

# iReasoning SNMP Agent Builder

## User Guide

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

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

# Table of Contents

<b>INTRODUCTION</b> .....	2
<b>About this document</b> .....	2
<b>Target Audience</b> .....	2
<b>INSTALLATION</b> .....	3
<b>Requirements</b> .....	3
<b>Installation Procedures</b> .....	3
<b>USING iREASONING SNMP AGENT BUILDER</b> .....	5
<b>Overview and Architecture</b> .....	5
<b>Master/Subagent Architecture</b> .....	6
<b>Integration with other SNMP Agents</b> .....	8
<b>Security Mechanisms</b> .....	11
<b>MibGen Usage</b> .....	13
<b>MibGen Command Line Tool</b> .....	20
<b>Code Generation</b> .....	21
> Automatically Generated Code .....	21
> Add Your Own Implementations .....	22
<b>Dynamic Row Creation And Deletion</b> .....	23
<b>Example 1: Build A Simple MIB-II Agent</b> .....	24
<b>Example 2: Build A Simple Master Agent</b> .....	29
<b>Example 3: Build A Simple Subagent</b> .....	29
<b>Advanced Example: Build A Multihomed Agent Using Master/Subagent architecture</b> .....	30
<b>Configuration</b> .....	33
> Logger configuration .....	33
> Agent configuration .....	34
> Agent configuration Example .....	47
<b>Example Code</b> .....	49
<b>RESOURCES AND REFERENCE</b> .....	58
<b>GLOSSARY OF TERMS</b> .....	59

---

## INTRODUCTION

### About this document

The purpose of this document is to provide the reader with enough knowledge to start developing SNMP agents using iReasoning's SNMP agent builder. Some examples are provided to better illustrate the usage of this product.

iReasoning SNMP Agent Builder is a toolset that greatly simplifies the development of SNMP agents. It provides complete support for SNMPv1, SNMPv2c and SNMPv3. This software was designed and implemented following object-oriented methodology and recognized design patterns. These advantages, combined with a highly optimized code base, make iReasoning SNMP Agent Builder stand out from the competition.

### Target Audience

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

## INSTALLATION

### Requirements

- JDK1.3 or a later version must be installed. You can download JDK from the SUN web site (<http://java.sun.com/j2se/>).
- At least 32MB memory

### Installation Procedure

#### 1. Download and unzip

Download iReasoning SNMP Agent Builder and unzip it to the desired directory, for example C:\lib\iReasoning\agent.

The directory structure will look like this:

<i>Directory Name</i>	<i>Description</i>
<i>bin</i>	Contains the Windows executable file ( <i>mibgen.exe</i> )
<i>javadoc</i>	Contains javadoc HTML files
<i>examples</i>	Contains source code examples
<i>config</i>	Contains configuration files
<i>mibs</i>	Contains standard MIB files
<i>lib</i>	Contains binary jar files

Note: For Windows operating systems, you can also download and run *setup.exe* to install the packages.

## 2. Set up CLASSPATH

To run the iReasoning agent builder, you need two jar files: *snmpagent.jar* and *ui.jar*. *snmpagent.jar* is executable (i.e. you can type “java -jar snmpagent.jar” on the command line to launch the agent builder).

To use the iReasoning agent builder, only *snmpagent.jar* needs to be added to the CLASSPATH environment variable.

On Windows:

If the iReasoning SNMP Agent Builder is installed at C:\lib\iReasoning\agent, use the command

```
set CLASSPATH=C:\lib\iReasoning\agent\lib\snmpagent.jar;%CLASSPATH%
```

On Unix/Linux:

If the iReasoning SNMP Agent Builder is installed at /usr/local/iReasoning/agent, use the command

```
CLASSPATH=/usr/local/iReasoning/agent/lib/snmpagent.jar; export CLASSPATH
```

## 3. Start the agent builder

On Windows, you can run *mibgen.exe* to start the agent builder.

On all platforms, you can run

```
java -jar snmpagent.jar
```

(Assuming that your current working directory is .../agent/lib/)

# USING IREASONING SNMP AGENT BUILDER

## Overview and Architecture

iReasoning SNMP Agent Builder was designed and implemented using object-oriented methodology and recognized design patterns. One of its design goals was to minimize the effort expended in building SNMP agents. Most of the hard work has therefore already been done in developing the base classes, and is hidden from the users. For example, the developer does not have to worry about complexities such as SNMPv3 engine auto-discovery, encoding and decoding, dynamic row creation and deletion, and inter-table indexing. All this functionality and logic is already built into the core library.

SNMPv1, SNMPv2c, and SNMPv3 (USM and VACM) are all completely supported, so the user does not have to learn the underlying differences between the three versions. A single unified interface is used to build SNMP agents for all SNMP protocols.

The agent builder can create three different types of agent: a traditional monolithic agent, a dedicated subagent, and a flexible master/subagent. Master/subagent architecture is based on Agent eXtensibility (AgentX) technology, a standard protocol that allows one to dynamically extend the managed objects in a node. This protocol permits a single SNMP agent and several subagents to connect and register several managed objects simultaneously, without having to interrupt the management service. This makes the whole management system more flexible, as each subagent can be closer to its managed information and the SNMP extensible agent needs to be started only once. For details on the AgentX protocol, please refer to RFC 2741.

Agent builder is built on Java Management Extensions (JMX<sup>TM1</sup>) technology, a standard for instrumenting manageable resources and developing dynamic agents. Each MIB table and group is modeled as a standard MBean. An MBean (Managed Bean) is a Java object that implements a specific interface and conforms to certain design patterns. It encapsulates attributes and operations through their public

1. *Agent Builder contains the JMX<sup>TM</sup> Technology. JMX and all JMX based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.*

methods, and exposes them to management applications according to standard design patterns. Standard MBeans specify the public operations accessible to SNMP clients. A GUI-based application, *MibGen*, automatically generates MBeans for a specified MIB. When the agent starts up, it registers all MBeans with MBeanServer. Upon receiving an SNMP request from a client, it employs a quick search algorithm to find the corresponding MBean and delegates part of the task to it. Developers do not have to learn about JMX in detail, but they must be aware that only methods in the MBean interface are accessible from outside the agent. Although MBean implementation classes have additional public methods, some of them are not accessible to SNMP managers.

There is always a trade-off between performance and resource usage. The performance of an agent can be improved by enabling a thread pool when configuring the agent. The thread pool can handle requests and send traps so that the agent does not get blocked during a long period of processing. Using a thread pool requires more system resources, however, so it is not recommended for resource-constrained systems.

The SNMP SET operation is transactional in nature, and the agent architecture fully supports this feature. If a SET operation fails, the system will roll back to its previous state and nothing will be changed.

## Master/Subagent Architecture

The master/subagent architecture is based on AgentX, the first IETF standard-track specification for extensible SNMP agents. Prior to its publication, users were forced to either use non-standard solutions (SMUX, DPI, etc.) or run multiple SNMP agents on different UDP ports. (The latter method generally uses proxies to access all agents through a single UDP port.) Both approaches have problems. The lack of a standard often required subagents to support multiple protocols if they were to be used on different operating systems. Proxies are also not transparent, and thus require more intelligence on the manager side.

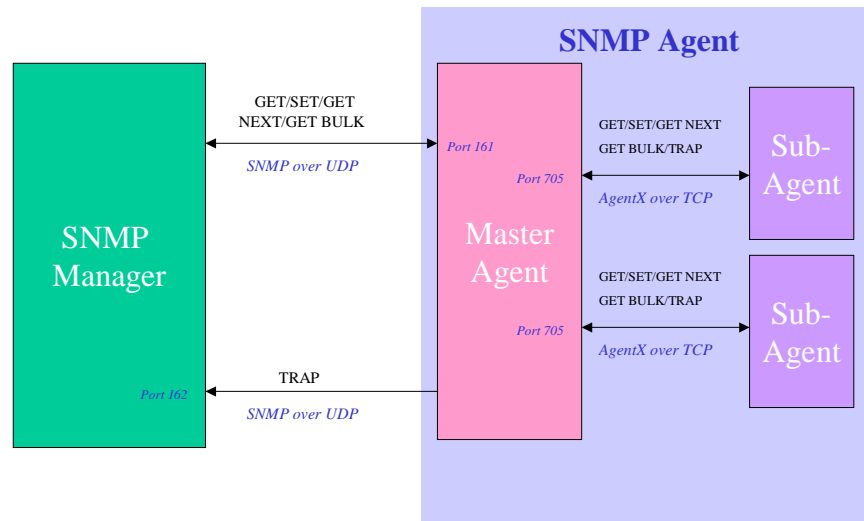
Aside from these management problems, running multiple subagents on a device is usually less costly than running an equivalent number of full-featured SNMP agents. The AgentX protocol meets this need by providing a standard agent extensibility solution. It allows multiple subagents to make MIB information transparently available to SNMP management applications. It is also designed to be independent of SNMP version. An AgentX subagent will work equally well with SNMPv1, SNMPv2c and SNMPv3 master agents without any changes.

The AgentX protocol specifies a method for subagents to advertise to the master agent that information for which they are willing to take responsibility. Each AgentX subagent can operate in its own process space, providing a robust alternative to multiple monolithic SNMP agents. A process can also use the AgentX protocol to provide access to its internal state, which then becomes available to the management station via SNMP. This last point is extremely important, considering the ever-increasing complexity of server and application

processes. Without a standard means of accessing the current state and historical data of server processes, large software systems would quickly become unmanageable. By making this information available through AgentX, we can use standard SNMP management tools to administer such software systems.

An AgentX SNMP environment consists of two types of processes: a master agent and one or more subagents, communicating with each other over TCP. The master agent speaks both AgentX and SNMP. It is the master's job to maintain a table specifying which subagents are responsible for which MIB regions. When the AgentX master receives a request via SNMP, it finds the subagent(s) responsible for the requested MIB region and dispatches the appropriate AgentX requests. The AgentX subagents are responsible for providing access to management information. When a subagent is started, it contacts the master and registers the various MIB regions for which it has information.

### Master/Subagent architecture



## Integration with other SNMP Agents

Two SNMP agents cannot both use the same IP address and port on a single machine, so in some cases you will have to integrate your agent with an existing SNMP agent. There are three ways to integrate it with other agents: through AgentX technology, by assigning a proxy agent, and by using a Windows SNMP extension agent. The choice is irrelevant to SNMP managers, which cannot tell whether query results come from the master agent, its subagents, or proxied agents.

