsSNMPv1.idl`. Either use the included `JDKStubs.jar` for Sun's ORB or the IDL must be compiled with the IDL compiler for your ORB to generate the CORBA stubs required for communication.

> *NOTE: although the IDL to Java mapping specifies portable stubs, Java ORBs do not necessarily generate stubs that are interoperable*

The IDL compiler will generate the Java source for the stubs. These stubs then need to be compiled using javac. The class files that result will need to be added to your CLASSPATH for a Java application. The following definitions will be referenced later:

STUBPATH: path to the class files of the generated stubs from IDL
ORBPATH: path to the class files required for your ORB (see ORB documentation)

CORBA Server

Requirements

The CORBA Server implementation is provided in the file `jSNMPEnterprise.jar` located in the *lib* subdirectory under the jSNMP Enterprise home directory. In order to incorporate the jSNMP CORBA Server services into a Java application, the developer must set the CLASSPATH environment variable to include this JAR file. Additionally, the classpath must include the path to the classes compiled from the IDL generated stubs as well as classes required by your ORB. If the distribution was unzipped in the root of the C: drive, this could be accomplished with the following command:

```
CLASSPATH=C:\jSNMP\lib\jSNMPEnterprise.jar;%STUBPATH%;%ORBPATH%;
%CLASSPATH%
```

for Win32 systems or

```
CLASSPATH=/jSNMP/lib/jSNMPEnterprise.jar:$STUBPATH:$ORBPATH:
$CLASSPATH; export CLASSPATH
```

for Unix systems.

> *NOTE: including more than one of the jSNMP JARs in the* CLASSPATH *will result in unpredictable behavior*

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

Use

Using the jSNMP Enterprise CORBA server is very simple; the class `CORBASnmpServer` is provided to handle the service initialization. Simply constructing the class will start the server. The server may then be queried via the methods `getServiceIOR()` and `getSnmpTrapProfileFactoryIOR()` to retrieve the references which a client may use to establish communication to the server. The following code shows how an application would start the server and export the object references to files:

```
try {
    //Open files to hold IORs
    RandomAccessFile serviceIORFile = new RandomAccessFile(args[1], "rw");
    RandomAccessFile factoryIORFile = new RandomAccessFile(args[0], "rw");

    //Start the Server
    CORBASnmpServer server = new CORBASnmpServer();

    //Write IORs to files
    serviceIORFile.writeBytes(server.getServiceIOR());
    serviceIORFile.close();
    factoryIORFile.writeBytes(server.getSnmpTrapProfileFactoryIOR());
    factoryIORFile.close();
}
catch (IOException ioe) {
}
```

CORBA Client (non-Java)

The CORBA Server provides the services described in the IDL file. These services may be retrieved by any CORBA Client implementation built from this IDL. The IORs generated by a jSNMP Enterprise server application can be published in a naming service or used manually to establish the Client/Server communication. See your appropriate ORB manuals for details.

CORBA Java Client

Requirements

The Java implementation of the CORBA Client is provided in both the `jSNMPCorbaClient.jar` and in the `jSNMPEnterprise.jar` which are located in the *lib* subdirectory under the jSNMP Enterprise home directory. The former is smaller and is appropriate for distribution to thin clients. In order to incorporate the Java CORBA Client into an application, the developer must set the CLASSPATH environment variable to include one of these JAR files. Additionally, the CLASSPATH must include the path to the classes compiled from the IDL generated stubs as well as classes required by your ORB. For example, this could be accomplished with the following command:

```
CLASSPATH=C:\ jSNMP\lib\jSNMPCorbaClient.jar;%STUBPATH%;
%ORBPATH%;%CLASSPATH%
```

for Win32 systems or

```
CLASSPATH=/ jSNMP/lib/jSNMPCorbaClient.jar:$STUBPATH:$ORBPATH:
$CLASSPATH; export CLASSPATH
```

for Unix systems.

> *NOTE: including more than one of the jSNMP JARs in the* CLASSPATH *will result in unpredictable behavior*

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

Use

Using jSNMP Enterprise with CORBA in a client application is no different than using jSNMP in a standalone application. Once the interfaces have been retrieved, the application developer can continue programming to jSNMP Enterprise just like it was a local application.