➤ AgentX technology

If the existing agent supports AgentX, you can make it the master agent and your agent its subagent or vice versa. If your agent is already a subagent, you will need to add code to register it with the master agent.

➤ Proxy Agent

If the existing agents do not support AgentX, you can make your agent act as their proxy. In the config file, you first define which OID subtrees are will be handled by other agents. The proxy agent then delegates any SNMP requests falling under those subtrees to appropriate agents and handles all other requests itself. This is also an easy way to upgrade legacy agents to support SNMPv2/v3. Legacy agents usually only support SNMPv1, which is far less efficient and secure than SNMPv2/v3. If a proxy agent is placed in front of the legacy agent, it instantly becomes an agent supporting all SNMP versions. What's more, the proxy agent can be easily configured to different security levels.

➤ Windows SNMP Extension Agent

The SNMP extension agent is only available on Windows platforms. The library *bin/subagent.dll* serves as a bridge between the Windows native SNMP service and your Java agent. Your Java agent has to run on a port other than port 161 if it is on the same machine as the Windows SNMP service. *subagent.dll* communicates with the Java agent via standard SNMP protocol and returns results to the Windows SNMP service. A GUI tool (Tools/Windows Extension Agent menu) is supplied for configuring extension agents. When the Windows SNMP service receives a SNMP request falling under the subtrees handled by the Java agent, it will delegate the request to *subagent.dll*. *subagent.dll* will then forward it to the Java agent for processing. After the Java agent finishes processing, it sends the result back to the Windows SNMP service via *subagent.dll*, which then returns the result to the SNMP request sender. The SNMP manager is thus not aware of the existence of the Java agent; it only communicates with the Windows SNMP service.

The following two case studies demonstrate the usage of proxy agents.

#### Case one:

Your agent will run on a UNIX machine, which already has an agent running on port 161. You do not want to implement MIB-II again, preferring instead to make the existing agent provide MIB-II information. If you want your agent to run on port 161, the existing agent must be reconfigured to run on another port (in the example, we choose port 200).

In this case, the following proxy entry needs to be added to the config file.

```
<proxy
  ipAddress="127.0.0.1"
  port="200"
  included="yes"
  subTree=".1.3.6.1.2.1"
  timeout="5000"
  readCommunity="yourReadCommunity"
  writeCommunity="yourWriteCommunity"
/>
```

#### Case Two:

Your agent will run on a UNIX machine, which already has an agent running on port 161. You only want your agent to implement a specific MIB subtree, for example 1.3.1.4.15145.10, so all other SNMP requests will be handled by the existing SNMP agent. If you want your agent to run on port 161, the existing agent needs to be reconfigured to run on another port (again, we use port 200 in this example).

In this case the following proxy entry needs to be added to the config file. The most important difference between this case and case one is that the *included* parameter is set to “no” instead of “yes”.

```
<proxy
  ipAddress="127.0.0.1"
  port="200"
  included="no"
  subTree=".1.3.1.4.15145.10"
  timeout="5000"
  readCommunity="yourReadCommunity"
  writeCommunity="yourWriteCommunity"
/>
```

#### Case Three: Integration with Windows built-in SNMP service

- *subagent.dll* will serve as a bridge between the native Windows SNMP agent and your Java agent. You can use the GUI found under “Tools/Windows Extension Agent” to set up the extension agent. The advantage of this approach is its seamless integration

with the Windows SNMP service; no change to the SNMP service is needed.

- As in cases one and two, it is a little tricky to configure the port number of the Windows SNMP agent. Because a Windows SNMP agent runs on port 161 by default, it needs to be changed to another port number (such as 200). Then the Java SNMP agent can take port 161.

To change the Windows SNMP agent port number, open this file:

C:\WINNT\system32\drivers\etc\services

Find the following line,

```
snmp      161/udp          #SNMP
```

and change 161 to a new port number. Then restart the SNMP agent service.

## Security Mechanisms

### ➤ Community-based access control for SNMPv1/v2c

You can specify multiple community strings for read and write operations in the *SnmAgent.xml* config file. These community strings will only apply to SNMPv1/v2c.

### ➤ USM and VACM for SNMPv3

SNMPv3 provides much more sophisticated security mechanisms, implementing a user-based security model (USM) and a view-based access control model (VACM). This allows both authentication and encryption of the requests sent between agents and their managers. Please refer to the “Configuration” section of this manual for details on implementing these security mechanisms in the config file.

### ➤ IP-based access control for SNMP managers

You can grant access rights only to certain manager hosts by adding IP addresses or host names to the *managerIpAddresses* field in the config file. If this field is empty, all hosts are granted rights to access the agent.

*managerIpAddresses* is a comma-separated list of hosts, each of which can be expressed in any of the following forms:

- A numeric IP address, such as 192.168.2.20,
- A host name, such as server.somewhere.com,
- Or CIDR format subnet notation, such as A.B.C.D/16. For example:

192.168.1.0/24 would include all addresses between 192.168.1.0 and 192.168.1.255,

192.168.1.0/25 would include all addresses between 192.168.1.0 and 192.168.1.127,

and 192.168.1.128/25 would include all addresses between 192.168.1.128 and 192.168.1.255.

➤ IP-based access control for subagents

To prevent unauthorized subagents from connecting and registering, the master agent can restrict subagent access by adding the IP addresses of authorized subagents to the *subagentIpAddresses* field in its configuration file. If this field is empty, all hosts are granted the right to connect to the master agent.

*subagentIpAddresses* has exactly the same format as *managerIpAddresses* (see above).

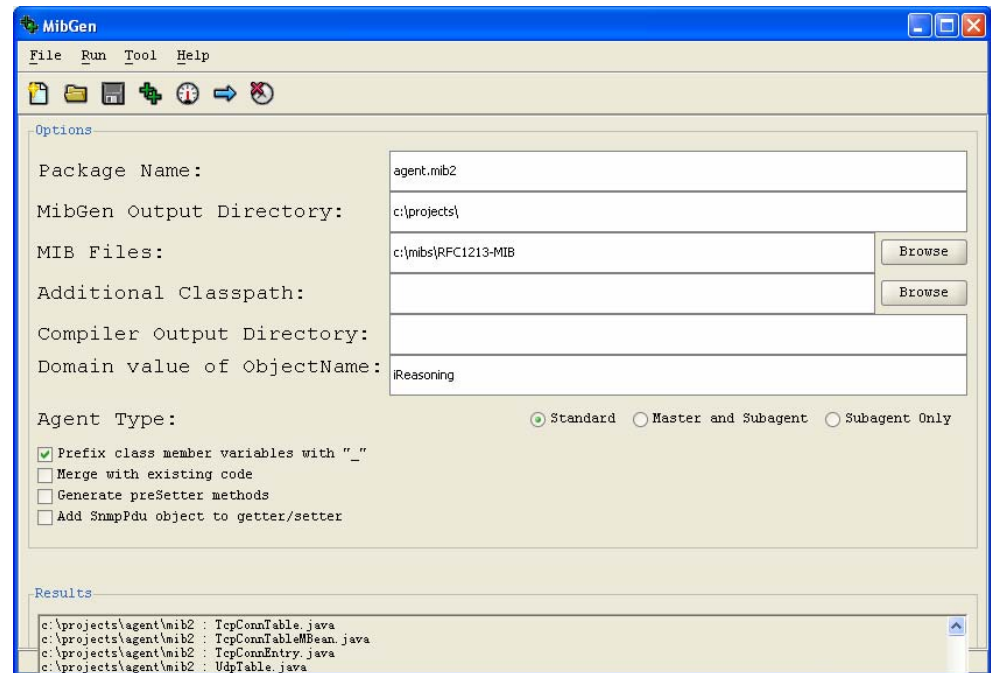
➤ Encrypted community names and passwords in the config file

Sensitive data such as community names and passwords can be optionally encrypted in the config file. See the “Configuration” section of this manual for details.

## MibGen Usage

MibGen is a GUI tool for generating the code stubs of an SNMP agent. It can also be used to compile and run the generated code.

### ➤ Main window



On this panel, the user can enter the following options for Java code generation:

<b><i>Package Name</i></b>	The package name of generated java classes.
<b><i>MibGen Output Directory</i></b>	The target directory for generated java code.
<b><i>MIB Files</i></b>	MIB files, separated by commas or semicolons.
<b><i>Additional Classpath</i></b>	Additional .jar files can be specified here for compilation and execution purposes.
<b><i>Compiler output directory</i></b>	The target directory for compiled Java classes. If this field is empty, the MibGen output directory will be used.
<b><i>ObjectName Domain</i></b>	The format of an object name is “ <b>domain</b> :name=value,...”. This field sets the domain component of all object names.
<b><i>Prefix class member variables with “_”</i></b>	If selected, all class member variables will start with an underscore.
<b><i>Generate preSet methods</i></b>	If selected, a <i>preSetFoo()</i> method will be generated for every <i>setFoo()</i> method.
<b><i>Add SnmpPdu object to getter/setter *</i></b>	If selected, each getter/setter method will be given an SnmpPdu parameter. This option is not recommended in most cases, because a SnmpPdu object is not usually necessary.
<b><i>Agent Type*</i></b>	Choose what kind of agent to create.
<b><i>Merge with existing code</i></b>	If this option is selected, MibGen will attempt to generate Java code that merges with existing code instead of overwriting it. For example, if you add a new column to an existing MIB table and regenerate the Java code, your old code for the table and its entry classes will be preserved. Methods corresponding to the new column will simply be added to the table and entry classes.
<b><i>Results</i></b>	A text area for displaying action output.

**\* “Add SnmpPdu object to getter/setter” notes:**

If this option is selected, each getter/setter method will be given an SnmpPdu parameter as was the case in versions prior to 4.x. For example, the `getSysDescr` method in the `SystemGroup` class will look like this:

```
public String getSysDescr(SnmpPdu pdu)
```

The pdu object contains information about the SNMP request, which is sometimes necessary to generate a response.

If this option is not selected, the `SnmPdu` parameter will be omitted in each getter/setter method and the above example will instead look like this:

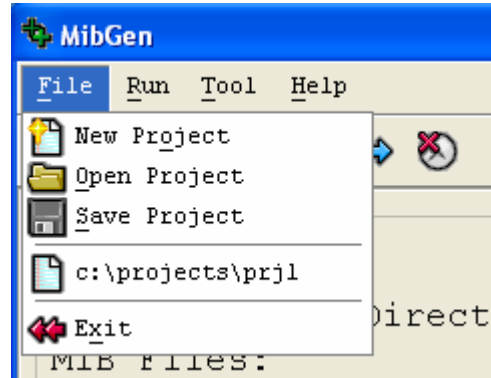
```
public String getSysDescr()
```

The object names of the classes should contain “noPdu=true”. For example, the object name of the system group class is “ireasoning:name=SystemGroup,noPdu=true”.

**\* “Agent Type” notes:**

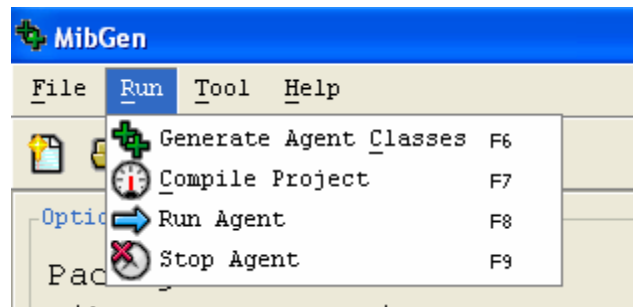
- Standard: a monolithic agent, which handles all MIB views and SNMP requests.
- Master and subagent: the agent can handle SNMP requests as well as AgentX requests. It can also act as a subagent, connecting to another master agent.
- Subagent only: the agent can only serve as a subagent, and cannot handle SNMP requests directly. SNMP requests must be forwarded from the master agent.

## ➤ File Menu



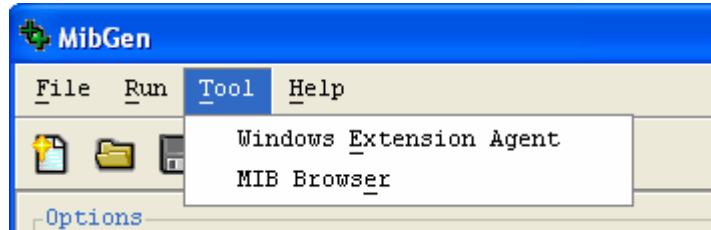
Current settings can be saved to a project file, so that next time you do not have to re-enter options. When the agent builder starts up, it automatically loads the previous project file. Clicking on “New Project” will reset all the fields.

## ➤ Run Menu



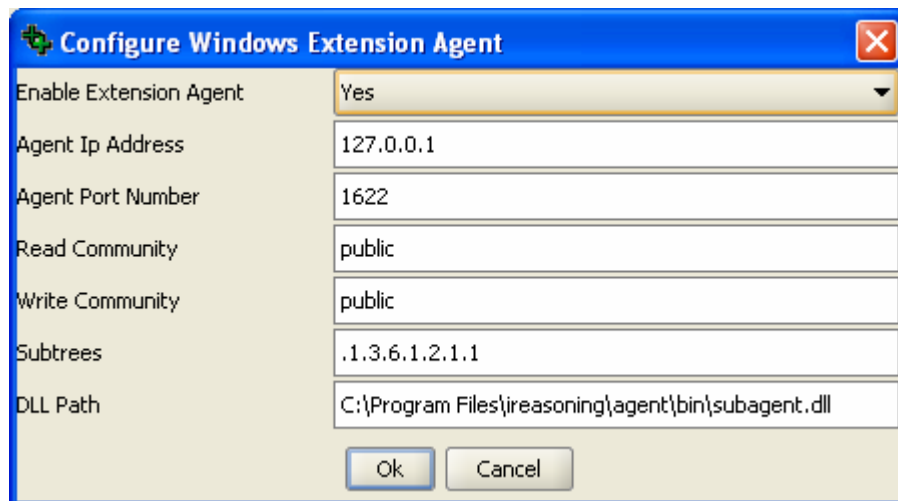
<b><i>Generate Agent Classes</i></b>	Start generating Java source code for the agent
<b><i>Compile Project</i></b>	Use <i>javac</i> to compile the generated Java code
<b><i>Run Agent</i></b>	Start the agent at port 161. Make sure this port is not taken by other programs first.
<b><i>Stop Agent</i></b>	Stop running the agent process

## ➤ Tools Menu



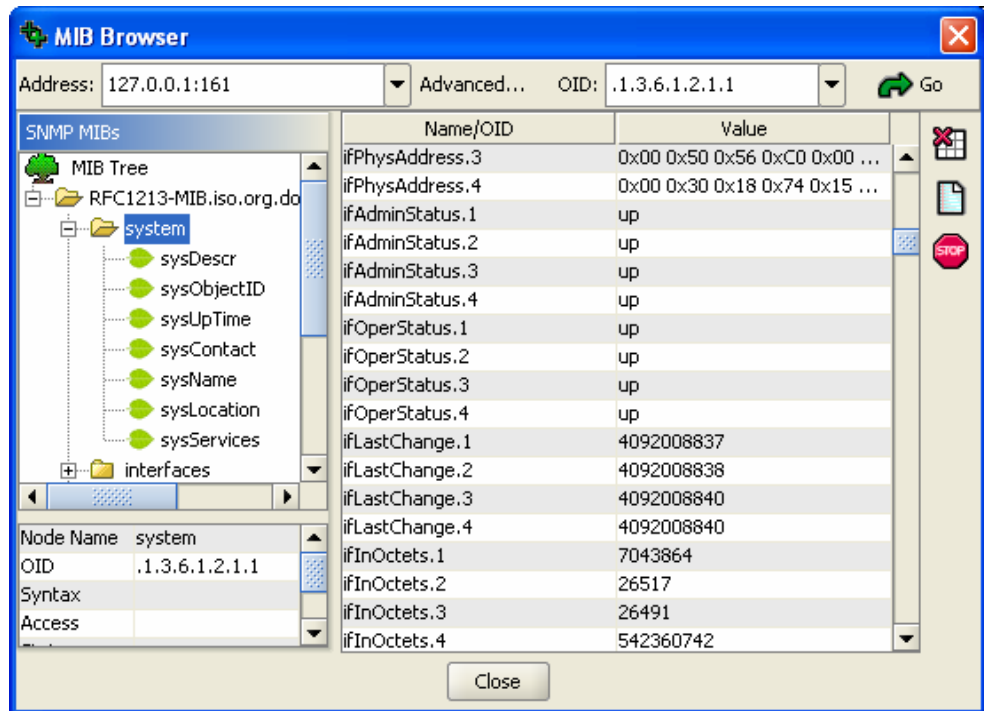
- **Windows Extension Agent**

This dialog window is used to configure a Windows SNMP extension agent. The Windows Registry will be updated after you click on the “Ok” button, but no java code will be generated. You still need to use the main window to generate a Java agent and make it run on the port specified in the “Agent Port Number” field. When the Windows SNMP Service and Java agent are both running, the former will delegate to the Java agent all SNMP requests falling under the subtrees specified in the following window:



<b><i>Enable Extension Agent</i></b>	Select “Yes” to enable the extension agent, and “No” to disable it.  <i>Warning:</i> This option may interfere with your Windows SNMP service if the extension agent is not correctly configured.
<b><i>Agent Ip Address</i></b>	Enter the IP address of your Java SNMP agent. The Java agent does not have to be on the same machine as the windows SNMP service.
<b><i>Agent Port Number</i></b>	Enter the port number of your Java SNMP agent. Because port 161 is already taken by the Windows SNMP service, you have to choose another port if the Java agent and Windows SNMP service are on the same machine.
<b><i>Read Community</i></b>	Read community name of your Java SNMP Agent.
<b><i>Write Community</i></b>	Write community name of your Java SNMP Agent.
<b><i>Subtrees</i></b>	One or more OID subtrees to be supported by your Java agent, separated by commas.
<b><i>DLL Path</i></b>	The path name of <i>subagent.dll</i> . By default, <i>subagent.dll</i> is located in the bin directory. You can, however, move <i>subagent.dll</i> to another directory.

- **MIB Browser**



This dialog window provides a tool for browsing MIBs and querying SNMP agents. Refer to <http://www.iReasoning.com/mibbrowser.shtml> for more details on its usage.

## MibGen Command Line Tool

MibGen can also be run from the command line.

Usage:

On Windows:

```
mibgencmd.bat projectFile
```

On Unix/Linux:

```
sh mibgencmd.sh projectFile
```

*projectFile* can be created using the MibGen GUI. Enter all the necessary fields and then click on “File/Save Project” in the menu to save it to a project file.

## Code Generation

### ➤ Automatically Generated Code

After clicking on the “Generate Agent Classes” menu item, the following files will be generated under the specified directory.

<b><i>README.txt</i></b>	Contains a simple description of the generated classes
<b><i>config/SnmpAgent.xml</i></b>	The XML config file, which can be customized later. See the “Configuration” chapter of this manual for more details
<b><i>Agent.java</i></b>	The main class for starting the agent
<b><i>AgentMib.java</i></b>	A class for registering MBeans
<b><i>OIDTree.java</i></b>	A tree representation of the MIB
<b><i>MBeans and their implementation classes</i></b>	MBeans representing MIB objects

**OIDTree.java** is not intended to be modified by users, because it is a tree representation of the MIB. If there are any changes in the MIB, the **OIDTree** class will need to be regenerated by the agent builder.

**AgentMib.java** registers the necessary MBeans. If you only plan to implement a subset of the MIB, you can save resources by commenting out MBean registration statements in the *registerMBeans* method.

**Agent.java** is the main agent class. It starts the agent and listens for requests on a specified UDP port. You can initialize MIB objects by adding code to this class.

The agent builder generates MBeans and their implementation classes for tables and groups. For example, if an agent implements MIB-II, **SystemGroup.java** and **SystemGroupMBean.java** will be generated for *systemGroup*. Similarly, **IfTable.java** and **IfTableMBean.java** will be generated for *ifTable*. MBean interfaces map exactly to the MIB SMI definitions, and should not be modified by users. The implementation classes of an MBean interface may have more getter or setter methods than the interface itself, but these extra methods are not accessible to the SNMP manager. Only getter and setter methods declared in an MBean interface can be accessed by the SNMP client. The user can add logic to the implementation classes so that the agent can correctly respond to client requests.