To retrieve the interfaces to the `SnmpService` and `SnmpTrapProfileFactory`, you need only to construct the `CORBASnmpClient`, passing in as parameters the IORs for the remote objects. Then, the methods `getService()` and `getSnmpTrapProfileFactory()` on `CORBASnmpClient` will return the interfaces. The following code shows how this can be accomplished given IORs from a file:

```
try
{
    //Read IORs
    RandomAccessFile factoryFile = new RandomAccessFile(args[0], "r");
    String szFactoryIOR = factoryFile.readLine();
    factoryFile.close();

    RandomAccessFile serviceFile = new RandomAccessFile(args[1], "r");
    String szServiceIOR = serviceFile.readLine();
    serviceFile.close();

    //Create the CORBA client
    CORBASnmpClient client = new
        CORBASnmpClient(szFactoryIOR,szServiceIOR);

    //Get the Interfaces
    SnmpService service = client.getService();
    SnmpTrapProfileFactory factory = client.getSnmpTrapProfileFactory();

    //Now those interfaces can be used just like local interfaces.
    //You place orders and work with traps the SAME way as you would
    //if they were local.
}
catch (IOException ioe){
}
```

CORBA Examples

Overview

jSNMP Enterprise is shipped with several CORBA applications that demonstrate basic jSNMP CORBA functionality. The applications are simple command line tools. The applet demonstrates the behavior of jSNMP under CORBA within a web browser.

Java Applications

The command line example applications for the local Java interfaces are `CORBAModTest.java`, `CORBATrapTest.java`, and `CORBAWalkerTest.java`, which are found in the *examples* directory of the distribution.

The `CORBAModTest` application illustrates how to perform a `SET` on a remote SNMPv1 agent. The `CORBATrapTest` application illustrates how to receive and filter incoming traps and informs. The `CORBAWalkerTest` application illustrates how to walk a MIB on remote SNMPv1 agents.

Building

Building these examples is straightforward. First, the CLASSPATH must be set correctly as described in the *Installation* section above. Next compile the Java source for these applications with `javac` as shown:

```
>javac CORBAModTest.java
>javac CORBATrapTest.java
>javac CORBAWalkerTest.java
```

This will put the Java class files in the local directory. Make sure the local directory is on the CLASSPATH or move the class files to a directory that is on a CLASSPATH.

Applications

Each application requires different command line parameters, which can be ascertained by running an application without any parameters.

CORBAModTest

A query and output of the `CORBAModTest` application to determine its arguments will look like the following:

```
>java CORBAModTest
Usage : java CORBAModTest <factoryIORfile> <serviceIORfile>
        <hostname> <writecommunity>
factoryIORfile : file from which to read the IOR of the
   SnmpTrapProfileFactory
serviceIORfile : file from which to read the IOR of the
   SnmpService
hostname : host on which agent resides
writecommunity : write community on host on which agent
   resides
```

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

A correct query and output of the `CORBAModTest` application will look something like the following:

```
>java CORBAModTest factoryfile servicefile foobar public
Setting sysLocation.0 on foobar with public
1.3.6.1.2.1.1.4.0 = jSNMPTester1234567890
```

CORBATrapTest

The `CORBATrapTest` application will run correctly, but with no output unless the remote SNMP device sends its notifications to the local host. This is typically configurable at the remote device. See your device's documentation for details. Once the local host is set as a notification recipient, start the `CORBATrapTest` application as follows:

```
>java CORBATrapTest factoryfile servicefile
```

The correct output of this application will look something like the following:

```
Listening for traps ...
Received a SNMPv1 trap ...
Port : 162
Enterprise OID : 1.3.6.1.4.1.2021.251
Generating Agent : 207.114.146.70
Generic Trap Type : 6
Specific Trap Type : 1
Time Stamp : 82035
VarBind information :
      Name : 1.3.6.1.4.1.9.2.9.3.1.1.2.1
      Type : Integer
      Value : 5
      Name : 1.3.6.1.2.1.6.13.1.1.192.168.1.2
      Type : Integer
      Value : 5
```

CORBAWalkerTest

A correct query and output of the `CORBAWalkerTest` application will look something like the following:

```
>java CORBAWalkerTest factoryfile servicefile foobar public
Walking MIB Tree of foobar with public
1.3.6.1.2.1.1.1.0
1.3.6.1.2.1.1.2.0
1.3.6.1.2.1.1.3.0
1.3.6.1.2.1.1.4.0
1.3.6.1.2.1.1.5.0
…
1.3.6.1.6.3.16.1.5.1.0
1.3.6.1.6.3.16.1.5.2.1.3.3.97.108.108.1
1.3.6.1.6.3.16.1.5.2.1.4.3.97.108.108.1
1.3.6.1.6.3.16.1.5.2.1.5.3.97.108.108.1
1.3.6.1.6.3.16.1.5.2.1.6.3.97.108.108.1
WalkerTest.deliveredFailedOrder(885, 2)
Ending MIB Walk
```

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

Java Server

In order to demonstrate the CORBA Server component of jSNMP Enterprise, a sample CORBA Server application (CORBAServer.java) is included.