### ➤ Add Your Own Implementations

After automatic code generation, you can add your own code to MIB groups and tables to customize the behavior of the agent. In this case the config file (SnmpAgent.xml) may also need to be customized to suit your needs. Let's take a look at some simple examples:

#### ○ Scalar variables

For the MIB-II system group, we want to return the *sysDescr* MIB object value. We just need to add some code to *SystemGroup.java*, which has already been generated by the agent builder.

```
public synchronized String getSysDescr()
{
    if(this._sysDescr.length() == 0)
    {
        this._sysDescr = System.getProperty("os.name") + " " +
            System.getProperty("os.arch") + " " +
            System.getProperty("os.version");
    } //New code
    return this._sysDescr;
}
```

Now when the agent receives an SNMP GET request with OID *sysDescr.0*, it will be delegated to the *getSysDescr* method given above. The returned value of *getSysDescr* will be used in the response sent back to the SNMP manager.

#### ○ Tabular variables

We want to add code to the MIB-II *ifTable* to return all current system interfaces. *IfTable.java* should be modified as follows:

```
public IfTable (OIDTreeNode root, String oid, Object[] args)
{
    super(root, oid);
    addInterfaces();
}
private void addInterfaces()
{
    _netifs = Util.ipconfig();
    for (int i = 0; i < _netifs.size() ; i++)
    {
        NetInterface netif = (NetInterface) _netifs.get(i);
        int index = i + 1;
        IfEntry entry = new IfEntry(this, index ,
            netif.descr,
            netif.physAddress)
        addRow(entry);
        _ipAddrMap.put(netif.ipAddress, "" + index);
    }
}
public String getIfIndex(String ipAddress)
{
    return (String) _ipAddrMap.get(ipAddress);
}
```

The *IfTable* class constructor calls the *addInterfaces()* method, which collects interface information and adds table rows. To add a row, a new entry

instance first needs to be created and then added to the table by calling `addRow(SnmpTableEntry)`. If `addRow(SnmpTableEntry)` is not called, this row will not take effect and the SNMP manager will not be able to see it. When the agent receives a SNMP GET request with OID `ifIndex.1`, it will be delegated to the `IfTable.getIfIndex(...)` method. The returned value of `getIfIndex` will be used in the response sent back to the SNMP manager.

For more information you can take a look at the MIB-II and IF-MIB agent implementations, which are located in `./examples/agent`. These two examples demonstrate how to create agents that gather static and dynamic system information, and how to handle inter-table indexing and dynamic tables.

## Dynamic Row Creation and Deletion

Dynamic row creation and deletion are automatically handled in the `SnmpBaseAgent` class; there is no need to add any of your own code. The corresponding MIB table must have a `RowStatus` or `EntryStatus` column.

You can customize the `preAddRow` and `preDeleteRow` methods of table classes (subclasses of `SnmpTable`) to disable dynamic row creation and deletion.

The IF-MIB sample agent (located in the directory `examples/agent/ifmib`) implements two dynamic tables: `ifStackTable` and `ifRcvAddressTable`. Because these two tables are dynamic, we can create and delete table rows on the fly.

For example, to create a row using the SNMP SET operation (assuming that the IF-MIB agent is running on localhost) you can write:

```
java snmpset localhost .1.3.6.1.2.1.31.1.2.1.3.2.2 i 4  
(to create a row for ifStackTable), and
```

```
java snmpset localhost .1.3.6.1.2.1.31.1.4.1.2.5.6.0.160.204.232.63.205 i  
4 .1.3.6.1.2.1.31.1.4.1.3.5.6.0.160.204.232.63.205 i 1  
(to create a row for ifRcvAddressTable).
```

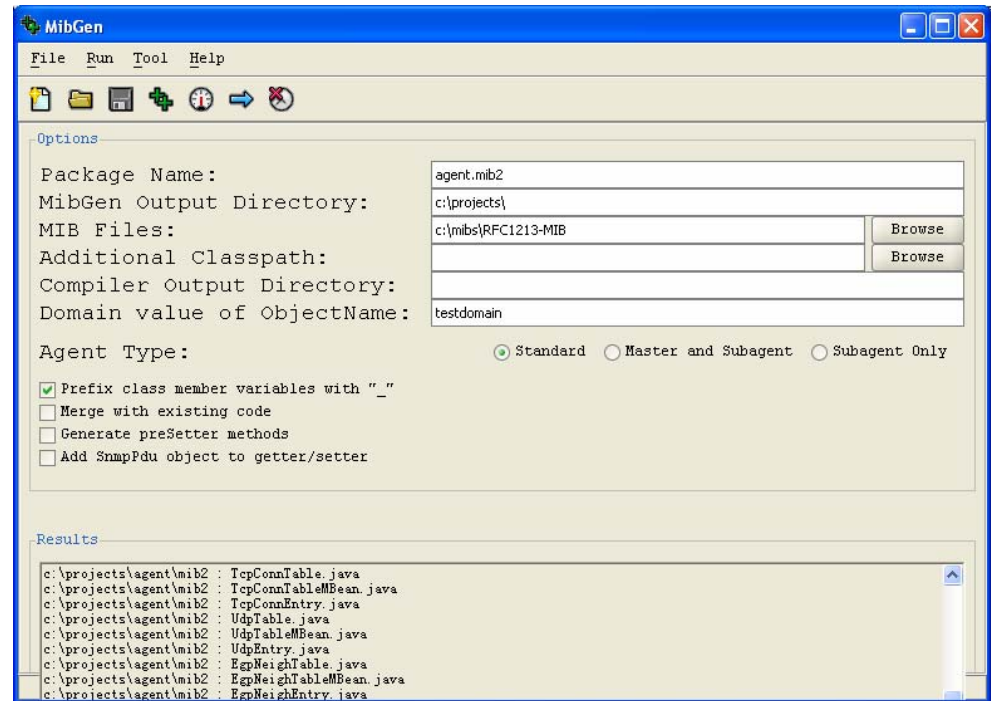
To delete a row using the SNMP SET operation you can write:

```
java snmpset localhost .1.3.6.1.2.1.31.1.4.1.2.5.6.0.160.204.232.63.205 i 6  
(to delete the row just created in ifRcvAddressTable)
```

## Example 1: Build a Simple MIB-II Agent

Now we demonstrate how to build a simple agent for a subset of MIB-II, implementing the system group and tcpConnTable. As MIB-II is defined in the file *RFC1213-MIB*, we first need to load that MIB into MibGen.

### ➤ Step 1: Code generation



In this screen, we specify the package name for generated classes to be “agent.mib2”. Java source code will be generated in the “C:\projects” directory. The MIB file to be used is *RFC1213-MIB*. The agent will be a traditional SNMP agent without master/subagent support.

The relevant files generated by MibGen are:

- Agent.java : Agent main class
- AgentMib.java : Class for registering MBeans
- OIDTree.java : Tree data structure representing the MIB
- SystemGroup.java : Class representing the System group
- SystemGroupMBean.java : MBean interface for the System group
- TcpConnEntry.java : Class representing a row in tcpConnTable
- TcpConnTable.java : Class representing tcpConnTable
- TcpConnTableMBean.java : MBean interface for tcpConnTable
- ...

## ➤ Step 2: Remove unused MBeans

MibGen generates MBeans for each table and group node in the MIB tree. In this example we only implement a subset of MIB-II, so we should only register relevant MBeans in order to save memory. In the `registerMBeans` method of the `AgentMib` class, we can comment out unnecessary registrations. Although these classes still exist, they will not be loaded into memory.

```
public static void registerMBeans(MBeanServer server, OIDTreeNode root)
{
    ...
    try
    {
        registerSystemGroup(); // System group
        registerTcpConnTable(); // tcpConnTable

        //registerIpAddrTable();
        //registerIfTable();
        //registerInterfacesGroup();
        // registerSnmppGroup();
        // registerUdpGroup();
        // registerTcpGroup();
        // registerEgpGroup();
        // registerIpGroup();
        // registerIcmpGroup();
        // registerAtTable();
        // registerIpRouteTable();
        // registerIpNetToMediaTable();
        // registerUdpTable();
        // registerEgpNeighTable();
    }
    ...
}
```

➤ Step 3: Add `system group` implementation

Here we implement methods for returning the values of the `sysDescr`, `sysName` and `sysUpTime` MIB objects. In the `SystemGroup` class, add :

```
public synchronized String getSysDescr()
{
    if(this._sysDescr.length() == 0)
    {
        this._sysDescr = System.getProperty("os.name") + " " +
            System.getProperty("os.arch") + " " +
            System.getProperty("os.version");
    }
    return this._sysDescr;
}
public synchronized String getSysName()
{
    if(_sysName.length() == 0)
    {
        try
        {
            _sysName = (java.net.InetAddress.getLocalHost()).getHostName();
        }
        catch(java.net.UnknownHostException e)
        {
            _sysName = "";
        }
    }
    return this._sysName;
}
public synchronized SnmpTimeTicks getSysUpTime()
{
```

```
    return new SnmpTimeTicks( SnmpBaseAgent.getSysUpTime() );
}
```

➤ Step 4: Add *tcpConnTable* implementation

In the *TcpConnTable* class constructor, we want to get current TCP connections and add each of them as a row in the table. There is no way to get current TCP connections with pure Java code, so we make use of the “netstat” shell command on Windows and parse its output to retrieve information on current connections. The parsing code is in the *Util.java* class. Here we only show how to create rows; you can refer to the complete source code in the MIB-II example for more details.

```
public TcpConnTable (OIDTreeNode root, String oid, Object[] args)
{
    super(root, oid);

    //Make use of “netstat” command to get connections, results are stored in an ArrayList object
    ArrayList conns = Util.netstat();

    for (int i = 0; i < conns.size(); i++)
    {
        Connection conn = (Connection) conns.get(i);

        //Create a row based on the connection information
        TcpConnEntry entry = new TcpConnEntry(this, conn.state, new SnmplpAddress(conn.srclp),
            conn.srcPort, new SnmplpAddress(conn.destlp), conn.destPort);

        //Add row to table
        addRow(entry);
    }
}
```

➤ Step 5 (Optional) : Send traps/informs

The super-class of the agent (*SnmpBaseAgent* class) provides static methods for sending SNMPv1/v2c/v3 traps. Traps are sent to all the trap sinks specified in the config file.