Building

To build this application, the developer must perform the following steps:

- Compile the IDL to generate the Java stubs and skeletons for the jSNMP service. This step is outlined in the *CORBA Integration, Prerequisites* section above.
- Compile the Java skeletons created in step one. From within the directory containing the generated stubs and skeletons, execute:

  ```
  >javac *.java
  ```

- Compile the CORBAServer application. The CLASSPATH must be set to include the jSNMPEnterprise.jar file, the class files generated from the second step, and the CLASSPATH required by your ORB (see ORB documentation).

Running

The CORBAServer application requires two parameters. Each of these is a filename to which the application will write IORs for the relevant jSNMP services. Any two filenames which may be overwritten and which will be accessible to the client will do. For example:

```
>java CORBAServer file1 file2
```

The correct output will look something like the following:

```
Successfully opened files to hold IORs...
Successfully started SNMP service...
Successfully wrote IORs to files...
Awaiting requests...
```

Java Applet

In order to demonstrate the use of the remote CORBA interfaces from within a Java applet, jSNMP Enterprise includes the sample applet CORBATestApplet.java. This applet provides an example of one approach to running jSNMP within a browser. It is very similar in nature to CORBAWalkerTest, the only substantive difference being the method by which the IOR files are retrieved, and the creation of an HTML file that references the applet.

Below is example HTML with an applet tag that references a JAR, which is the standard package format supported by most browsers (see readme.txt for other options). Note that the Java classes for both the ORB (in this case OrbixWeb) and the CORBA stubs generated by the IDL compiler must be in the "/classes" directory of your web server which explains the use of codebase. codebase is required when using Netscape Communicator, since Communicator doesn't support multiple JARs in the archive tag.

```
<html>
<body>

<applet codebase="/classes" archive="jSNMPCorbaClient.jar"
        code="CORBAWalkerApplet" height=400 width=500>
  <param name="org.omg.CORBA.ORBClass" value="IE.Iona.OrbixWeb.CORBA.BOA">
```

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

```
    <param name="factoryIOR" value="factory.ior">
    <param name="serviceIOR" value="service.ior">
    <param name="managedHost" value="myRouter">
  </applet>

  </body>
  </html>
```

The parameters required are:

- *org.omg.CORBA.ORBClass*: used to specify the base CORBA ORB class, overriding any ORB bundled with the browser

- *factoryIOR*: the factory IOR file generated by the CORBAServer; this must reside in the same directory as the HTML page

- *serviceIOR*: the service IOR file generated by the CORBAServer; this also must reside in the same directory as the HTML page

- *managedHost*: the host name of the managed device (the device hosting the SNMP agent)

Also note the use of `jSNMPCorbaClient.jar`. This JAR file is included with the jSNMP distribution for use in client applets. Only the jSNMP client-side classes are included, allowing any jSNMP-based applets to load more quickly. The JAR may also be compressed to improve download speed. See the `readme.txt` file for details.

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

Error Messages

The jSNMP service emits errors to standard out under certain exceptional conditions. Unless otherwise indicated, the effect of the error is contained in the error message. The service is not otherwise affected and will continue to run normally. The possible error messages are listed below.

Invalid PDUs

```
Warning : SNMP service received incomplete PDU from x.x.x.x.  Some
requests may timeout.
Warning : SNMP service received invalid PDU from x.x.x.x.  Some requests
may timeout.
```

These messages will be displayed if jSNMP receives bad responses from remote agents. As the PDUs are bad, no further action can be taken on them. Any order corresponding to the bad PDU will timeout.

```
Warning : SNMP service received invalid trap/inform PDU from x.x.x.x.
Trap discarded.
Warning : ASN.1 Error in decoding received trap.
Warning : IO Error in decoding received trap.
```

These messages will be displayed if jSNMP receives bad traps from remote agents. As the trap PDUs are bad, no further action can be taken on them. The bad traps will then be discarded.

```
Warning : SNMP service unable to send PDU to …
```

This message will be displayed if jSNMP is unable to generate a PDU to be sent to a remote agent because of a security or message processing exception. The warning message will include an explanation for the cause of the exception. The offending PDU will not be sent and the associated SnmpCustomer will receive a deliverFailedOrder().

Socket Failure

```
Warning : SNMP service communication layer encountered socket failure.
Warning : SNMP trap receiver communication layer encountered socket
failure.
Warning : Traps being received on port x were interrupted due to an
exception, some traps may have been lost.
```

These messages are displayed if there has been an underlying socket problem. jSNMP tries to recover from a crashed socket and will try to open a new one. However, any orders that were placed over the previous socket will be cancelled.

```
Warning : SNMP service communication layer restarted successfully.
Warning : SNMP trap receiver communication layer restarted successfully.
```

These messages are displayed after the socket failure messages if a new socket is successfully opened.

```
Warning : SNMP service communication layer cannot re-establish
communication; SNMP service is inactive.
Warning : SNMP trap receiver communication layer cannot re-establish
communication; SNMP trap receiver is inactive.
Warning : Unable to receive traps on port x -- socket exception
encountered.
```

These messages are displayed after the socket failure messages if a new socket cannot be opened. Note that a single socket failure will only affect either the general SNMP service (GETs, SETs, etc.) or the trap receiver function (traps, informs, etc.).