In this example, we send a coldStart trap when the agent starts up.

In the agent class, add:

```
public static void main(String[] args)
{
    try
    {
        ...
        Agent agent = new Agent(configFile);

        //send SNMPv2 coldStart trap to all trapsinks
        sendV2ColdStartTrap();
    }
    ...
}
```

➤ Step 6: Modify the config file

MibGen creates a default config file for you, Which allows you to set the version number, community names, trap sinks, and much more. Please refer to the “Configuration” section of this manual for details on modifying the config file.

More details can also be found in the “mib2” example.

## Example 2: Build a Simple Master Agent

Building an SNMP master agent is quite similar to building a standard agent. During step 1, we choose “Master and subagent” instead of “Standard agent” in the “Agent Type” field of the MibGen GUI. The generated agent class will be an extension of *SnmAgentX* rather than *SnmBaseAgent*. The remaining steps are almost the same.

The master agent needs to start a TCP server socket on the port specified in the config file so that subagents can connect to it. Here is an example:

In Agent class:

```
public static void main(String[] args)
{
    try
    {
        ...
        Agent agent = new Agent(configFile);

        //Starts master agent, using port number specified in config file
        agent.startMasterAgent();
    }
    ...
}
```

Please examine the “master” example for more details. In that example, IF-MIB is used as the MIB.

## Example 3: Build A Simple Subagent

Building a subagent is quite similar to building a standard agent. During step 1, we choose “Subagent Only” instead of “Standard agent” in the “Agent Type” field of the MibGen GUI. The generated agent class will be an extension of *SnmAgentX* rather than *SnmBaseAgent*. After starting up, the subagent needs to connect to the master agent and register its MIB subtrees.

The following code shows how to connect to the master agent and register subtrees. More details can be found in the subagent class example.

```
//connect to master agent
SubAgentSession session = agent.connect(host, port);

//To register OID subtrees ...

//Register InfoGroup subtree
int ret1 = session.register(".1.3.6.1.4.1.15145.1.1.1"); //InfoGroup
//Register Subagent table
int ret2 = session.register(".1.3.6.1.4.1.15145.1.1.2"); //SubagentTable
```

The subagent does not usually send traps directly to the SNMP manager. Instead, it forwards traps to the master agent, which then sends them to the SNMP manager. So *sendV1Trap*, *sendV2Trap*, and other similar methods of *SnmplibBaseAgent* cannot be used. The subagent should instead call the *SubagentSession.sendTrap* method to send traps to the master agent.

The “subagent” example provides additional details, using *Subagent.mib* as its MIB file.

## Advanced Example: Build a Multihomed Agent Using Master/Subagent Architecture

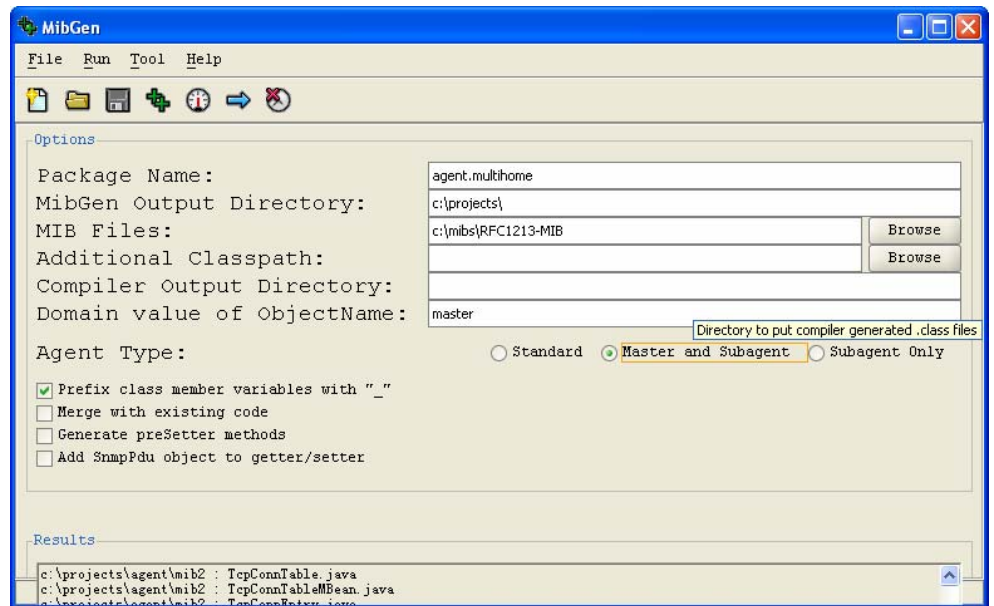
The goal is to build an SNMP agent with the following requirements:

The machine has three IP addresses: A, B and C. If the SNMP manager queries *ifTable* against address A, the SNMP response should only contain network interface information on address A. If the SNMP manager queries *ifTable* against address B, the SNMP response should only contain network interface information on address B. If, on the other hand, the SNMP manager queries *ifTable* against address C then the SNMP response should contain network interface information for both A and B.

Our solution is to build a master agent bound to address C and two subagents bound to addresses A and B. All three agents can reside in one JVM. When the program starts up the master agent will be started first, followed by the two subagents which connect to the master agent and register their own *ifTable* rows.

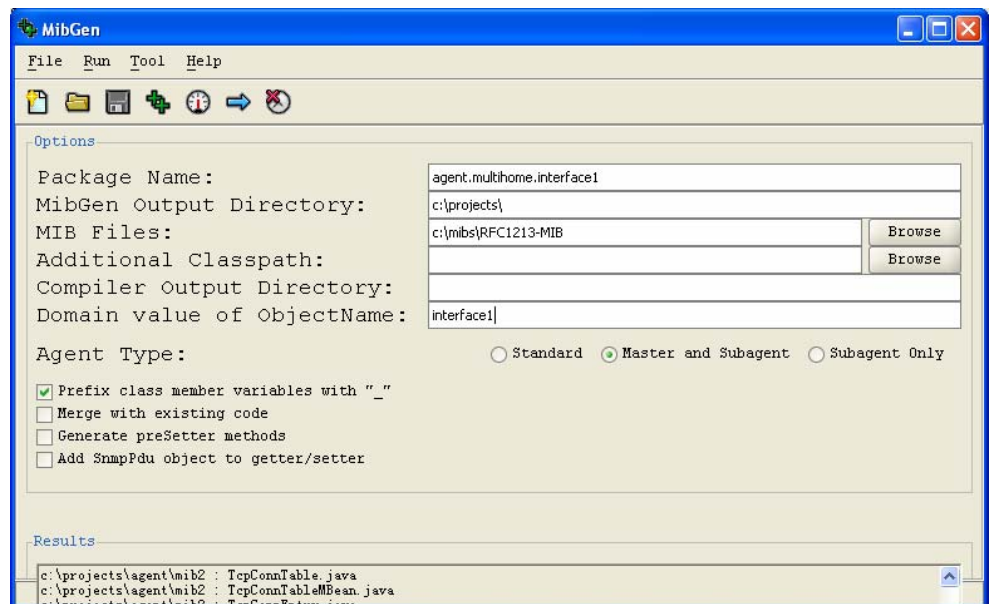
The complete code for this example is located under `examples/agent/multihome`. Here we describe the steps required to build this agent.

1. Use MibGen to generate the master agent code.



The domain name used here is “master”, but you can pick a different name. The only restriction is that each of the three agents must have a unique domain name.

2. Generate code for the two subagents.



Although they serve as subagents for the master agent, they must also be able to process SNMP requests from the manager directly. We therefore still choose “Master and Subagent” for the “Agent Type”.

The domain names given to the two subagents are “interface1” and “interface2”.

3. In the *agent.multihome.AgentMib* class, comment out all unnecessary *registerXXX()* methods in the *registerMBeans* method, and all *unregisterXXX()* methods in the *unregisterMBeans* method. The master agent does not have its own OID tree; all OID trees will be from the subagents.
4. To avoid name conflicts, rename the *agent.multihome.interface1.Agent* class to *If1Agent* and *agent.multihome.interface2.Agent* to *If2Agent*. Their constructors will need to be changed as well.
5. Add table rows to both subagents by modifying the *agent.multihome.interface1.IfTable* and *agent.multihome.interface2.IfTable* classes.
6. Add the following implementation code to the main method:

```
public static void main(String[] args)
{
    ...
    String configFile = "config\\MultiHomeSnmpAgent.xml";
    Agent agent = new Agent(configFile);
    agent.startMasterAgent();
    //Now start two subagents and connect them to master agent
    If1Agent if1agent = new If1Agent ("config\\If1SnmpAgent.xml");
    SubAgentSession session1 = if1agent.connect("localhost", 705);
    //register first row of ifTable
    session1.register(OIDTree.IFENTRY + ".1.1" , 10, 22, 30);

    If2Agent if2agent = new If2Agent ("config\\If2SnmpAgent.xml");
    SubAgentSession session2 = if2agent.connect("localhost", 705);
    //register second row of ifTable
    session2.register(OIDTree.IFENTRY + ".1.2" , 10, 22, 30);
    ...
}
```

7. Modify the config files. The *ipAddress* field value in the master agent’s config file should be address C. The *ipAddress* field values of the two subagent config files should be addresses A and B.
8. Run the master agent. Now you can issue SNMP queries against addresses A, B and C, and see the results!

## Configuration

Config files are located in the `./config` directory by default. You can, however, add a Java environment variable to set a new config directory. For example,

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

will make “`d:\agent\config`” the new config directory.

### ➤ Logger configuration

The logger is configurable through the *Logger.prop* file located in the `./config` directory. You can change the logging level, or just disable the logger altogether. The output stream of logging messages is also configurable to either standard out or a named file. *Logger.prop* has detailed information on the available options.

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

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

If you are using log4j, the file *Logger.prop* will be ignored. You will need to either configure the log4j logger manually or provide it with a new config file. The logging methods from `Logger` can still be used, as they will be delegated to the corresponding methods from log4j.

➤ **Agent configuration**

Agent configuration is normally specified in an XML file called *SnmAgent.xml*, but configuration settings can also be stored in other sources such as databases. (Please see examples/agent/[DbConfig.java](#) for an example of storing settings in a database.) Such files should be put in the ./config directory, the current working directory, or in the /resources directory of the agent's jar file. Alternatively, you can add an environment variable to tell the agent where to find its config file.