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

Low Memory Conditions

```
Warning : SNMP service trap queue is in a out-of memory condition.
Running the garbage collector to reclaim memory.
Warning : SNMP service trap queue is in a out-of memory condition.  All
outstanding traps are void and cancelled.
```

These messages will be displayed if the trap receiver queue is out of memory.

```
Warning : deliverSuccessfulOrder(): RemoteException: …
Warning : deliverFailedOrder(): RemoteException: …
Warning : RMISnmpServer error: …
```

These messages will be displayed if a jSNMP RMI server attempts to communicate with a RMI client that has disappeared without unregistering itself from the service. The jSNMP RMI service will disconnect from the non-existent client . Any of the above warning messages is an indication of an undeliverable client order, which will be dropped.

Fatal Errors

```
Error : Unable to start SnmpService …
```

This message will be displayed if the SNMP service is unable to obtain a socket on startup. This may be caused by not having root privileges on the local host.

```
Error : ThreadResource.run() : Caught Exception (…) and aborting current
execution
```

This message will be displayed if the SNMP service encounters an internal error which causes a thread worker to abort a job. The effects are unknown. This error should never be seen.

CORBA/RMI Errors

```
Warning : deliverSuccessfulOrder(): CORBA.COMM_FAILURE …
Warning : deliverSuccessfulOrder(): CORBA.NO_IMPLEMENT …
Warning : deliverSuccessfulOrder(): RemoteException …
Warning : deliverSuccessfulOrder(): Exception …
Warning : deliverFailedOrder(): CORBA.COMM_FAILURE …
Warning : deliverFailedOrder(): CORBA.NO_IMPLEMENT …
Warning : deliverFailedOrder(): RemoteException …
Warning : deliverFailedOrder(): Exception …
Warning : CORBA.COMM_FAILURE …
Warning : CORBA.NO_IMPLEMENT …
Warning : trapReceived(): Exception: …
Warning : trapReceived(): RMISnmpTrapListenerProxy error: …
```

These messages will be displayed if a jSNMP CORBA or RMI server attempts to communicate with a CORBA or RMI client that has disappeared without unregistering itself from the service. The jSNMP CORBA/RMI service will disconnect from the non-existent client. Any of the above warning messages is an indication of an undeliverable client order, which will be dropped.

```
Warning : removeTrapListener(): Tried to remove a null remoteListenerId
```

This message will be displayed if a jSNMP RMI server receives a request to remove a null TrapListener.

```
Error : Error marshalling varBind; Type …
```

This message will be displayed if a jSNMP CORBA server is unable to marshall a return type.

Copyright © 2003 jSNMP Enterprises, All Rights Reserved.

Additional Resources

For more information we suggest:

jSNMP Enterprise™ Home Page	http://www.jsnmp.com/products.html
jSNMP Enterprise™ FAQ	http://www.jsnmp.com/docs/jSNMP/jSNMPEnterpriseFAQ.pdf
jSNMP Enterprises' collection of MIBs and jMIBC dictionary files	http://www.jsnmp.com/MIBs/
Comp.protocols.snmp FAQ (official archive)	ftp://rtfm.mit.edu/pub/usenet/comp.protocols.snmp/comp.protocols.snmp_SNMP_FAQ_Part_1_of_2 ftp://rtfm.mit.edu/pub/usenet/comp.protocols.snmp/comp.protocols.snmp_SNMP_FAQ_Part_2_of_2
SimpleWeb Internet Network Management, University of Twente (the Netherlands)	http://wwwsnmp.cs.utwente.nl/
General SNMP Discussions	**Listname**: snmp@lists.psi.com **Subscribe**: mail snmp-request@psi.com with a body of: "subscribe snmp [your address]" **Unsubscribe**: mail snmp-request@psi.com with a body of: "unsubscribe snmp"
Java Web Site	http://java.sun.com/
Java Cryptography Extension Web Site	http://java.sun.com/products/jce/index-14.html
RMI-Oriented Web Sites	http://java.sun.com/j2se/1.4/docs/guide/rmi/spec/rmi-title.html http://java.sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmiTOC.html
CORBA-oriented Web Site	http://www.omg.org
CORBA-oriented Newsgroup	news://comp.object.corba/

Figure 4: Additional Resources