The *SnmAgentConfig* class is provided to manipulate the config file programmatically. It provides methods to get/set/save values and methods to add and delete trap sinks.

➤ **properties section**

The properties node defines some agent properties and initial values for MIB variables. If a MIB object's initial value is already specified in the config file, its corresponding getter and setter methods in MBean's implementation class will not take effect.

<b><i>Version</i></b>	The version number of this agent: 1 for SNMPv1, 2 for SNMPv2c, or 3 for SNMPv3
<b><i>encryptPassword-AndCommunity</i></b>	If set to “yes”, all the unencrypted communities and passwords are changed to an encrypted format and saved to a file when the agent starts. Changes can be made by replacing entries in the file with a new, unencrypted value which will automatically be encrypted again when the agent restarts. If set to “no”, values will not be changed even if they're in encrypted format (but they will still be decrypted by the agent). The length of communities and passwords must be less than 32 characters in order to be successfully encrypted.
<b><i>readCommunity</i></b>	A comma-separated list of community names for the SNMP GET/ GETNEXT/ GETBULK operations. If this value is empty, no READ actions are allowed. All community names are case insensitive.
<b><i>writeCommunity</i></b>	A comma-separated list of community names for the SNMP SET operation. If this value is empty, no SET action is allowed. All community names are case insensitive.
<b><i>maxPacketSize</i></b>	The maximum number of bytes in a packet. Note that the maximum ethernet packet size is 1500 bytes.
<b><i>useThreadPool</i></b>	If set to “yes”, a thread pool is used to improve performance. This also means that more system resources will be needed by the agent.
<b><i>engineID</i></b>	The SNMPv3 engineID of this agent. If empty, the first IP address found for the hostname of the machine will be used.
<b><i>engineBoots</i></b>	The number of times the agent will reboot. Its value is updated whenever the agent restarts under SNMPv3.
<b><i>informTimeout</i></b>	The timeout value for the agent to send an INFORM request, in milliseconds
<b><i>informRetries</i></b>	The number of INFORM requests that will be sent after timeout if no response is received.
<b><i>reloadConfigOID</i></b>	The OID for reloading this file, which may be any integer. If the SNMP manager issues a SET request using only this OID, it forces the agent to reload the config file.
<b><i>port</i></b>	The SNMP agent UDP port number. Its default value is 161.

<i>ipAddress</i>	The IP address of this agent, which is used as one of the fields in SNMPv1 trap. The local IP address will be used if this field is empty.
<i>masterAgentPort</i>	This is the master agent TCP port number.
<i>remoteMasterAgent-Port</i>	For subagents, this property specifies the remote master agent's listening port.
<i>remoteMasterAgent-IpAddress</i>	For subagents, this property specifies the remote master agent's IP address or host name.
<i>subagentIpAddresses*</i>	A comma-separated list of authorized subagent IP addresses. An empty value here means that all IP addresses are accepted.
<i>managerIpAddresses*</i>	A comma-separated list of authorized SNMP manager IP addresses. An empty value here means that all IP addresses are accepted.
<i>allowNonV3ReadRequestsForV3Agent</i>	If set to “yes”, an SNMPv3 agent will process SNMPv1/v2c non-set requests (GET/GET_NEXT, GET_BULK). The default value is “no”.
<i>allowNonV3SetRequestsForV3Agent</i>	If set to “yes”, an SNMPv3 agent will process SNMPv1/v2c SET requests. The default value is “no”.
<i>noCommunityNameCheck</i>	If set to “yes”, the agent will ignore community names in checking permissions. The default value is “no”.
<i>communityViewEnabled</i>	If set to “yes”, the community name is tied to views. The default value is “no”. This option only takes effect if the agent is from a version prior to SNMPv3.
<i>system.sysObjectID</i>	This value is used as the Enterprise OID in an SNMPv1 trap.
<i>system.sysLocation</i> <i>system.sysContact</i> <i>system.sysObjectID etc.</i>	These are static values for MIB objects. Use the format “ <i>GroupName.ScalarObject</i> ”. <b>Their values will be updated and saved if changed by SNMP SET.</b> For example, if the manager issues a SNMP SET request and changes <i>sysContact</i> successfully, then the <i>sysContact</i> value in the config file will also be updated and saved.  <b>These values will overwrite corresponding methods in java classes.</b> For example, if <i>system.sysLocation</i> 's value is “building1”, then the <i>getSysLocation()</i> method in the <i>SystemGroup</i> class will be ignored; <i>sysLocation</i> will always have the value “building1”. For that method to take effect, <i>system.sysLocation</i> needs to be deleted from the config file.

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

The agent's config file can be reloaded at run time. The SNMP manager sends a SET request whose OID is the value of "reloadConfigOID", and varbind's value can be any integer. Upon receiving this request, the agent will reload its config file and re-initialize its internal states. For example, if *reloadConfigOID*'s value is ".1.3.6.1.2.1.5000.1.0", the SNMP manager can force the agent to reload config with the following request.

```
java snmpset localhost .1.3.6.1.2.1.5000.1.0 i 1
```

To disable this feature, just leave *reloadConfigOID* blank in the config file.

\* *managerIpAddresses* and *subagentIpAddresses* are comma-separated lists of hosts, each of which can be expressed in any of the following formats:

- A numeric IP address, such as 192.168.2.20,
- A host name, such as server.somewhere.com,
- or CIDR format subnet notation, such as A.B.C.D/16. For example,

192.168.1.0/24 includes all addresses between 192.168.1.0 and 192.168.1.255,

192.168.1.0/25 includes all addresses between 192.168.1.0 and 192.168.1.127,

and 192.168.1.128/25 includes all addresses between 192.168.1.128 and 192.168.1.255.

➤ **trapSink section**

This section defines the properties of trap receivers. An agent will send traps to all defined trap receivers.

▪ **SNMPv1 and SNMPv2 TrapSink**

<i>Hostname</i>	The host name or IP address of the trap receiver
<i>Port</i>	The port number of the trap receiver
<i>Community</i>	The trap receiver's community name
<i>Version</i>	The trap receiver's SNMP version number
<i>isInform</i>	Set this parameter to "yes" to send SNMP INFORM request instead of traps. INFORM requests are more reliable than traps.

▪ **snmpV3TrapSink**

<i>Hostname</i>	The host name or IP address of the trap receiver
<i>Port</i>	The port number of the trap receiver
<i>isInform</i>	Set this parameter to "yes" to send SNMP INFORM requests instead of traps
<i>userName</i>	Set this parameter to one of the user names in the trap receiver's user list
<i>Auth</i>	The authentication algorithm used, either "MD5" or "SHA"
<i>authPassword</i>	An authentication password
<i>Priv</i>	The privacy algorithm used, either "DES" or "AES". The default is "DES".
<i>privPassword</i>	A privacy password

In the `snmpV3TrapSink` section, the security level is determined by `authPassword` and `privPassword`. If both are empty strings, the security level is set to `noAuthNoPriv`. If only `privPassword` is empty, the security level is set to `authNoPriv`. If neither of them is an empty string, the security level is set to `authPriv`.

For example, assume that three trapsinks are defined: one SNMPv1, one SNMPv2, and one SNMPv3. If the agent sends an SNMPv2 trap, it will be sent to all three trapsinks. It will be converted to an SNMPv1 trap before it is sent to the SNMPv1 trapsink. If the agent sends an SNMPv1 trap, it will be converted to an SNMPv2 trap before being sent to the SNMPv2 and SNMPv3 trapsinks.

➤ **user section**

This section defines the properties of authorized users.

<b><i>Name</i></b>	User name. Two users cannot have same name.
<b><i>auth</i></b>	The authentication algorithm used, either “MD5” or “SHA”.
<b><i>authPassword</i></b>	Authentication password.
<b><i>priv</i></b>	The privacy algorithm used, either "DES" or "AES". The default is "DES".
<b><i>privPassword</i></b>	Privacy password.
<b><i>group</i></b>	The group that this user is associated with. A user can only be associated with one group.

Unlike the `snmpV3TrapSink` node, the user’s security level is determined by the security level of its group rather than the values of *authPassword* and *privPassword*. If the security level is `authNoPriv`, then the *privPassword* field will be ignored.

➤ **group section**

This section defines SNMPv3 group properties.

<b><i>name</i></b>	Group name. Two groups cannot have same name.
<b><i>securityLevel</i></b>	The group's security level, which must be one of {noAuthNoPriv, authNoPriv, authPriv}
<b><i>match</i></b>	Context matching method, which must be set to either "prefix" or "exact". (See below)
<b><i>contextPrefix</i></b>	If <i>match</i> is set to "prefix", then context matching only checks whether the context starts with the string <i>contextPrefix</i> . If <i>match</i> is set to "exact", then the context must be exactly matched.
<b><i>readView</i></b>	The view associated with this group for "READ" operations such as GET, GETNEXT, and GETBULK.
<b><i>writeView</i></b>	The view associated with this group for "WRITE" operations such as SET.
<b><i>notifyView</i></b>	The view associated with this group for notification operations.

Note: *SecurityLevel* must be one of the set {noAuthNoPriv, authNoPriv, authPriv}. "noAuthNoPriv" means that neither authentication nor encryption is applied to packets. "authNoPriv" means that only authentication is applied to packets. "authPriv" means that both authentication and encryption are applied to packets. All users in a group have the same security level.

➤ **view section**

This section defines the SNMPv3 VACM view properties.

<b><i>name</i></b>	View name. Multiple views can have the same name, in this case, the user tied to this view name is associated with all of them.
<b><i>type</i></b>	View type, which is set to either “included” or “excluded”. A view subtree can be defined as either including or excluding all the object instances that it contains.
<b><i>subTree</i></b>	Subtree OID. A subtree is a node in the MIB’s naming hierarchy and all of its subordinate elements.
<b><i>mask</i></b>	A list of ones and zeroes separated by '!' or '!'. A view mask can be defined to reduce the amount of configuration information required for fine-grained access control. Each element in this bit mask specifies whether or not the corresponding sub-identifiers must match when determining whether an OBJECT IDENTIFIER is in this family of view subtrees. Thus a '1' indicates that an exact match must occur in that element, and a '0' indicates that any sub-identifier value matches (a 'wild card').

For example,

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

defines a view named view1, which includes all tree nodes whose OIDs start with “.1.3” To take another example,

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

defines a view which includes all tree nodes whose OIDs start with “.1.3.6.1.2”. The last digit of the mask is 0, which means it does not care about the subtree’s OID at that index. Here’s another example:

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

In this view, all OIDs matching .1.3.6.1.2.[1, 2, ...].7.\* are included. The character “\*” represents any series of valid integers separated by periods, and the character “?” represents any single valid OID subelement. Finally, in

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

all OIDs matching .1.3.6.?.2.\* are included, such as “.1.3.6.1.2.1.\*”, “.1.3.6.2.2.1.\*”, and “.1.3.6.3.2.2.\*”.

### ➤ **communityView section**

This section defines the mapping between community names and view names. It takes effect only if *communityViewEnabled* (in the Properties section of the config file) is set to “yes”, and it only applies to SNMPv1/v2c agents.

If a community name is tied to a view name, only the subtree(s) defined by the specified view name will be visible to requests made under the community name. Multiple views can have the same name, in this case, the community name tied to this view name is associated with all of them.

<b><i>community</i></b>	The community name, which must be unique.
<b><i>readView</i></b>	View name for the read community.
<b><i>writeView</i></b>	View name for the write community.

➤ **proxy section**

When this agent serves as a proxy for other agents, this section defines how SNMP requests are delegated.

<b><i>ipAddress</i></b>	The IP address or host name of the agent being proxied.
<b><i>port</i></b>	The listening port number of the agent being proxied.
<b><i>included</i></b>	If set to “yes”, the agent being proxied handles the requests falling under <i>subTree</i> . Otherwise, the agent being proxied handles all requests <u>except</u> for those falling under <i>subTree</i> . Only one proxied agent of the latter type is allowed in the config file.
<b><i>subTree</i></b>	Subtree OID(s). You can specify multiple subtrees, separated by commas.
<b><i>timeOut</i></b>	The timeout value before querying the agent being proxied, in milliseconds.
<b><i>version</i></b>	The SNMP version number of the agent. The possible values are {1, 2, 3}. The default value is 1.
<b><i>readCommunity</i></b>	The community name for SNMP GET / GET_NEXT/ GET_BULK operations of the agent being proxied. This value does not matter if version number is 3.
<b><i>writeCommunity</i></b>	The community name for SNMP SET operations of the agent being proxied. This value does not matter if version number is 3.
<b><i>userName</i></b>	(SNMPv3 property) SNMPv3 user name.
<b><i>auth</i></b>	(SNMPv3 property) The authentication algorithm used, either “MD5” or “SHA”.
<b><i>authPassword</i></b>	(SNMPv3 property) Authentication password.
<b><i>priv</i></b>	(SNMPv3 property) The privacy algorithm used, either "DES" or "AES". The default is "DES".
<b><i>privPassword</i></b>	(SNMPv3 property) Privacy password.

The following example defines an agent being proxied. Its IP address is 127.0.0.1, and it listens on port 200. All the requests falling under subtree 1.3.6.1.2.1 will be handled by this SNMP agent instead of the master agent.

```
<proxy
  ipAddress="127.0.0.1"
  port="200"
  included="yes"
  subTree=".1.3.6.1.2.1"
  timeout="5000"
  version="1"
  readCommunity="yourReadCommunity"
  writeCommunity="yourWriteCommunity"
  userName=" "
  auth=" "
  authPassword=" "
  priv=" "
  privPassword=" "
/>
```

The following proxy section defines a different type of proxied agent. Because *included* is set to “no” this agent handles all SNMP requests except for those with OIDs falling under the subtrees .1.3.1.4.15145.10 and .1.3.1.4.15145.20, which are handled by the master agent itself.

```
<proxy
  ipAddress="127.0.0.1"
  port="200"
  included="no"
  subTree=".1.3.1.4.15145.10,
          .1.3.1.4.15145.20"
  timeout="5000"
  readCommunity="yourReadCommunity"
  writeCommunity="yourWriteCommunity"
/>
```

### ➤ **trapProxy section**

If the master agent also needs to forward traps sent from proxied agents, you can add a trapProxy XML node to the config file. Only one trap proxy can be defined in the config file.

The following example defines a trapProxy node that will start a trap receiver listening on port 192. Traps received will be forwarded to all trap sinks defined in the config file.

```
<trapProxy
  trapForwarderPort="192"
/>
```

## ➤ Agent configuration Example

Here is a sample complete agent configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<SnmpAgent>
  <properties
    version          = "3"
    port             = "161"
    readCommunity    = "public"
    writeCommunity   = "public,private"
    maxPacketSize    = "4000"
    useThreadPool    = "yes"
    engineID         = "12345"
    engineBoots      = "0"
    informTimeout    = "1000"
    informRetries    = "3"
    reloadConfigOID  = ".1.3.6.1.2.1.5000.1.0"
    masterAgentPort  = "705"
    ipAddresses      = ""
    subagentIpAddresses = ""
    managerIpAddresses = ""
    system.sysLocation = "corporate drive"
    system.sysContact = "somebody@somewhere.com"
    system.sysObjectID = ".1.3.1.4.12345"
    system.sysServices = "72"
  />
  <trapSink
    hostname        = "localhost"
    port            = "162"
    community       = "public"
    version         = "2"
    isInform        = "yes"
  />
  <user
    name            = "newUser"
    auth           = "MD5"
    authPassword   = "abc12345"
    privPassword   = "abc12345"
    group          = "aGroup"
  />
  <group name        = "aGroup"
    securityLevel = "authPriv"
    contextPrefix = ""
    match        = "exact"
    readView     = "view1"
    writeView    = "view1"
    notifyView   = "view1"
  />
  <view name      = "view1"
    type         = "included"
    subTree      = ".1.3"
    mask         = ".1.1"
  />
</SnmpAgent>
```

From the properties section, we can tell that this is an SNMPv3 agent listening on port 161. It defines a user called “newUser”, who belongs to the group *aGroup*. *aGroup* maps to *view1*, which allows access to the subtree “.1.3” (meaning that all SNMP OIDs starting with “.1.3” are allowed). All of the users in *aGroup* require the authPriv security level, so authentication and privacy passwords are provided for the user “newUser”. Both passwords are set to “abc12345”. Authentication uses the MD5 algorithm, and encryption uses the “DES” algorithm (by default).

## Example Code

Four agent examples are shipped with this product: MIB-II, IF-MIB, master agent, and subagent. The first two examples demonstrate how to gather static and dynamic system information, handle inter-table indexing, and manage dynamic tables. The master and subagent examples demonstrate the master/subagent architecture.

JAR files for these agents are also provided. To run the examples, type

```
java -jar mib2agent.jar ,
```

```
java -jar ifmibagent.jar ,
```

and so on in the same directory as the .jar files.

To compile the examples, go to the examples directory and run compile.bat (under Windows).

Because some of these agents make use of Windows-specific commands, it is recommended to try them out on Windows NT/2000/XP.

The agent builder generated more than 98% of the code in these examples. To find the manually added code, you can do a search for “New code” in the source directory.

### ➤ MIB2 agent example

The MIB2 example implements the following MIB groups and tables:

```
{system, interfaces, ipAddrTable, ifTable, tcpConnTable}
```

ipAddrTable only gathers static system information. tcpConnTable keeps track of current TCP connection states, so its information needs to be collected whenever the agent receives a new request asking for tcpConnTable values.

### ➤ IF-MIB agent example

The IFMIB example implements the following MIB groups and tables:

```
{interfaces, ifTable, ifXTable, ifRcvAddressTable}
```

ifXTable is an augmenting table of ifTable, meaning that it uses ifTable’s index and there is a one-to-one mapping of rows between the two tables. ifRcvAddressTable has dynamic rows, so you can issue an SNMP SET command to add, delete or modify rows. ifRcvAddressTable also has a dependent expansion relationship with ifTable; its index includes both ifTable’s ifIndex and its own index (ifRcvAddressAddress).

➤ **Master agent example**

The master agent example is almost the same as the IFMIB example, except it also has master agent functionality that can process subagent requests. Its agent class is an extension of the *Snmplib.AgentX* class instead of the *Snmplib.BaseAgent* class.

➤ **Subagent example**

This example can only be used as a subagent, but is otherwise similar to other agent examples. The main difference is that its agent class is an extension of *Snmplib.AgentX* instead of *Snmplib.BaseAgent*. Its MIB file, *Subagent.mib*, is located in the same directory as its source code. It connects to a local master agent and registers two subtrees: 1.3.6.1.4.1.15145.1.1.1 and 1.3.6.1.4.1.15145.1.1.2. A master agent needs to be started locally before running this example.

## FAQ

### **Q. What's the difference between iReasoning SNMP Agent Builder and SNMP API?**

**A.** Agent Builder is a tool for SNMP agent development. SNMP API is for SNMP manager application development.

### **Q. How does iReasoning SNMP Agent Builder differ from other agent builder products?**

**A.** Here are just some of its advantages over the competitors.

- It creates high-performance agents.
- It is the first Java SNMP product to support both DES and strong 128-bit AES encryption algorithms.
- It conforms to RFCs. Many competitors do not really understand RFCs, and have incorrect implementations.
- It dramatically reduces the complexity of agent development. Most complex agent functionalities are already built into the base classes.
- It supports master and subagent architecture based on standard AgentX protocol.
- Its agents have a small footprint compared with other Java-based SNMP agents.

### **Q. What are the key features of Agent Builder?**

**A.**

- Complete support for SNMPv1, v2c, and v3 (USM and VACM).
- It supports master/subagent architecture based on AgentX technology.
- An intuitive GUI tool for automatic generation of Java source code from MIBs.
- It greatly reduces the complexity of agent development. Many tricky SNMP issues are hidden from developers.
- Many optimization techniques are employed to create high-performance agents.
- It conforms to SNMP RFCs.
- Its agents have a small footprint.
- Easy-to-understand, XML-based configuration file format.
- It is re-configurable at run time.

**Q. Do the SNMP security vulnerabilities reported by CERT affect iReasoning Agent Builder?**

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

- VU#107186 - Multiple vulnerabilities in SNMPv1 trap handling. SNMP trap messages are sent from agents to managers. A trap message may indicate a warning or error condition, or simply notify the manager about the agent's state. SNMP managers must properly decode trap messages and process the resulting data. OUSPG found multiple vulnerabilities in the way that many SNMP managers decode and process SNMP trap messages. iReasoning SNMP agent builder successfully passed all 24,100 tests in the OUSPG test suite! We conclude that this advisory does not affect our agent builder.
- VU#854306 - Multiple vulnerabilities in SNMPv1 request handling. SNMP request messages are sent from managers to agents. Request messages might be issued to obtain information from an agent or to instruct the agent to configure the host device. SNMP agents must properly decode request messages and process the resulting data. OUSPG found multiple vulnerabilities in the way that many SNMP agents decode and process SNMP request messages. Agents developed with our agent builder successfully passed all tests in the OUSPG test suite! We conclude that this advisory also does not affect agents built with iReasoning SNMP Agent Builder.

**Q. Which versions of SNMP are supported by iReasoning SNMP Agent Builder?**

**A.** iReasoning SNMP Agent Builder supports SNMPv1, SNMPv2c and SNMPv3 (USM and VACM).

**Q. How's the SNMPv3 support?**

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

**Q. Are the agents compatible with JMX 1.2 spec?**

**A.** Agents are compatible with JMX 1.0, 1.1, and 1.2 specs.

**Q. Can I use other JMX implementations?**

**A.** Yes. We currently use the MX4J JMX implementation in iReasoning SNMP Agent Builder, but it also works with other implementations such as the [SUN JMX reference implementation](#). To switch to other JMX implementations, you just need to place the appropriate jar file before snmpagent.jar in your classpath.

**Q. Can SNMP agent run as a JMX adaptor?**

**A.** iReasoning Agent architecture is based on JMX technology. Agents can run as standalone applications, as SNMP agent services inside an application, or as JMX adaptors. Please refer to the example code [examples/agent/mib2/AgentMX4J.java](#) for details. You can use a web browser connecting to port 8000 to view the currently registered MBeans. (For instance, if your AgentMX4J runs at localhost, point the browser to <http://localhost:8000> )

**Q. Can I stop/restart an agent remotely? Can I change an agent's port number at runtime? Can I change the agent logger's logging level at runtime?**

**A.** Yes to all these questions. First, you need to start a server which makes MBeans remotely accessible. One approach is to start an http adaptor such as HttpAdaptor (included in MX4J) or HtmlAdaptor (included in SUN's JMX RI), so you can use a web browser to remotely manage agents. Check out the [AgentMX4J.java](#) and [JMXAdaptor.java](#) examples for details. Another approach is to start JRMP of MX4J (the [startJRMPAdaptor method in AgentMX4J.java](#)), which lets you use a client such as [MC4J](#) to monitor and control agents via RMI.

**Q. What operating systems does iReasoning SNMP Agent Builder run on?**

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

**Q. What is AgentX technology?**

**A.** Agent eXtensibility (AgentX) is a standard protocol that meets the very real need to dynamically extend managed objects in a node. The protocol allows you to have a single SNMP agent and several subagents, which can connect and register multiple managed objects without interrupting the management service.

AgentX is the first IETF standard-track specification for extensible SNMP agents, and is expected to gradually replace all other agent extensibility solutions, both open and proprietary (e.g. SMUX, DPI, etc.). For more details on AgentX, please refer to [RFC 2741](#).

**Q. Are master/subagents interoperable with SMUX agents?**

**A.** No. Our master/subagents are based on AgentX, which is a newer technology not interoperable with SMUX.

**Q. Are master/subagents interoperable with subagents and master agents from other vendors?**

**A.** Master/subagents built with agent builder should be interoperable with other AgentX-based agents, no matter what language (C, Java, ...) they use. Net-snmp's snmpd (version 5+) has been reported to interoperate successfully with our master/subagents. (Note: NET-SNMP's snmpd is a SNMP daemon written in C, available for the Windows and UNIX platforms)

**Q. I understand that your agent is written in Java, so can it communicate with C/C++ and VB programs?**

**A.** Yes, it is independent of language and platform. It is interoperable with other SNMP managers as long as they conform to the SNMP protocol, and with other master/subagents as long as they conform to the AgentX protocol.

**Q. Can a subagent connect to multiple master agents?**

**A.** Yes, you just need to create a new SubAgentSession for each master agent.

**Q. Can multiple subagents run on one machine?**

**A.** Yes.

**Q. How do I migrate code from version 2.x to 3.x?**

**A.** AgentX support is one of the most important new features of 3.x. If you don't need AgentX support, no code change is required. If you plan to add master agent or subagent support then you just need to make the agent class an extension of *SnmpAgentX* instead of *SnmpBaseAgent*, and add a few lines of code to the *main* method. Check the master and subagent sample code for details.

There are also some minor changes in the SnmpConfig.xml file. If you need to dynamically update the config file, you have to replace "trapd" with "trapSink" and "snmpV3Trapd" with "snmpV3TrapSink".

**Q. How do I migrate code from version 3.x to 4.0?**

**A.** iReasoning Agent Builder 4.0 is compatible with 3.x. There is no need to change your existing code.

**Q. What's the difference between a master agent and a proxy forwarder?**

**A.** An SNMP master agent can route parts of an SNMP request to multiple subagents. The subagents are completely hidden from the SNMP manager, which only sees the master agent. In a proxy forwarding application, however, the manager needs to be aware of all proxied agents and target each request to a single specific subagent. The request also needs to include information that

enables the proxy forwarder application to determine which subagent is the target of the request.

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

A. Yes. Add one line of code:

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

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

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

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

**Q. I want to use the agent config file to make more information persistent. Can I add more entries to the agent config file?**

A. Yes. You can add more entries to the properties, trapSink, and snmpV3TrapSink sections. New entries will be loaded and saved automatically. You can use the *SnmpAgentConfig.getProperty* and *TrapSink.getProperty* methods to retrieve those entries.

**Q. When I ran an SNMP agent built with agent builder on Solaris, I got the error message "BindException: Permission denied". Why?**

A. The default agent port number is 161. You need root privilege on UNIX to run agents on this port. Alternatively, you can modify the port number in the agent config file.

**Q. Can I use log4j for logging instead of your Logger?**

A. Yes, please refer to the Logger javadoc for details and a usage example.

**Q. How can I integrate iReasoning SNMP agent with JBoss?**

A. Check out the Java code and JBoss config file (jboss-service.xml) contained in [jboss.zip](#). Basically, an MBean is created and registered during JBoss startup. An SNMP agent is started in this MBean's *start* method. snmpagent.jar needs to be included in the classpath of JBoss. For example, it can be put in the .../server/default/lib directory. The MBeanServer object ("server") must be passed to the agent's constructor, otherwise a new MBeanServer will be created during agent startup and JBoss will behave differently.

**Q. The passwords are stored in plain text in the config file, and this could pose a security risk if someone were to casually open them up or break into a machine that was the DMZ host and also running an agent. So how can I make passwords encrypted in the config file?**

A. All the passwords and community names can be encrypted. To do this, you need to set *encryptPasswordAndCommunity* to "yes" in the config file. Enter your unencrypted passwords and community names. When the agent starts, all these values will be encrypted and saved to the config file, so that all passwords and community names are secure.

**Q. Does your SNMP Agent Builder support IPv6?**

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

**Q. What is the AES standard, and how does 128-bit AES encryption compare to DES?**

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

*"The Advanced Encryption Standard (AES) is a new Federal Information Processing Standard (FIPS) Publication that will specify a cryptographic algorithm for use by U.S. Government organizations to protect sensitive (unclassified) information. NIST also anticipates that the AES will be widely used on a voluntary basis by organizations, institutions, and individuals outside of the U.S. Government - and outside of the United States - in some cases.*

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

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

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

**Q. How can I store config settings in a database?**

**A.** Config settings can be stored in XML files, databases, and other data sources. You just need to create a subclass of [DefaultAgentConfig.java](#), and implement all the abstract methods. We provide a reference implementation that stores config settings in a database at [DbConfig.java](#). Config settings are stored in 8 tables: properties, trapsink, snmpv3trapsink, proxy, trapproxy, user, v3group, and view. The column names of the tables are the same as those in the XML config file. The data type of all columns is varchar for the sake of simplicity.

**Q. I have a MIB table which can grow to thousands of rows, so it could potentially use up all the memory. How can I handle it?**

**A.** For huge tables, it would consume too much memory for all rows to be stored. First, call `setProcessSnmpRequestDirectly(true)` to make this table handle SNMP requests itself. This table then needs to implement the `getOID` and `getNextOID` methods to pass information on the OID tree to the base class. No corresponding table entry object is created for the rows; each getter

method in the table class fetches data from somewhere else and returns the appropriate results. Please refer to the example code at <examples/agent/mib2/AtTable.java>.

**Q. Do I have to write C/C++ code if I want to build a Windows Extension Agent?**

**A.** No. You only need to write Java code to implement a Java SNMP agent. The *subagent.dll* module delegates SNMP requests from the Windows SNMP service to your Java agent.

## 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/>

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

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

- ❖ JAVA web site

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

- ❖ JMX web site

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

- ❖ AgentX

<http://www.ietf.org/rfc/rfc2741.txt>

# GLOSSARY OF TERMS

---

## JAVA

Java is a programming language expressly designed for use in the distributed Internet environment. 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.

## MBean

A Managed Bean (MBean) is a java object that implements a specific interface and conforms to certain design patterns. It encapsulates attributes and operations through their public methods and follows specific design patterns in exposing them to management applications. JMX defines four types of MBeans: standard, dynamic, open and model.

## MBean Server

An MBean server is a registry for the objects that are exposed to management operations in an agent. Any object registered with the MBean server becomes visible to management applications. The MBean server only exposes an MBean's management interface, however, never its direct object reference.

## 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, and 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 element 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**

A Protocol Data Unit (PDU) is basically a fancy word for a 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 keeps track of the individual data units (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. In connection mode, a transmission may be sent or arrive in the form of packets that must 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 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.

### **SMUX**

SNMP Multiplexer Protocol. One of the open, extensible SNMP agent solutions prior to AgentX.

### **DPI**

SNMP Distributed Protocol Interface. One of the open, extensible SNMP agent solutions prior to AgentX.

### **AgentX**

Agent eXtensibility Protocol was proposed in 1998 as an evolution of DPI. It enhances several technical aspects of DPI and handles different versions of the SNMP protocol. It is the first IETF standard-track specification for extensible SNMP agents, and is expected to gradually replace all other open and proprietary solutions.